Vijay Atluri
Claudia Diaz (Eds.)

LNCS 6879

Computer Security – ESORICS 2011

16th European Symposium on Research in Computer Security
Leuven, Belgium, September 2011
Proceedings

Springer

Lecture Notes in Computer Science 6879

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

Vijay Atluri   Claudia Diaz (Eds.)

Computer Security – ESORICS 2011

16th European Symposium
on Research in Computer Security
Leuven, Belgium, September 12-14, 2011
Proceedings

Springer

Volume Editors

Vijay Atluri
Rutgers University
MSIS Department and CIMIC
1 Washington Park
Newark, NJ 07102, USA
E-mail: atluri@rutgers.edu

Claudia Diaz
K.U. Leuven ESAT/COSIC
Kasteelpark Arenberg 10
3001 Leuven-Heverlee, Belgium
E-mail: claudia.diaz@esat.kuleuven.be

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-23821-5 e-ISBN 978-3-642-23822-2
DOI 10.1007/978-3-642-23822-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011935429

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

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

The European Symposium on Research in Computer Security (ESORICS) has a tradition that goes back two decades. It brings together the international research community in a top-quality event that covers all the areas of computer security, ranging from theory to applications.

ESORICS 2011 was the 16th edition of the event. It was held in Leuven, Belgium, during September 12–14, 2011. The conference received 155 submissions. The papers went through a careful review process in which each paper received at least three independent reviews. The papers were then discussed by the Program Committee to arrive to the final decision. The Program Committee selected 36 papers for the final program, resulting in an acceptance rate of 23%. The authors of accepted papers were requested to revise their papers, based on the comments received. The program was completed with an invited talk by Ross Anderson, from the University of Cambridge.

First and foremost, we would like to thank the members of the Program Committee for their extensive efforts both during the review and the discussion phase. Our task would not have been feasible without their collective knowledge and wisdom. We would also like to express our thanks to the numerous external reviewers for their contributions.

We are indebted to Bart Preneel, the General Chair of this symposium, for his continuous support. Our appreciation goes to Claudio Ardagna for acting as Publicity Chair; and to Pela Noe, Saartje Verheyen, Sebastiaan Indesteege, Roel Peeters, and the rest of the COSIC team for their help with the organization of the event. We are also grateful to Google, Technicolor, and BCRYPT for sponsoring ESORICS 2011.

Finally, we would like to thank the submitters, authors, presenters, and participants who, all together, made ESORICS 2011 a great success.

We hope that the papers in this volume can help you with your research and professional activities, and serve as a source of inspiration during the difficult but fascinating route toward an on-line world with adequate security.

June 2011

Vijay Atluri
Claudia Diaz

Organization

General Chair

Bart Preneel	K.U. Leuven, Belgium

Program Committee Chairs

Vijay Atluri	Rutgers University, USA
Claudia Diaz	K.U. Leuven, Belgium

Publicity Chair

Claudio Ardagna	Università degli Studi di Milano, Italy

Program Committee Members

Mikhail Atallah	Purdue University, USA
Michael Backes	Saarland University and MPI-SWS, Germany
Feng Bao	Institute for Infocomm Research, Singapore
Lujo Bauer	Carnegie Mellon University, USA
Carlo Blundo	Università di Salerno, Italy
Jan Camenisch	IBM Research - Zurich, Switzerland
Srdjan Capkun	ETH Zurich, Switzerland
Véronique Cortier	LORIA-CNRS, France
Jason Crampton	Royal Holloway, University of London, UK
Frédéric Cuppens	IT TELECOM Bretagne,France
George Danezis	Microsoft Research, UK
Sabrina De Capitani di Vimercati	Università degli Studi di Milano, Italy
Roger Dingledine	The Tor Project, USA
Orr Dunkelman	Weizmann Institute, Israel
Simon Foley	University College Cork, Ireland
Dieter Gollmann	Hamburg University of Technology, Germany
Thorsten Holz	Ruhr University Bochum, Germany
Sushil Jajodia	George Mason University, USA
Stefan Katzenbeisser	T.U. Darmstadt, Germany
Angelos Keromytis	Columbia University, USA
Aggelos Kiayias	University of Athens, Greece
Michiharu Kudo	IBM Research - Tokyo, Japan
Klaus Kursawe	University of Nijmegen, The Netherlands
Adam Lee	University of Pittsburgh, USA

Ronald Leenes	Tilburg University / TILT, The Netherlands
Peng Liu	Pennsylvania State University, USA
Javier Lopez	University of Malaga, Spain
David Molnar	Microsoft Research, USA
Steven Murdoch	University of Cambridge, UK
Gregory Neven	IBM Research - Zurich, Switzerland
Radia Perlman	Intel Corporation, USA
Indrakshi Ray	Colorado State University, USA
Ahmad-Reza Sadeghi	Ruhr University Bochum, Germany
Rei Safavi-Naini	University of Calgary, Calgary
Pierangela Samarati	Università degli Studi di Milano, Italy
R. Sekar	Stony Brook University, USA
Basit Shafiq	Rutgers University, USA
Vitaly Shmatikov	University of Texas Austin, USA
Einar Snekkenes	Gjovik University College, Norway
Paul Syverson	Naval Research Laboratory, USA
Patrick Traynor	Georgia Institute of Technology, USA
Carmela Troncoso	K.U. Leuven, Belgium
Jaideep Vaidya	Rutgers University, USA
Will Winsborough	University of Texas at San Antonio, USA

External Reviewers

Zahra Aghazadeh
Hadi Ahmadi
Mohsen Alimomeni
Sami Alsouri
Todd Andel
Elena Andreeva
George Argyros
Md. Shamim Ashik
Elias Athanasopoulos
Ero Balsa
Manuel Barbosa
Lejla Batina
Meriam Ben Ghorbel
Joseph Bonneau
Mike Brennan
Sven Bugiel
Joan Calvet
Sambuddho Chakravarty
Omar-Salim Choudary
Omar Chowdhury
Cheng-Kang Chu
Cas Cremers
Nora Cuppens-Boulahia
Paolo D'Arco
Jun Dai
Angelo De Caro
Emiliano De Cristofaro
Elke De Mulder
Mario Di Raimondo
Maria Dubovitskaya
Nicholas Farnan
Carmen Fernández-Gago
Sara Foresti
Aurelien Francillon
Martin Franz
Keith Frikken
Martin Gagne
Joaquin Garcia-Alfaro
Deepak Garg
Dimitris Geneiatakis
Johannes Hoffmann
Yuan Hong
Ralf Hund
Vincenzo Iovino
Rob Jansen
Kangkook Jee
Ravi Jhawar
Aaron Johnson
Ghassan Karame
Charlie Kaufman
Vasileios P. Kemerlis
Johannes Kinder
Marc Kuehrer
Hoi Le
Fengjun Li
Joseph Liu
Giovanni Livraga
Mohammad Hossein Manshaei
Claudio Marforio
Damon Mccoy
Nicky Mouha
Francisco Moyano
Sascha Müller
Pablo Najera
Kris Narayan

Ana Nieto
David Nuñez
Femi Olumofin
Vasilis Pappas
Andreas Pashalidis
Cristina Perez
Ray Perlner
Michalis Polychronakis
Murillo Pontual
Christina Popper
Georgios Portokalidis
David Ramos
Aanjhan Ranganathan
Indrajit Ray
Joel Reardon
David Rebollo-Monedero
Ruben Rios
Rodrigo Roman
Ahmad-Reza Sadeghi
Nashad Safa
Thomas Schneider
Steffen Schulz
Reza Shokri
Ben Smyth
Angelos Stavrou
Kun Sun
Xiaoyan Sun
Gelareh Taban
Mehdi Talbi
Naoki Tanaka
Nils Tippenhauer
Mathieu Turuani
Sebastian Uellenbeck
Jeffrey Vaughan
Jose Luis Vivas
Christian Wachsmann
Jun Wang
Gaven Watson
Marcel Winandy
Yongdong Wu
Ji Xiang
Xi Xiong
Yanjiang Yang
Eugen Zalinescu
Qiang Zeng
Bin Zhao
Wanying Zhao

Table of Contents

Wireless Security

Web Security I

Web Security II

Forensics, Biometrics and Software Protection

Access Control

Cryptography and Protocol Analysis

Information Flow, Side Channels, and Vulnerability Analysis

Usability, Ttrust, and Economics of Security and Privacy

Privacy I

Privacy II

Privacy III

Secure Localization Using *Dynamic Verifiers*

Nashad A. Safa, Saikat Sarkar, Reihaneh Safavi-Naini, and Majid Ghaderi

Department of Computer Science
University of Calgary
{nasafa,ssarka,rei,mghaderi}@ucalgary.ca

Abstract. We consider secure positioning in wireless environments where mobile nodes use a trusted infrastructure to prove their location: a node claims a position and wants to prove to the verification infrastructure that it is actually located in that position. We propose a system that uses the notion of *dynamic verifiers* and provides security against *collusion attack* in which the adversary corrupts a set of nodes and its aim is to claim a position where none of the corrupted nodes are located. We give a detailed analysis of the system and show that under reasonable assumptions the protocol will reject false claims and the success probability of the adversary can be made arbitrarily small. We also give the results of our simulation that closely match the analysis. Our protocol is the first secure positioning protocol with security against collusion attack.

1 Introduction

Finding location or range (distance from a fixed node) of nodes in wireless environments has been extensively studied [1,2,3,4,11,13]. These information can be used for network services such as routing [2,8,9], as well as for location based services. Location information has also been used for user authentication [7,20] and access control [21].

Positioning and ranging systems in wireless environments use attributes of wireless signals including time-of-flight, angle of arrival and signal strength [5]. In all cases there is a trusted infrastructure that verifies the claims of nodes (position or range). The infrastructure usually consists of a number of trusted verifiers with fixed, known locations, that have out-of-band communication with each other. Positioning and ranging systems however are vulnerable to malicious nodes that report false information. The strongest attack is when the adversary controls a number of nodes to make false claim (e.g. false location or range). This is called *collusion attack*. In [3] it was shown that an adversary always succeeds if the number of colluding nodes is sufficiently large. This impossibility result however is only valid if the location of verifiers is known to the adversary. A theoretically feasible result for this scenario is in *bounded storage* model that assumes that the adversary is restricted in terms of her storage capacity and can not store large messages broadcasted by verifiers.

A practically attractive solution for secure positioning systems is to use *hidden verifiers* [11]: that is to employ verifiers whose locations are not publicly

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 1–20, 2011.

known. This solution however becomes vulnerable if the adversary can adaptively interact with the system over a period of time. To our knowledge there is no positioning protocol that can withstand collusion attack.

Our Work: We consider the problem of secure location verification in the presence of collusion attack. We assume verifiers are connected by out-of-band communication channels and have fixed publicly known locations. Other nodes are mobile and their locations may vary with time. A node can use a positioning protocol to prove a location claim p to this infrastructure. The communication between nodes is wireless and the time-of-flight (TOF) of radio signals is used to calculate distances (the protocol however can be adapted to other localization techniques). We assume any node in the *verification region* can receive the transmission of all verifiers and other nodes. Colluding nodes coordinate their actions by communicating with each other using out-of-band channels and use directional antennas to target a specific receiver. The aim of the adversary is to claim a location that is "far" from all the corrupted nodes. Here "far" is determined by a threshold Δ, which is a system parameter and depends on the accuracy of the positioning system.

The proposed system: We propose a protocol, referred to as *Secure Localization using Dynamic Verifiers (or SLDV for short)* that, with a chosen high probability, will reject false location claims of provers. The protocol is efficient and practical and does not need any unrealistic assumption. Moreover it does not require any additional fixed infrastructure and is particularly applicable to cellular networks, mobile ad hoc networks or vehicular ad hoc networks where system devices have sufficient computing capability. The protocol requires a single public key operation per verification round from the device and so can be easily used for smart-phones, tablets and other similar handheld devices.

We introduce the notion of *dynamic verifiers*: these are verifiers that are 'recruited' in each run of the protocol. The set of dynamic verifiers is a random subset of users that is chosen from the active (connected) nodes in the system, and their role is to help the static verifiers to correctly verify a location claim. The subset is changed in each run of the protocol and so the locations of the dynamic verifiers remain unknown to the adversary. At a high level the protocol between the verifiers and a prover is a basic two phase challenge-response that measures the time-of-flight for the challenge and response, to determine the location. To avoid collusion attack, dynamic verifiers also 'monitor' and collect timing information on the challenge and response and provide it to the static verifiers. Since corrupted and honest nodes are indistinguishable to the verifiers, the set of dynamic verifiers may include corrupted nodes that would provide incorrect information to the verifiers. We will show that as long as sufficiently many dynamic verifiers are honest nodes, the final decision will be correct with high probability.

We give a detailed analysis of the protocol, and calculate the best chance of the adversary in two cases: (i) the adversary does not know the location of the mobile nodes in the system, and (ii) the adversary does know the location of mobile nodes in the system, but does not know which ones are chosen as dynamic verifiers. As long as the majority of nodes in the system are honest, the observations provided by the dynamic verifiers will guarantee a correct decision. In Section B, we give an argument similar to [3], to show that without this assumption it is impossible to provide secure positioning using dynamic verifiers approach.

The rest of the paper is organized as follows. We provide the model and background for our system in Section 3. Section 4 and 5 contain the protocol description and corresponding security analysis. Simulation results and efficiency of the protocol are discussed in Section 6, and Section 7 concluded the paper. An important advantage of dynamic verifier approach is that it does not need infrastructure cost. Correct working of the system requires sufficiently many "good" nodes (roughly more than half of the user population). To overcome this threshold we propose a *hybrid* approach that combines dynamic verifier approach with hidden base station approach [11] which is discussed in Appendix A.

2 Related Work

Protocols for location verification have been studied in the literature from theoretical and practical view points.

In [3], a theoretical framework for location verification was proposed and it was proved that secure positioning against collusion attack is impossible if the locations of the verifiers are public, and this result holds even if verifiers have unbounded computation. The authors then proposed a secure protocol in the Bounded Retrieval Model (BRM), where it was assumed that the verifiers can generate large strings with high minimum entropy that can not be stored by any entity in the system (including adversaries), but can be used by the prover. This work is mainly of theoretical interest.

Secure and practical positioning systems have been well-studied in the field of wireless security. These protocols can be broadly categorized into two groups: one in which the goal is to verify that the prover is within a certain distance from the verifiers, and the second in which a location claim must be verified. Protocols in [14],[12] belong to the first group. The protocol in [14] uses time-of-flight of radio signals while the Echo protocol in [12] uses time-of-flight of ultrasound signals. The protocols in [1,2,4,11,13] consider the problem of verifying location claims. In most cases multilateration (i.e. apply distance bounding strategy from a set of verifiers) based on the time-of-flight of radio signals is used. The 'two tests'-based scheme in [1] can detect false position claims from a single attacker, but is not secure against collusion attack. The system in [22] uses signal intensity of 802.11 card for accurate location determination, but does not consider collusion attack.

The protocols in [2,13] are able to protect against collusion attack but this is achieved using extra assumptions such as the existence of tamper-proof hardware or device fingerprints. The system in [4] provides security against collusion attack with the limitation that the number of colluders is less than the number of fixed verifiers (small number) in the system. In [11], a system with hidden and mobile base stations is considered. For better security, the number of hidden base stations must be high and this incurs significant infrastructure cost. In [3], it is shown that the locations of the hidden base stations can be found by an attacker, assuming that the attacker can run the protocol multiple times and know the result of a failed protocol run. A similar attack is presented in [3] for the case that the base stations are mobile.

In [23] hidden sensors are used to detect fake sensors. Fake sensors are not allowed to exchange messages and so the attack model is weaker than the standard collusion attack. The work of [25] is on geo-localization of wireless sensor nodes where a sensor finds its position from the nodes in its neighbourhood. On the contrary we consider location verification by trusted verifiers. In [24], location is verified by using the feedbacks of beacon nodes. Beacon nodes can be malicious. The authors show an upper bound on the number of malicious nodes that can be tolerated. A possible attack that is not considered by the authors is by positioning malicious nodes close to the honest nodes and so succeeding in false location claims. In comparison to the above previous works, we analyzed security of our protocol when user can provide malicious feedbacks and can position themselves distributively to provide timely responses without being at the claimed location.

3 Model and Background

3.1 Time-of-Flight for Positioning

We will use time-of-flight to determine distance between two nodes. This uses the round trip time of a sent message and the received response, assuming negligible delay in response by the receiver, and the constant speed of the wireless signal during this time. Accuracy of the TOF based schemes depends on the used technology. For example the system in [15] uses ultrasound and can achieve accuracy of 2 cm within the communication range of up to 100 meters. Implementation of [16] (using time-of-flight of radio signals as the underlying mechanism) has an average error of 1.17 m in outdoor environments whereas for indoor environments average error goes up to 2.1 m.

Attacks on the time-of-flight based localization systems aim at changing the timing information of the signals. In extreme cases, the signal is completely blocked (jammed) and the protocol fails. We do not consider these attacks here. In positioning systems with many nodes, the number of nodes that the adversary can corrupt and control (colluding nodes) determines its colluding power. The following Lemma is proved in [3].

Lemma 1. *If the number of colluders scales to the number of verifiers, they can devise a collusion attack to simulate any time-of-flight based location verification protocol to establish a fake position claim in a general setting.*

Readers are referred to [3] for the proof of this lemma.

3.2 Model

There is a set U of users (mobile nodes) that are connected to the system and their locations are registered to the positioning infrastructure. There is a fixed set of trusted static *verifiers*, $V = \{v_1, v_2, \ldots, v_g\}$ with publicly known locations. We refer to these as simply, *verifiers.* Verifiers have a pair (public and private) of encryption keys $((K_e, k_e))$, and a pair of public and private keys $((K_s, k_s))$ for a secure digital signature scheme. The public keys of the encryption and digital signature schemes are assumed to be known by all users. The corresponding secret keys are known only by the verifiers. Verifiers share all key information. Any user can send an encrypted message to the verifiers and can verify a message signed by them. Verifiers may also have out-of-band secure channel among themselves (possibly a wired connection). Data is shared among all the verifiers using this channel.

Verifiers maintain a table that records information about users that have appeared in the system so far. The first successful position verification of a node results in an entry in the table containing information such as pseudonym (Id) of the node, a shared key between the verifier and the node, the registered location of the node and an initialization vector. These information will be updated every time that the node can prove a location.

A *Prover* P_x, is a user who claims a new location p to be verified by V. A prover determines its location, e.g. using a GPS, and then transmits its claimed position p to the verifiers. Verifiers collaboratively execute a protocol with the prover to verify p. During the verification process for a position p, a subset of users, denoted by D_p, is selected, that will serve as the set of *Dynamic Verifiers* for that claim. Time is synchronized among the verifiers and the provers [19,18].

Security: The *Adversary* corrupts and controls one or more nodes with the aim of falsely claiming a position which is not "close" to any of the colluding nodes. Colluders are equipped with omnidirectional and directional antennas, and can use them to broadcast messages or target a particular node. We assume that communications between static verifiers and provers are of broadcast nature.

We assume that majority of nodes in the verification region are honest. We note that the number of static verifiers (e.g. base stations, satellites) that form the positioning infrastructure is in general much less than the number of users in a verification region. For example, in a typical cellular network, a cell-site usually contains from 1 to 3 base stations whereas the number of users may be in the range of hundreds of users.

We say *a location verification protocol Π is (Δ, λ) secure* if a prover at location p' can establish its fake location at p with probability $\leq \lambda$ such that $dist(p, p') > \Delta$. Probability is taken over all random coins of the adversary, and assuming uniform distribution for users in the verification region.

3.3 Notation

We used Π^{SLDV} to denote the localization protocol. DV denotes a dynamic verifier. P_x denotes a new prover who needs to be authenticated. We used the

Table 1. Notations used in Π^{SLDV}

Notation	Description
P_x	New prover
N	Number of users in the system
p_x	Position of user x
V	Set of static (i.e. system) verifiers
v_i	Static verifier i
DV_i	Dynamic verifier i
ID_i, ID_i^r	Id (pseudonym) for user i, and its refreshed form
IV_i	Initialization vector chosen by node i
$k_{V,i}$	Shared symmetric key between V and node i
K_e, k_e	V's Public and private key for public key encryption algorithm.
K_s, k_s	V's Public and private key for digital signature algorithm
$PubE(m, K_e), PubD(c, k_e)$	Public key encryption/decryption of plaintext m using key K_e/k_e.
$SymE(m, k), SymD(c, k)$	Symmetric Encryption/decryption of plaintext m using key k
$Sign(m, k_s), Verify(s, K_s)$	Digital sign/verify of message m using private key k_s/public key K_s
b_{DV_i}, b_{SV_i}	Status bit output of DV_i and v_i
q_+^{DV}, q_-^{DV}	Number of positive/negative verdicts of DV's responses
t_b	Broadcast instance of nonce x
c	Speed of light
$t_{r_j}^{SV}$	Time instance of response received by v_j
$t_{r_i}^{DV}$	Time instance of response received by DV_i

notation $dist(p, q)$ to indicate physical distance between positions p and q. We denote the acceptable error in location measurement by Δ. The notations used in the paper are summarized in Table 1.

4 The Proposed System

Consider a new prover P_x who wants to prove its location p to the system. We assume N users are connected to the network. Upon receiving a location claim message from P_x, verifiers execute a challenge response protocol and measure TOF to determine the location. However before sending the challenge, they increase the number of verifying nodes by selecting a set D_p of $k(\leq N)$ users to play the role of 'dynamic verifiers' in that round. The TOF measurement of static and dynamic verifiers, both will be used by the verifiers to verify the location of P_x. The adversary does not know the location of dynamic verifier nodes and so cannot position the corrupted nodes to control the view of the verifiers. The set D_p changes in every run of the protocol. Moreover the pseudonyms of the nodes, and hence DVs, are refreshed in every run. This ensures that the adversary cannot 'trace' nodes in multiple runs of the protocol. Algorithm 1 describes the main steps of Π^{SLDV}.

4.1 Algorithm Description

Verifiers maintain a user list $UserList$ with one entry for each user that has made a verified claim at one stage in the system. This table is updated after

Algorithm 1. Protocol Π^{SLDV}

```
1. Position claim
   $P_x$ :[1]
     $\rightarrow)$ : $p$ ;   ; if first time prover.
     $\rightarrow)$ : $SymE((p, ID_{P_x}), k_{V,P_x})$ ;   ; if returning prover[2].
2. Verifiers' challenge:
   $v_\ell \in V = \{v_1, v_2, \ldots, v_g\}$; select leader verifier.
   $v_\ell$ :
     $D_p = \{DV_1, DV_2 \ldots DV_k\} \subset UserList$.
     $ch \in_r \{0,1\}^n$;   challenge $ch$ is generated.
     $ID_i^r \leftarrow IDRefresh(ID_i)$, where $IDRefresh(ID_i) := (ID_i \oplus SymE(IV_i, k_{V,i}))$
   $v_\ell \rightarrow D_p$ : $[ID_1^r, \ldots, ID_k^r]$
   $v_\ell \rightarrow)$ : $ch$, $Sign(ch, k_s)$
3. Prover's response
   $P_x$ :
     $k_{V,P_x} \in_r \{0,1\}^{n'}$;   select symmetric key.
     $IV_{P_x} \in_r \{0,1\}^l$;   select initial vector.
     $\rightarrow)$ : $m = (ch, PubE((k_{V,P_x}, IV_{P_x}), K_e))$
4. Dynamic verifiers' response
   $DV_i$, $i = 1, \cdots k$ :
     set $b_i^{DV} = 1$, if $ch$ in response $m$ matches with challenge; $b_i^{DV} = 0$ otherwise.
     record time instance of receiving $m$ ($t_{r_i}^{DV}$) and current position $p_{DV_i}$.
     $\rightarrow v_\ell$ : $SymE((t_{r_i}^{DV}, p_{DV_i}, b_i^{DV}, ID_i^r), k_{V,DV_i})$.
5. Response validation and decision
   $v_\ell$:
   (i) Generate a vote for SV: increment $q_+^{SV}$ if $VerfyRp(t_b, t_{r_i}^{SV}, p, p_{v_i})$ returns
       True ; increment $q_-^{SV}$, if False.
   (ii) Generate a vote for DV: increment $q_+^{DV}$ if $VerfyRp(t_b, t_{r_i}^{DV}, p, p_{DV_i})$ returns
       True and $b_i^{DV}$=1 ; increment $q_-^{DV}$, if False.
   (iii) Accept $p$ if $q_+^{DV} > q_-^{DV}$ and $q_-^{SV} < \theta$; reject otherwise.
   (iv) Update user table: if first time prover: add new entry and send
       $SymE(ID_{P_x}, k_{V,P_x})$ ; if returning: update location.
```

each run of the protocol. The first successful verification of a user (P_x) results in an entry in $UserList$ that includes (i) current position of the user, (ii) a shared key k_{V,P_x}, (iii) an initial vector IV that will be used in symmetric cipher in a chaining mode and (iv) a pseudonym ID. k_{V,P_x} and IV are chosen by the prover and sent to V in encrypted form (public key encryption). ID is generated by V and is sent to the node in encrypted form (symmetric key encryption). ID is used as the user's pseudonym and is refreshed in every usage. The update of ID is done simultaneously by V and the corresponding user using IV for the symmetric key algorithm in a chaining mode. A returning prover with a new rejected claim will fail to establish its new position to verifiers.

[1] '$a : s$' indicates that node a executes statement s

[2] '$a \rightarrow) : m$' is used to indicate sending of a broadcast message m by node a. Similarly '$a \rightarrow b : m$' represents sending of message m from node a to node b.

Update(UserList) algorithm is run after each verification session. Let $V = \{v_1, v_2, \ldots, v_g\}$ denote the set of verifiers.

P_x Claims Location p. A user P_x wants to claim a location p. If this is the first time claiming a location, P_x simply broadcasts p to verifier set V. If it is a returning prover with ID_{P_x} and k_{V,P_x}, it also sends its ID_{P_x} along with p in encrypted form.

Verifiers' Challenge. Verifiers select a lead verifier v_ℓ who coordinates the communication and decision of that protocol run. v_ℓ randomly selects a subset of k users, denoted by D_p, from $UserList$ as dynamic verifiers for that round. v_ℓ refreshes the IDs of the selected subset using $IDRefresh(ID)$; selects a random challenge ch; and broadcasts the list of refreshed IDs and then the signed challenge.

Prover's Response. P_x receives the challenge and verifies the signature. If this is the first time verification (i.e. a new prover), it selects a random key k_{V,P_x} and an initial vector IV_{P_x}; encrypts the key and the IV using the public key of the verifiers[3] and broadcast the message $m = (ch, PubE((k_{V,P_x}, IV_{P_x}), K_e))$. If a returning prover, P_x only broadcasts $m = ch$.

Dynamic Verifiers' Response. A dynamic verifier DV_i is a user who has been verified before and shares a symmetric key k_{V,DV_i} and an IV_{DV_i} with the verifiers. DV_i maintains its current ID in synchrony with the verifiers, and so by monitoring verifiers' broadcast, will be able to determine if it has been selected as a dynamic verifier. It verifies the signature of challenge ch and records the time of receiving P_x's response, denoted by $t^{DV}_{r_i}$. It outputs a bit b_{DV_i} that is 1 if the replayed nonce by P_x is equal to the challenge nonce (ch), and zero otherwise. Each DV_i also determines its position p_{DV_i} using the GPS functionality.

Response of DV_i (i.e. $p_{DV_i}, t^{DV}_{r_i}, b_{DV_i}, ID^r_i$) is encrypted using the shared key with the verifiers using the symmetric key algorithm (used in a secure mode).

Response Validation and Decision. v_ℓ receives a vote $b_{SV_i}, i = 1 \ldots g$ from each static verifier. b_{SV_i} is generated using the time that v_ℓ broadcasts ch (t_b) and v_i receives P_x's response ($t^{SV}_{r_i}$), using algorithm $VerfyRp(t_b, t^{SV}_{r_i}, p, p_{v_i})$. These votes are securely sent to v_ℓ (out-of-band communication, or public key encryption).

v_ℓ also receives $(p_{DV_i}, t^{DV}_{r_i}, b_{DV_i}, ID^r_i), i = 1 \ldots k$. If b_{DV_i} is set to 1, v_ℓ uses algorithm $VerfyRp(t_b, t^{DV}_{r_i}, p, p_{DV_i})$ to determine if P_x's response was within the expected time for the respective DV_i. If not, vote of this DV_i is considered negative directly. The claim is accepted if for the majority of DVs, the response is valid, and less than a threshold θ of static verifiers reject the claim. The threshold θ is chosen according to quality of reception in the verification region and the required level of assurance.

[3] We comment that encryption should be completed before receiving the challenge to reduce processing delay.

Algorithm 2. Update($UserList$)

```
if position claim is accepted then
    if $P_x$ is a new prover then
        $v_\ell : Data_{P_x} \leftarrow \{IV_{P_x}, p, k^{V,P_x}\}$
        $UserList[ID_{P_x}] \leftarrow Data_{P_x}$
    else
        Update position of $P_x$
        in $UserList[ID_{P_x}]$ to $p$
    end if
end if
for every dynamic verifier $DV_i$ do
    Increase $IV_i$ in $UserList[ID_i]$ by 1
end for
```

Algorithm 3. $VerfyRp(t_1, t_2, p_1, p_2)$

```
if $|(t_2 - t_1)c - (dist(p_{v_\ell}, p_1) + dist(p_1, p_2))| \leq \Delta$ then
    return True
else
    return False
end if
```

List Update. The next steps of the protocol relate to update $UserList$ after a verification scenario and are described in Algorithm 2. Let us assume $Data_{P_x}$ represents the collection of attributes (IV_{P_x}, p, k_{V,P_x}) associated with an accepted prover P_x. If P_x is verified for the first time, its attributes are included in $UserList$ table indexed by ID_{P_x}. Otherwise, its position in $UserList$ is updated to p. Regardless of acceptance or rejection, IV_i of each DV_i participated in this verification session are increased (or updated).

4.2 Reliability of Verification Decisions

Verifiers randomly select k users as dynamic verifiers from the set of all registered users in the network. We use a parameter h_p to show the level of trust that can be put on a randomly selected node and so to its observation. For k randomly selected users, the probability that at least $k/2$ users are dishonest is given by $\sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h_p)^i h_p^{k-i}]$ which is $\mathcal{O}(e^{-2(k-1)(h_p-0.5)^2})$ for $h_p \geq 0.5$. We can denote this probability as P_{dh}. The following Lemma shows that, if $h_p \leq 0.5$, it is impossible to provide a secure localization protocol in the general setting (aka 'vanilla model'). The proof of the lemma is given in Appendix B.

Lemma 2. *If the number of colluding nodes is at least half of the total number of users and verifiers (i.e.$(N+g)/2$), the dynamic verifier system will loose its security. That is colluding nodes can formulate a collusion attack that guarantees success to establish a fake position claim in a general setting.*

5 Security Analysis

An adversary will be successful if a location p can be successfully verified without any colluding node being present at that location (within accuracy parameter Δ). The best success chance of the adversary is when it initiates a verification session rather than tampering with an existing session. In this case the adversary completely controls the initial location claim and the prover's response to the verifiers' challenge, and will succeed if it can modify (or dominate) the vote of sufficiently many dynamic verifiers. The barriers for this success are two: the location of dynamic verifiers are unknown and so correct timing information cannot be generated, and DVs use secure encryption algorithms to construct their messages in a format that is expected by the verifiers.

We first discuss the cryptographic protection offered by the system, and then analyze the success probability of the adversary in providing correct timing information. The set of DVs is chosen independently for each verification round and so the success chance of finding the DV set is the same as random guessing. Note that refreshing the IDs of DVs in each verification round ensures that long term monitoring of the network cannot be used for tracing users and better guessing the location of DVs. (If IDs are fixed, a very powerful adversary may be able to activate its corrupted nodes that are right next to the selected DVs of a verification round, immediately after seeing the broadcasted IDs.) To construct the response of a DV, the adversary must know the secret key shared by the DV and the verifiers. This is because the messages sent by DVs contain fresh information for the current round (including the refreshed ID of the node) and so can not be reused or correctly guessed which will have a small success chance. Finally one can consider a man-in-the-middle attack where the adversary intercepts m from an honest prover and replaces it with her own chosen key, which needs to be performed within expected response-time for most DV_i. This requires guessing D_p or their locations and adjusting the instance of sending the response accordingly.

So, the viable option remains for the adversary is to control the timing of sending the response message by anticipating the locations of DV or the set D_p itself (when the adversary is able to track the current location of all users). The latter case can arise if the users are not sufficiently mobile and thus a persistent adversary can find all of their locations by observing communications to each other (if allowed in the system) and to dynamic verifiers. In the following we will assume that a colluding node will succeed if it is well positioned. That is we do not use the added layer of security provided by the cryptographic primitives used in the system. In other words we assume that corrupted nodes can construct well formed responses and their success chance is only limited by the unknown locations of DVs.

5.1 Security against Collusion Attacks

Theorem 1 states the security of Π^{SLDV} against collusion attack. Let h_p denote the probability that a randomly selected user be honest and $h'_p = 2h_p - 1$. We assume $2r$ is the diameter (largest dimension) of the verification region, and N denotes the total number of users in the system.

Theorem 1. *Π^{SLDV} is (Δ, λ) secure against collusion attack provided the number of corrupted nodes in the system is less than the number of honest nodes. The value of λ is as follows:*

1. *$\lambda = 1 - P_{SLDV} = P_{dh} + (1-P_{dh})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^{k/2}$, when users locations are unknown to adversaries;*
2. *$\lambda = 1 - P'_{SLDV} = P_{dhm} + (1-P_{dhm})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^{k/2}$, when users locations are available to adversaries.*

Here, $P_{dh} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h_p)^i h_p^{k-i}]$, and $P_{dhm} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h'_p)^i h_p'^{k-i}]$.

Location of Users Are Unknown. To deceive a DV_i in D_p (without knowing its location), the distance between a colluding node and DV_i must be the same as the corresponding distance between the claimed location and DV_i. The adversary succeeds if this is the case for majority of users in D_p. As DV are assumed to be randomly distributed in the network space, the adversary's only strategy would be to guess the approximate locations of D_p and try to adjust the instance of transmitting m based on this guess. The following lemma gives the security of Π^{SLDV} against collusion attack when users are in unknown positions with respect to the adversary.

Lemma 3. *If $N \geq k$ mobile users are available in the system, Π^{SLDV} can detect collusion attack in a two dimensional region with probability, $P_{SLDV} = 1 - (P_{dh} + (1-P_{dh})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^{k/2})$, where $P_{dh} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h_p)^i h_p^{k-i}]$ and $\Delta << r$.*

Proof: For simplicity we assume positioning of devices takes place in a circular region of radius r. Here we consider both two-dimensional (2D) and three-dimensional (3D) localization scenarios. Suppose an adversary chooses its position A_p uniformly over the region of radius r and the distance between a DV_i and the claimed prover's location is s. We first determine the success probability in the two-dimensional case. A similar argument can be used for three-dimensional case.

First, we compute the probability that the distance between a single adversarial node and a single DV_i is the same as the distance between the claimed location and that DV_i. Then we will extend the result for multiple colluding nodes and dynamic verifiers. Using the geometric approach of [17], the probability distribution function (pdf) of adversary's distance d_A from the randomly distributed location of a dynamic verifier can be expressed by:

$$Pr_{2D}[d_A = s] = \frac{4s}{\pi r^2}\cos^{-1}(\frac{s}{2r}) - \frac{2s^2}{\pi r^3}\sqrt{1 - \frac{s^2}{4r^2}}$$

And for $3D$,

$$Pr_{3D}[d_A = s] = \frac{3s^2}{r^3} - \frac{9s^3}{4r^4} + \frac{3s^5}{16r^6}$$

Plotting these functions for different values of $s, (0 \leq s \leq 2r)$ establishes that for $s = 0.84r$, $Pr_{2D}[d_A = s]$ achieves its maximum value (which is $0.809/r$) whereas for $s = 1.05r$, $Pr_{3D}[d_A = s]$ achieves its maximum value $0.942/r$. The maximum probability value decreases (which is always achieved at the same s

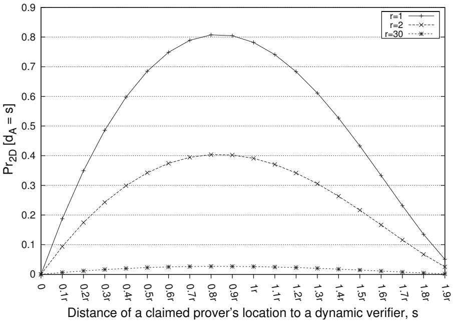


Fig. 1. $Pr_{2D}(d_A = s)$ vs r

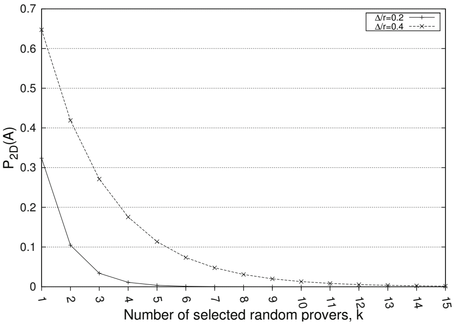


Fig. 2. Value of $Pr_{2D}(A, k)$ As k Increases

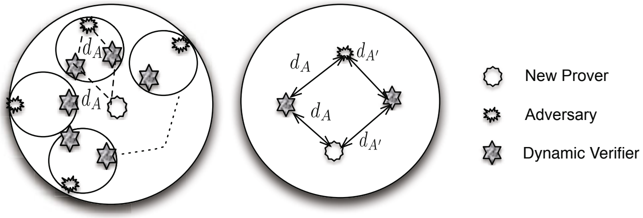


Fig. 3. Colluding Adversarial Scenario

value) as the radius of the network r increases. The relation is depicted in Fig 1 for the two dimensional case. This means that a malicious node has the highest probability of success when it guesses the distance of a random DV_i from its position as $s = 0.84r$ (in 2D case). As we are also allowing an error range Δ to be accepted (to compensate for processing and transmission delay), average success probability of an attacker for a network dimension of radius r can be computed as follows [11]:

$$Pr_{2D}(A) = \int_{0.84r-\Delta}^{0.84r+\Delta} Pr_{2D}[d_A = s] \approx \frac{0.809}{r} \times 2\Delta$$

$$Pr_{3D}(A) = \int_{1.05r-\Delta}^{1.05r+\Delta} Pr_{3D}[d_A = s] \approx \frac{0.942}{r} \times 2\Delta$$

Here, Δ has been considered with respect to radius r and $0 < \Delta < 0.5r$. The above success probabilities have been calculated considering a single DV_i and a lone adversary. We note that, as the radius of the verification region r increases, this probability decreases drastically. In practice, we expect r to be a large value. In Π^{SLDV}, verifiers select k independent random users in D_p for the purpose of response verification. For calculating probability of a successful attack which involves k random DV against a set of colluding nodes, we identify the following two cases (see Fig 3):

(1) A single adversarial node tries to deceive the k DV based on conjectures. In this case the probability of a successful attack would be the probability that distances between every DV_i in D_p and this sole adversary are same as the corresponding distances between these DV_i and the claimed location. So, the average success probability of attack in this scenario would become:

$$Pr_{2D}(A,k) = [Pr_{2D}(A)]^k \approx (\frac{0.809}{r} \times 2\Delta)^k \tag{1}$$

$$Pr_{3D}(A,k) = [Pr_{3D}(A)]^k \approx (\frac{0.942}{r} \times 2\Delta)^k \tag{2}$$

For reasonable small Δ and moderate values of k, success probability of adversarial spoofing of a fake position becomes negligible. Fig 2 shows how $Pr_{2D}(A,k)$ value decreases when we increase k for two different values of Δ.

(2) Alternatively we can assume l colluding nodes are collaborating for the deception. Now if a dynamic verifier receives responses from more than one malicious node who is sending response on behalf of the actual prover, it will invalidate the position claim. Thus, from adversarial view point it will be wise to divide the whole validation zone (of area πr^2) into l subregions where each colluder will broadcast the response using directional antennas or alternatively broadcast with low signal power. Members of D_p are selected uniformly over the verification region. The number of dynamic verifiers (assuming they are uniformly distributed and all the subregions possess equal area) in each of these subregions would be k/l. Without loss of generality we can assume that each subregion is a circle of radius r' where $r' \approx r/\sqrt{l}$ and $r' \geq 1$. Hence, the probability that colluder i would be successful to deceive his part of dynamic verifiers (k/l in number) is given by:

$$Pr_{2D}(A_{single}) = [Pr_{2D}(A)]^{k/l} \approx (\frac{0.809}{r'} \times 2\Delta)^{k/l}$$

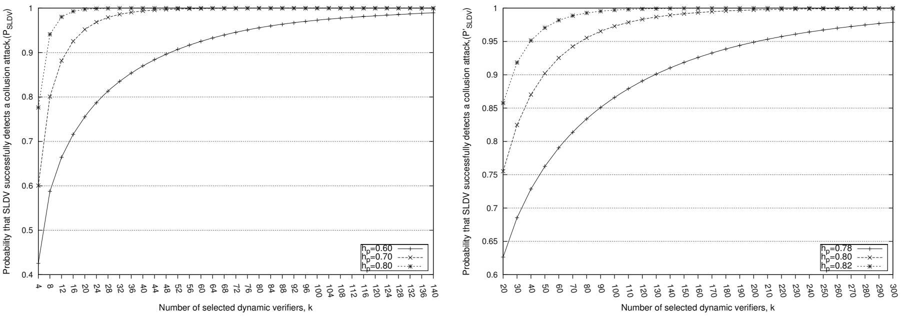
$$Pr_{3D}(A_{single}) = [Pr_{3D}(A)]^{k/l} \approx (\frac{0.942}{r'} \times 2\Delta)^{k/l}$$

Hence the probability that all the l colluders become successful in their attack would be:

$$Pr_{2D}(A_l,k) = [Pr_{2D}(A_{single})]^l \approx (\frac{0.809}{r'} \times 2\Delta)^k \tag{3}$$

$$Pr_{3D}(A_l,k) = [Pr_{3D}(A_{single})]^l \approx (\frac{0.942}{r'} \times 2\Delta)^k \tag{4}$$

Suppose h_p represents the probability with which a random user can be trusted. Now if we select k random users in D_p, the probability that $k/2$ dynamic verifiers or more would be dishonest is $\sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h_p)^i h_p^{k-i}]$. We can denote this probability as P_{dh}. If total number of nodes in the system is N, the number of expected malicious nodes would become $N(1-h_p)$. Therefore, r' in Eq. 3 will be



(a) P_{SLDV} vs k for different h_p when $\varDelta/r = .01$ and $N = 1000$

(b) P'_{SLDV} vs k for different h_p when $\varDelta/r = .01$ and $N = 1000$

Fig. 4. Probability of Detecting Collusion Attack

$r/\sqrt{N(1-h_p)}$. Consequently, $\varPi^{SLDV}$ will be able to detect a collusion attack (in this scenario) with probability P_{SLDV}, where

$$P_{SLDV} = 1 - (P_{dh} + (1 - P_{dh})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\varDelta)^{k/2})$$

In typical networks, we expect the value of h_p to be high. It is evident in Fig. 4 that for reasonable values of k, probability of detecting a collusion attack becomes close to one and we claim that $\varPi^{SLDV}$ is $(\varDelta, 1-P_{SLDV})$ secure against collusion attack under the stated assumptions. This completes the proof of our lemma. □

Location of All Users is Known to the Adversary. The following lemma specifies the security of $\varPi^{SLDV}$ against collusion attack when users are in known positions with respect to the adversary.

Lemma 4. *If $N \geq k$ users are available in the system (whose locations are known to adversaries), $\varPi^{SLDV}$ can detect collusion attack in a two dimensional region with probability, $P'_{SLDV} = 1 - (P_{dhm} + (1 - P_{dhm})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\varDelta)^{k/2})$ where $P_{dhm} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h'_p)^i h'^{k-i}_p]$, $\varDelta << r$ and $h'_p = 2h_p - 1$.*

Proof: With N users connected to the system and h_p trust assumption, the number of expected attackers is, $l = N(1-h_p)$. If the size of set D_p is k, out of the k DV, $k(1-h_p)$ are expected to be corrupted. Now l attackers can choose l of the users randomly, out of total Nh_p honest users, to take appropriate position and perform the distributed time-delayed response attack (as described in Lemma 1). Among these l users, $l\frac{kh_p}{Nh_p}$ of them are expected to be in the set D_p. Consequently, expected number of unmonitored honest DV in set D_p is, $k' = k - k(1-h_p) - N(1-h_p)\frac{k}{N}$.

Let h'_p represents the probability that a random DV in set D_p is both honest and unmonitored. Then, $h'_p = k'/k = 2h_p - 1$. From this expression, it is easy to find that for h'_p to be greater than 0.5 (for reliable decision), we need $h_p \geq 0.75$. Now using the same arguments presented in Lemma 3, it can be derived that probability of detecting a collusion attack by Π^{SLDV} is,

$$P'_{SLDV} = 1 - (P_{dhm} + (1 - P_{dhm})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^{k/2})$$

Where, $P_{dhm} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i} [(1-h'_p)^i h'^{k-i}_p]$ and $h'_p = 2h_p - 1$. □

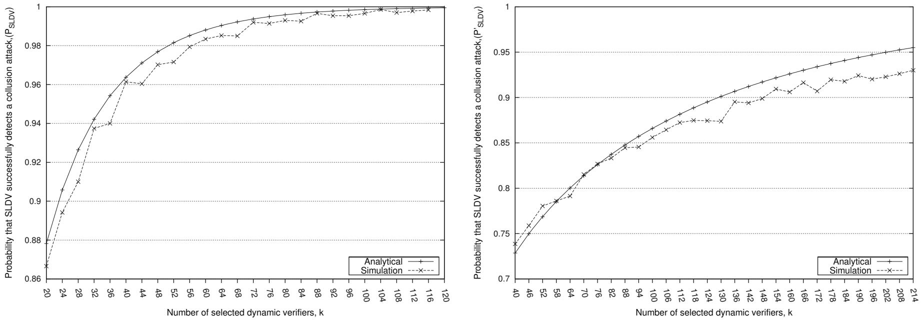
To calculate attacker's success probability in three dimensional regions, we just need to replace 0.809 with 0.942 in both P_{SLDV} and P'_{SLDV}.

We have numerically computed and plotted the probability of collusion attack in Fig 4. In Fig. 4(a) we can see that as long as the users' locations cannot be tracked by the adversary, for reasonable values of k, the probability of detecting a collusion attack, (P_{SLDV}) becomes close to one. In the case of all users' locations are known to the adversary, Fig. 4(b) shows the probability values of detecting a collusion attack (P'_{SLDV}) for different h_p assumptions and $\Delta/r = 0.01$. As expected, when users' locations are compromised to the adversary, we need higher trust assumption and larger number of k to detect a collusion attack with high probability. However, we emphasize that in a typical mobile environment, current locations of very few users will be compromised to the adversary.

6 Simulation Results

We performed simulations to estimate the value of P_{SLDV} for two cases: users' location is unknown, and users' location is known to the adversary. Our simulation program is written in Java and run on an intel 2.66 GHz core 2 duo processor. The number of user nodes, range of co-ordinates for location and trust assumption h_p are given as simulation parameters. To generate random numbers we used *Random* class provided in *java.lang.Math* package. We considered $h_p = 0.65$, i.e. on average 350 of the 1000 users are corrupted in each system run. Co-ordinates of the users are sampled randomly within the given topological range. Users are also randomly selected as honest or corrupted in accordance with the trust assumption h_p.

Let l denote the total number of corrupted nodes and k denotes the number of randomly chosen DVs, t of which are corrupted. For simplicity we assumed that static verifiers are all deceived by the attackers and hence verification decision only depends on the dynamic verifiers. A random position p in the given range is assumed to be the falsely claimed location. We found the number of honest users (denoted by s) among k, such that the Euclidean distance between them and p is approximately equal (with allowed error distance of 0.01) to the distance between them and one of the corrupted user's location. If the sum of s and t is less than $k/2$, we regard this run of the simulation as a successful detection of collusion attack, otherwise not.



(a) P_{SLDV} when $\Delta/r = .01$, $h_p = 0.65$ and $N = 1000$

(b) P'_{SLDV} when $\Delta/r = .01$, $h_p = 0.78$ and $N = 1000$

Fig. 5. Comparison of Analytical and Simulated values

For each value of k, we ran this simulation 5000 times and determined during how many times the system is able to detect the collusion attack. If m times out of the 5000 runs, the system is able to detect the attack, then P_{SLDV} is estimated as $\frac{m}{5000}$. Fig 5(a) shows the close match of our simulation results and the analytical results when the locations of users are unknown. For the case that the locations of the users are known to the adversary, we assumed $h_p = 0.78$. Fig. 5(b) compares the values of P'_{SLDV} for analytical and simulated cases. Again the results closely match.

Efficiency. The verification protocol has two rounds of communication for a new prover together with one broadcast message by verifiers intended for all "dynamic verifiers", and k response messages from the DVs. That is the overall communication cost of the protocol is $\mathcal{O}(k)$.

Maintaining $UserList$ requires storage equal to $\mathcal{O}(N)$, where N is the total number of verified nodes in the network. The value k determines the security of the system and can be chosen based on the required level of security and the trust assumption on the users (see Fig 4). The main computation required by the system is constructing messages that use efficient public and symmetric cryptographic primitives. This makes the protocol computationally very efficient.

7 Conclusion

We proposed a protocol for secure localization of mobile devices using the notion of dynamic verifiers, and proved its security against collusion attacks provided there are sufficient number of honest users active in the network. To our knowledge, this is the first protocol with security against large number of colluding nodes (more than the number of static verifiers) without making extra, and in many cases unrealistic, assumptions. A possible future work in this context would be to implement the protocol in a real wireless environment and evaluate the performance in different scenarios.

References

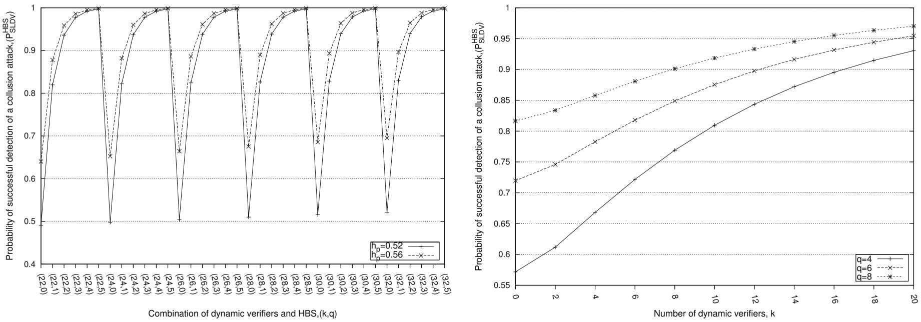
1. Čapkun, S., Hubaux, J.P.: Secure positioning of wireless devices with application to sensor networks. In: IEEE INFOCOM, Miami, USA (2005)
2. Čapkun, S., Hubaux, J.P.: Secure positioning in wireless networks. IEEE Journal on Selected Areas in Communications 24(2), 221–232 (2006)
3. Chandran, N., Goyal, V., Moriarty, R., Ostrovsky, R.: Position based cryptography. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 391–407. Springer, Heidelberg (2009)
4. Chiang, J.T., Haas, J.J., Hu, Y.-C.: Secure and precise location verification using distance bounding and simultaneous multilateration. In: WiSec, Zurich, Switzerland (2009)
5. Ferris, B., Haehnel, D., Fox, D.: Gaussian processes for signal strength-based location estimation. In: Robotics: Science and Systems, Philadelphia, USA (2006)
6. Jain, R., Puri, A., Sengupta, R.: Geographical routing using partial information for wireless ad-hoc networks. IEEE Personal Communications 8(1), 48–57 (2001)
7. Jansen, W., Korolev, V.: A location-based mechanism for mobile device security. In: World Congress on Computer Science and Information Engineering, Los Angeles, USA (2009)
8. Navas, J.C., Imielinski, T.: Geocast-geographic addressing and routing. In: MobiCom, Budapest, Hungary (1997)
9. Rasmussen, K.B., Čapkun, S.: Implications of radio fingerprinting on the security of sensor networks. In: SecureComm, Nice, France (2007)
10. Maurer, U.M.: Conditionally-perfect secrecy and a provably-secure randomized cipher. Journal Cryptology 5(1), 53–66 (1992)
11. Čapkun, S., Čagalj, M., Srivastava, M.: Secure localization with hidden and mobile base stations. In: INFOCOM, Barcelona, Spain (2006)
12. Sastry, N., Shankar, U., Wagner, D.: Secure verification of location claims. In: WiSE, Rome, Italy (2003)
13. Singelee, D., Preneel, B.: Location verification using secure distance bounding protocols. In: MASS, Washington, DC, USA (2005)
14. Brands, S., Chaum, D.: Distance-bounding protocols (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 344–359. Springer, Heidelberg (1994)
15. Savvides, A., Han, C.-C., Srivastava, M.B.: Dynamic fine-grained localization in ad-hoc networks of sensors. In: Mobicom, Rome, Italy (2001)
16. Wibowo, S.B., Klepal, M., Pesch, D.: Time of Flight Ranging using Off-the-self IEEE802.11 WiFi Tags. In: POCA, Antwerp, Belgium (2009)
17. Tu, S.-J., Fischbach, E.: A New Geometric Probability Technique for an N-dimensional Sphere and Its Applications to Physics. arXiv:math-ph/0004021 (2000)
18. Elson, J., Estrin, D.: Time Synchronization for Wireless Sensor Networks. In: IPDPS, San Francisco, USA (2001)
19. Song, H., Zhu, S., Cao, G.: Attack-resilient Time Synchronization for Wireless Sensor Networks. In: MASS, Washington, DC, USA (2005)
20. Bardram, J.E., Kjær, R.E., Pedersen, M.Ø.: Context-aware user authentication – supporting proximity-based login in pervasive computing. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) UbiComp 2003. LNCS, vol. 2864, pp. 107–123. Springer, Heidelberg (2003)

21. Ardagna, C.A., Cremonini, M., Damiani, E., De, S., di Vimercati, C., Samarati, P.: Supporting Location-based Conditions in Access Control Policies. In: ASIACCS, Taipei, Taiwan (2006)
22. Traynor, P., Schiffman, J., La Porta, T., McDaniel, P., Ghosh, A.: Constructing Secure Localization Systems with Adjustable Granularity Using Commodity Hardware. In: IEEE GLOBECOM, Miami, USA (2010)
23. Delaët, S., Mandal, P.S., Rokicki, M.A., Tixeuil, S.: Deterministic Secure Positioning in Wireless Sensor Networks. In: IEEE DCOSS, Santorini Island, Greece (2008)
24. Jadliwala, M., Zhong, S., Upadhyaya, S., Qiao, C., Hubaux, J.: Secure Distance-Based Localization in the Presence of Cheating Beacon Nodes. IEEE Transactions on Mobile Computing 9(6), 810–823 (2010)
25. Garcia-Alfaro, J., Barbeau, M., Kranakis, E.: Secure Geolocalization of Wireless Sensor Nodes in the Presence of Misbehaving Anchor Nodes. Annals of Telecommunications (2011)

A Hybrid Approach with Hidden-Base Stations

In a Hybrid system the two approaches to secure positioning, dynamic verifier approach and hidden base station (HBS) approach, are combined to save on infrastructure, while providing protection against location tracking attack on these latter systems as proposed in [3]. In this hybrid scheme a set $HiddenV$ of fixed hidden verifiers work as part of the positioning infrastructure along with the set of dynamic verifiers. We provide a modification to the basic HBS approach of [11]. Instead of using all the static hidden verifiers from set $HiddenV$, we select a random subset S_{HV} (of size q) from it. Randomized selection of HBS ensures that the attack mentioned in [3] can not be executed as stated. This is because the outcome of the protocol now depends on the selection of HBS which resides in different locations at different executions. However, to achieve sufficient randomness for S_{HV}, the size of $HiddenV$ (denoted by z) needs to be much larger than q so that $\frac{1}{\binom{z}{q}}$ becomes small.

When HBS are used in conjunction with DV, z can be smaller as verification decision will also rely on D_p which is randomly selected from a large set of nodes. Suppose, according to [3], adversary needs to execute the localization protocol c (i.e. $\mathcal{O}(\log(1/\delta))$) times to find out the locations of HBS, where δ is the precision of location. To make the attack successful, selection of S_{HV} needs to be same for all these c rounds, which is proportional to $(\frac{1}{\binom{z}{q}})^{c-1}$. Moreover, observations of DV either can not have impact on these verifications or D_p needs to be static for these c rounds, which leads to a success chance proportional to $\left(\frac{1}{\binom{z}{q}}\left(P_{dh} + (1 - P_{dh})\frac{1}{\binom{N}{k}}\right)\right)^{c-1}$ for the adversary to determine locations of HBS. Thus by increasing z with respect to q, it is possible to achieve security in this hybrid approach even when h_p is less than 0.5 (i.e. P_{dh} is close to 1). Specifically, when $h_p \geq 0.5$ (i.e. P_{dh} is small), randomization of S_{HV} is not necessary and z can be equal to q. Now, if z hidden base stations are used (along with k dynamic verifiers) as part of infrastructure, the probability that any q



(a) Values of P^{HBS}_{SLDV} vs (k, q) for different h_p when $\Delta/r = .01$ and $N = 1000$

(b) Values of P^{HBS}_{SLDV} vs k for different q when $\Delta/r = .05$, $h_p = 0.7$ and $N = 1000$

Fig. 6. Values of P^{HBS}_{SLDV}

Table 2. Dynamic Verifiers vs Hidden Base Stations

	HBS-only System	Hybrid System	
Δ/r	# of HBS	# of HBS	# of DV
0.05	6	4	6
	8	4	11
	10	6	11
0.07	8	4	11
	10	4	15
	10	6	11

of them are deceived by colluding attackers is bounded by $(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^q$. Hence, the probability of detecting a collusion attack (i.e. P^{HBS}_{SLDV}) would become $1 - (P_{dh} + (1 - P_{dh})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^{k/2})(\frac{0.809}{r/\sqrt{N(1-h_p)}} \times 2\Delta)^q$, where $P_{dh} = \sum_{i=\lfloor k/2 \rfloor}^{k} \binom{k}{i}[(1-h_p)^i h_p^{k-i}]$. Here we assume z is appropriately chosen according to h_p. HBS are particularly useful when we have a comparatively low trust assumption for users. With $h_p = 0.52$ and 22 dynamic verifiers alone, we can achieve P_{SLDV} around 0.5 where as the combination of 22 dynamic verifiers and 5 HBS give P^{HBS}_{SLDV} of approximately 0.998 (see Fig. 6(a)).

Even if we consider that assumptions stated in the attack of [3] do not hold, this hybrid system can help to save on infrastructure cost required for setting up multiple HBS in the verification region. From Fig. 6(b) we can see that instead of using 8 HBS we can incorporate a combination of 4 HBS and 11 dynamic verifiers (with $h_p = 0.7$) to achieve similar security. Table 2 shows few comparisons to illustrate the equivalency between a HBS-only system and a hybrid system containing both DV and HBS (for different values of Δ/r and $h_p = 0.7$).

B Impossibility of a Secure Protocol with Less Honest Users

Proof of Lemma 2. A valid user can lie about its location and share all of its cryptographic credentials with other collaborators. Moreover, a user can be honest for a certain period of time and then can attempt to establish a false location. Hence it's not possible for verifiers to determine whether a specific user is honest or dishonest before an authentication session with certainty. If the location verification protocol does not involve users to delegate verifying duty, we can devise a collusion attack similar to Lemma 1. Now, if the location verification protocol involves users to perform verifying duty and the number of colluding nodes scales up to them, we can form a collusion attack in any of the following ways:

1. In any selection of the users (D_p) to perform verifying duty, the number of expected malicious nodes will be as much as the honest nodes. Thus, taking a majority decision will not provide correct verification result regardless of the protocol. If unanimous decision is adopted for positive verification, it will always be possible for adversary to undermine a valid verification case.
2. If the network model allows users to communicate to each other which are not mobile, persistent adversary by observing communications of the users over a long period of time can determine their positions. Hence colluding nodes can take positions (with respect to the false location) in such a way that they can provide the correct response-time ($t^u_{r_i}$) for each user u. Consequently, regardless of which users are selected as DV, adversary becomes successful to establish the false location.

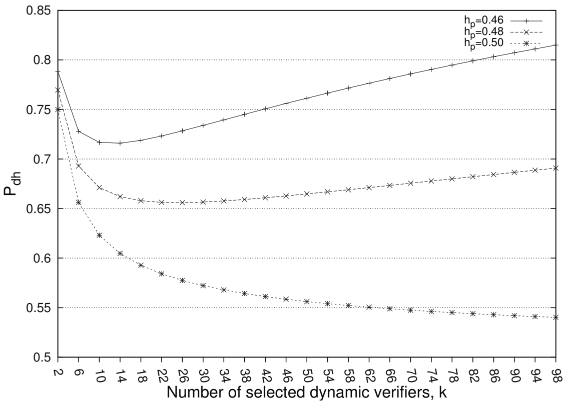


Fig. 7. Change in P_{dh} Value as k Increases

Figure 7 illustrates this fact by showing how reliability decreases (i.e. P_{dh} increases) when h_p is assumed to be ≤ 0.5 and we increase k indefinitely.

WiFiHop - Mitigating the Evil Twin Attack through Multi-hop Detection

Diogo Mónica and Carlos Ribeiro

Instituto Superior Técnico / INESC-ID Lisboa,
Rua Alves Redol 9, sala 605, 1000-029, LISBOA
{diogo.monica,carlos.ribeiro}@ist.utl.pt
http://www.gsd.inesc-id.pt

Abstract. Public hotspots have undeniable benefits for both users and providers. Users get ubiquitous internet access and providers attract new potential clients. However, the security mechanisms currently available (e.g. WEP, WPA) fail to prevent a myriad of attacks. A particularly damaging attack to public WiFi networks is the evil twin attack, where an attacker masquerades as a legitimate provider to mount wireless interposition attacks. This paper proposes WiFiHop, a client-sided tool that leverages the intrinsic multi-hop characteristics of the evil twin attack, to detect it. The proposed tool is technology independent (e.g. network bandwidth or latency), and detects the attacks in real time (i.e. before any user traffic is transmitted). It works with both open and encrypted networks. This tool was tested in a real-life scenario, and its effectiveness demonstrated.

1 Introduction

Wi-Fi networks have become a ubiquitous technology. These networks enable users to access the internet at home, at work, and even when traveling, but they are vulnerable to a number of threats, mainly because wireless access point (AP) operators often don't take the time to activate the adequate security features. The theft of private information is, therefore, becoming a growing concern. Users accessing the internet with wireless devices in public places (e.g. cafes or airport terminals) are particularly susceptible to these attacks.

An increasingly common strategy for the theft of private information is the evil twin attack, which consists of having an unsuspecting user automatically associating to an AP under the control of the attacker (a rogue AP). This access point is configured to mimic a legitimate access point (for example, by copying the legitimate network's SSID name), and enables attackers to eavesdrop all wireless communications done by the victims. Due to these two characteristics, these rogue access points are usually called *evil twin* APs.

There are essentially three different strategies for attackers to lure victims into connecting to their rogue AP. The first one is by having a higher signal strength than the other AP. This strategy works, since several operating systems choose the AP with the strongest signal strength, even when several APs with the same SSID are available. Also, users tend to choose the network with the higher signal strength when manually choosing a network to connect to. The second strategy uses the automatic re-association

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 21–39, 2011.

feature that several end-user systems provide. These systems have preferred network lists, containing the SSID names of the networks a user has previously connected to in the past. To exploit these lists, the attacker simply choses the evil twin AP SSID name to be one of the most commonly used SSID names (e.g. linksys), and waits for victims to connect. Finally, the third strategy involves using a denial-of-service attack against 802.11 networks. Attackers use well-known vulnerabilities in 802.11 to prevent a client from initially associating to a legitimate AP, or even to disassociate clients already associated [13]. The loss of connectivity resulting from the continuous disassociations, forces users to select other available wireless networks. This strategy can be highly effective, especially when combined with one of the previous two.

The evil twin attack is usually launched at public places where open-access WiFi networks are available. Locations like airports or cafes, are ideal, since there is no way for the users to distinguish rogue from legitimate APs [9]. Using an evil twin AP, attackers can effectively intercept all kinds of sensitive data such as passwords, credit-card information or even launch man-in-the-middle and phishing attacks. The malicious potential of this attack, together with the ease in configuring and deploying rogue APs, makes this attack a serious threat to wireless networks. This attack is particularly hard to trace, since it may occur only for a short amount of time. Depending on the objective of the attacker, after a few minutes of operation, the attack can be terminated, causing nothing more than a network disconnection for the victims (something somewhat common in wireless networks). In this short time frame, the attacker may already have compromised user's sensitive information.

The rest of this paper is organised as follows. Section 2 introduces the existing solutions to the generic problem. In Section 3 we detail the exact setup and assumptions addressed in the paper. Section 4 describes in detail the operation of our evil twin attack detection mechanism. In Section 5, our algorithm implementation is described. Finally, the results of the implementation of our mechanism are presented in Section 6. Section 7 concludes the paper.

2 Related Work

Existing solutions are mainly focused on the detection of rogue APs by the network administration, and not by the users themselves. One of the original ways of detecting these rogue APs relied on the manual verification of the available APs by a network administrator, using network enumeration tools such as Netstumbler [2]. This proved to be ineffective, since manual scans are time-consuming and expensive, and were therefore conducted infrequently. Since then, several automated systems have been proposed [1,8,6,11,10]. These solutions monitor the wireless medium and other types of information gathered at the network routers/switches, and compare them with a known authorisation list. For example, AirDefense [1] uses a combination of radio-frequency sensors jointly with an intrusion detection server to capture, process, and ultimately correlate network events, in search for APs with unknown ”fingerprints”. In other works [10,11], special diagnostic software was installed on mobile clients to perform wireless medium monitoring, helping the detection of rogue APs. Several variations, such as using sensors instead of sniffers to scan the wireless medium, have also been proposed ([6]). However, most of these solutions suffer from some, or all, of the following problems: they do guarantee a complete coverage of the network (required to ensure effective rogue AP detection); they may flag a normal AP (e.g. the access point of a nearby coffee shop) as a rogue AP; they do not work for rogue APs that possess authentication mechanisms such as WEP and WPA; and finally they may access unauthorised networks in the process of testing all the available APs in the vicinity.

A different line of work has also been pursued, where researchers attempt to distinguish wireless, from wired hosts, by analysing wired network traffic ([25,18,26,24,28,20,12,23]). The RIPPS system [18], for example, deduces wireless connectivity from the existing wired network traffic. The objective of RIPPS is to detect rogue APs without using wireless sensors. It uses active network traffic conditioning together with passive packet timing analysis. Wei et al. [26] use wireless traffic characteristics to distinguish wireless nodes. They present two detection algorithms that apply sequential hypothesis tests to packet-header data. These algorithms are able to detect wireless TCP traffic by considering specific properties of the 802.11 CSMA/CA mechanism. A similar work was done by Xie et al. [27], where the TCP jitter was proposed as the distinguishing characteristic between wired and wireless nodes. Several works [20,14] propose the analysis of the differences in inter-packet spacing to achieve the same objective. Other characteristics of the wireless traffic flows, like the client-side bottleneck bandwidth or the round trip time of network traffic, are also used in [16] and [17] (respectively). However, this general scheme assumes that there is a way to analyse all the network traffic, which might not be practical, and severely limits the scalability of these systems.

Finally, there are also some hybrid solutions like Yin et al. [28], where both sniffers and wired traffic analysis are used. In this particular case, they use an intrusion detection mechanism that uses a verifier on the internal wired network to send test traffic towards wireless edges. This mechanism can detect rogue access points, by detecting the relay of the test packets to the wireless edge. Liran Ma et al. [17] also proposed a hybrid solution, by correlating anomalies using both wired and wireless scans, being able to detect not only unauthorised but also compromised APs.

In these administrator-oriented solutions, a user must trust the network. Also, most of these solutions are not real-time, allowing short time attacks to remain undetected. Secondly, even if the detection is done in a timely fashion, many users can still fall prey to the attack, since there is no automated way of denying access to the evil twin APs or even warn the users of the attack. Our work shifts the usual paradigm, and empowers users with a tool that allows them to detect if an evil twin attack is being launched, before they actually start using the network.

To the best of our knowledge, only one author employs a client-side method, designed for evil twin attack detection. Song et al. [21], propose ETSniffer, using timing measurements to distinguish a one-hop from a multi-hop setting. However, as stated in [18], these timing measurements are technology dependent. With the increase in wireless networks transmission rates (e.g. 802.11n), differences in delay between wireless and wired settings will fade, or at least, vary. This means that a wireless node may become indistinguishable from a wired node, or, in the particular case of ETSniffer, a multi-hop setting, indistinguishable from a one-hop setting in a faster technology.

Our solution differs from previous work in that it does not depend on timings to detect a multi-hop setting. Instead, detection is based on the behaviour of the legitimate AP, thwarting the attacker from evading the mechanism. Additionally, we do not require the knowledge of an authorisation list, and allow the user to proactively test the network, prior to using it. The fact that detection is done by the users makes this an efficient and cost-effective solution, where no modifications to the client hosts are required.

3 Problem Statement

We assume the existence of an 802.11 wireless LAN device, capable of operation in monitor mode, being operated by a user that wishes to access the internet through a free hotspot. This user has no knowledge of the infrastructure, and cannot, thus, verify the authenticity of any available access points in the vicinity. We further assume that the user is in radio range of the legitimate access point, where the wired connection to the internet is located. In fact, this scenario describes almost all public locations where a free hotspot is available, such as cafes or airports.

In normal operation, a user in the vicinity of a legitimate AP will associate directly to it, and access the internet. However, in the case where there is a malicious attacker launching an evil twin attack, the user might be fooled into associating to the evil twin AP, instead of directly connecting with the legitimate one (Figure 1). This evil twin AP will also allow internet access to the user, by forwarding all the information received from the user, wirelessly, to the legitimate AP. It will, therefore, be capable of intercepting all the user information, without being detected. Notice that, in both scenarios, the legitimate AP is within range of the user.

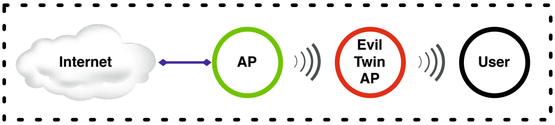


Fig. 1. Illustration of the problem being addressed

Our objective is to provide a convenient and usable technique to detect the existence of an evil twin attack. The detection scheme will be required to possess the following properties:

- Operation not detectable by the attacker;
- Capable of operation in encrypted networks;
- Independent control of both the probability of detection p_D and of false alarm p_{FA};
- Non-disruptive operation;
- User-sided operation.

The rationale for requiring most of these properties is clear. A note must, however, be made, concerning the last property listed (*user-sided operation*). The detection algorithm to be developed is intended to constitute a tool which users may ubiquitously possess, to be used automatically whenever the user joins a public hotspot, or in any instance where it may be considered necessary. The use of solutions such as end-to-end VPNs, is much more complex in terms of implementation/setup and still leaves users vulnerable to layer 2 and denial-of-service attacks.

4 Detecting the Evil Twin AP

The presence of an evil twin AP will be detected using the fact that, when such an attack is in place, the user's data must transit the wireless channel between the evil twin and the legitimate AP (see Figure 1). If we can detect the existence of this extra wireless hop, we will then have attested the presence of the evil twin AP.

The basic overall scheme for detecting the existence of a multi-hop setting between a user and the internet is the following (details will be addressed later on, and will depend on the particular type of network under analysis): the user sends a watermarked packet (or packet sequence) to the internet, through the access point to which it is currently associated with; the watermark signature is known only to the user; after sending this packet in the channel associated with its AP connection, the user listens to a different wireless channel, and tries to detect the presence of the watermark in the traffic passing through that wireless channel. If an evil twin attack is being launched, the watermark will necessarily appear on the wireless link between the evil twin and the legitimate APs, and the attack will, therefore, be detected, if the user repeats the procedure for every one of the available wireless channels. We called this generic scheme WiFiHop.

Since the time taken for the wireless access point to retransmit the watermarked packet (the store-and-forward delay of the AP) is orders of magnitude lower than the time taken for a user to switch channels and begin parsing wireless traffic, detection of the watermark cannot be done on the outward path, since by the time the user is ready to detect the watermark, the packet has already been forwarded to the legitimate AP. We therefore send the watermark packet to an external server on the internet, which we assume not to be malicious, having the packet being replayed back to the user a pre-defined number of times. The watermark will then be detected on the incoming, returning path, when being forwarded to the evil AP by the legitimate one. This scheme not only allows the user to properly switch channels and initiate monitoring mode (if that is the case), but it also allows the option of simply requesting the watermark from

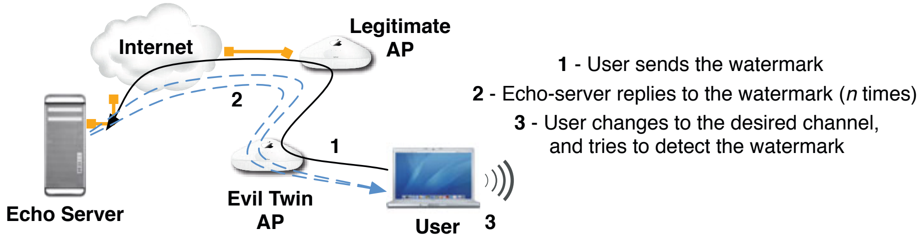

Internet
Legitimate AP
2
1
Echo Server
Evil Twin AP
User
3
1 - User sends the watermark
2 - Echo-server replies to the watermark (n times)
3 - User changes to the desired channel, and tries to detect the watermark

Fig. 2. The WiFiHop mechanism

the server, avoiding the need for its outward transmission, a scheme which will be used in the later parts of the paper. Figure 2 depicts this evil twin detection mechanism.

There is however, one important issue that has to be taken into account. We have to consider the possibility of existence of an encrypted channel between the evil twin AP and the legitimate AP. While most hotspots do not have encryption in place, wireless security mechanisms such as WEP and WPA, are commonplace nowadays. In the case where the link between the evil twin and the legitimate AP is protected with one of these security mechanisms, the user will be unable to detect a watermark embedded into the payload of the encrypted packet. We will, thus, provide two different solutions for detecting the watermark in these two scenarios (plain and encrypted channels): Open WiFiHop and Covert WiFiHop.

In both solutions, we will need to take into account the possibility of packet loss. In multi-hop wireless networks, specially with high traffic load, packet losses are frequent. This possibility of packet loss will, therefore, be accounted for in the proposed scheme.

Finally we note that there are legitimate uses for multi-hop schemes (such as range expansion). While false positives may occur when both APs (the original and the expander) are in range, the user will simply be instructed to use the AP that is directly connected to the internet. If only one of the APs is in range no false positive will occur.

4.1 Open WiFiHop

In Open WiFiHop, we assume that there is no encryption between the evil twin AP and the legitimate AP. Watermark detection can, thus, be easily done. The user creates a random bitstring, and sends it to the echo-server. Then, it tries to find the exact same pattern in the payload of all the packets being sent in one of the alternative wireless channels. The test is repeated for every wireless channel, other than the one being used in the association between the user and the service providing AP.

In this particular case, and since we can increase the size of the random bitstring, there is almost no possibility of having false positives (false alarm). A false alarm would happen if the random bit string chosen by the user happened to be present, by chance, in any wireless packet other than the echo-server's answer. However, by choosing a bitstring of any reasonable size (e.g. 128 bits), we can bring this probability arbitrarily close to 0. This aspect will be discussed again, when discussing the detection performance of the algorithm.

However, there still is the possibility that either the client's request or the reply watermarked packets are lost in transit, something that would make Open WiFiHop return a false negative (miss). There are four situations that will translate in Open WiFiHop not detecting the watermark: packets being lost between the echo-server and the legitimate AP; packets being lost in the air, between the legitimate AP and the evil twin AP; packets being delayed more than the time window allocated to the test; and finally, reply packets not being detected by the sniffer (the user network card). Of these, the last one is conspicuous, since it creates a considerable asymmetry between the probabilities of losing a request, and the probability of not detecting a reply. As will be seen, the efficiency of the sniffer can vary widely (we used both TCPDump [5] and Scapy [4]), but will, in all cases, constitute a major source of loss of reply packets. We will, therefore, consider two different probabilities of packet loss, one for outgoing packets (requests),

p_{plo}, and one for incoming packets (replies), p_{pli}. To be able to control the statistical performance of the method in this packet loss environment, clients will repeat the request c times for each wireless channel, and the server repeats the watermarked packet n times, in response to each received request.

Statistical Performance. It is assumed that the observation window (t) is chosen to be wide enough so as to render negligible the probability that the transmitted packets fall outside the window (which might otherwise occur, due to excessive latency in the network). The choice of t will be addressed later on in the paper. Since the number of bits constituting the watermark can be arbitrarily long (within the restriction imposed by the network's MTU), the probability of a particular pattern appearing in an external packet (p_{ra}) is, for all practical purposes, as close to zero as desired. Even with watermarks as small as 64 bits, the probability of random appearance is of the order of 10^{-20}. Hence, the only parameters of interest to the analysis of the statistical performance of the detection scheme are the end-do-end probabilities of packet loss (p_{plo} and p_{pli}) of the server-sniffer channel. The relevant probabilities are, thus, easily obtained:

- **Probability of miss** - p_M. An error of the second kind (failing to detect the watermark) may result both from the loss of the c requests transmitted by the client, or from the loss of all the packets transmitted by the echo-server (n, for each received request). Assuming statistical independence between packets (a waiver to all cases such as jamming or long interference bursts), the conditional probability that the n replies are not detected, given that a request was received by the echo-server, is given by:

$$p_{M|r} = p_{pli}^{n}. \tag{1}$$

  The overall probability of a false negative is given by:

$$p_M = \sum_{i=0}^{c} \binom{c}{i} (p_{plo})^{(c-i)} (1-p_{plo})^{i} (p_{M|r})^{i}. \tag{2}$$

- **Probability of detection** - p_D. The probability of correct detection of the watermark is given by: $p_D = 1 - p_M$
- **Probability of false alarm** - p_{FA}. An error of the first kind (spurious detection of a watermark) can be considered negligible, as previously discussed, since even small watermarks will bring this probability to negligible values: $p_{FA} \approx 0$.

As can be seen from (1) and (2), increasing n will imply a decrease in the probability of false negatives (p_M). However, there will be a residual probability of false negatives which no increase in n can affect, since it is associated with the probability of loss of all the request packets (the term for $i = 0$, in (2)), a situation where n plays no role whatsoever. This implies that, for a desired probability of detection p_D^*, and even though a trade-off between c and n is possible in general, there is a minimum threshold for c, which can not be compensated by an increase in n:

$$c \geq \frac{log(1-p_D^*)}{log(p_{plo})} \tag{3}$$

The observed particular values of the parameters (and the resulting probabilities) will be presented at a later section, when discussing the implementation made by the authors. Also, even though we are required to obey (3), there is a trade-off to be made between c and n, since both parameters will affect p_D. However, their effect on the time to complete the test is substantially different: increasing c will imply considerably higher accrued delays. Hence, in our implementation, we kept c as small as possible, and adjusted n to achieve the desired p_D.

4.2 Covert WiFiHop

If the wireless link between the evil and legitimate APs is encrypted, we cannot hope to access the payloads of the exchanged packets. In this case, we modify our scheme, focusing its principle of operation on the one measure we can rely on: packet length. Since the relevant security mechanisms (e.g. WEP, WPA) have deterministic, predictable behaviours, concerning the increase in length of the unencrypted packets [3], we can create an effective watermark using a sequence of packets with pre-determined lengths. Detecting the watermark will then become a problem of detecting a set of packets with the appropriate length sequence. This mechanism also works seamlessly in networks without encryption, being, however, slightly more complex to implement.

Another option, which will not be presented in this article, is to use, as a watermark, not a sequence of chosen lengths, but a sequence of length differentials. This will make the scheme invariant to all network cyphers whose final lengths are affine functions of the plain packet lengths, but will both increase complexity, and decrease the overall statistical performance of Covert WiFiHop. The increase in complexity results directly from the fact that, since we will be operating on a packet switched network, extraneous packets may, and will, be inserted between the watermark packets (more on this will be said later). This creates a hard setup for the detection of length differentials, and will imply an increased scheme complexity. Also, any particular length difference is, statistically (all other things being equal), orders of magnitude more probable than a particular length (many candidate pairs may create that length difference, while length coincidence has a single candidate (that particular length itself). This approach should only be used, therefore, on a *need to* basis, and will not be addressed here.

Before pursuing the analysis of Covert WiFiHop, some notes must be made. The first one concerns the fact that, while in Open WiFiHop, p_{FA} could be considered null, due to the statistical improbability of a random generation of the watermark, in this case, the probability of appearance of extraneous packets with coincidental lengths is not infinitesimal, and such events must, therefore, be considered. In fact, even though with low probability, the appearance of coincidental lengths are not rare events, and its probability depends on the amount of time spent in the analysis of network traffic to support the choice of the least observed packet lengths. To illustrate this, we processed a 4 day sequence of packets from the widely used SIGCOMM conference trace [19] with 10 different analysis periods (0.1 s to 1 s, in 0.1 increments). For each analysis period, we computed the probability that the least observed packet length during the analysis period (or, if the set of least observed lengths is multi-valued, one randomly chosen length from that set) was observed in the 5 minutes immediately following the analysis period. The obtained results can be seen in Figure 3. As can be observed, longer

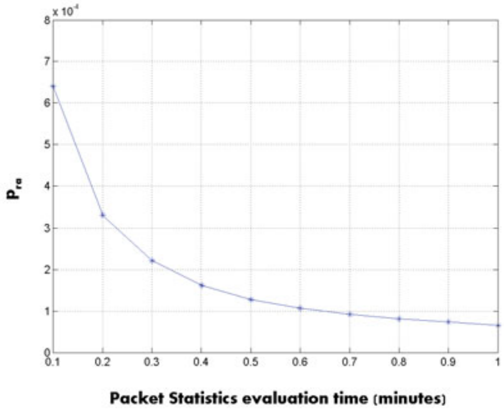


Fig. 3. Probability of extraneous appearance of packets with the chosen length

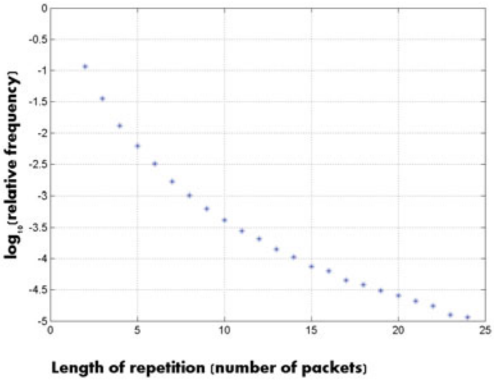


Fig. 4. Probability of sequences of repeated lengths

analysis periods will provide length choices less prone to extraneous appearance. Also, we note that small observations periods (e.g. 6 s) are already very effective in providing low values of p_{ra}.

Secondly, we must consider the fact that common life sequences of packet lengths do not constitute white noise processes. That is, there are significant correlations between successive lengths. Namely, repeating lengths are very frequent. As an illustrative example, we evaluated the number of repetitions in the time series of observed packet lengths from the SIGCOMM trace [19]. To avoid bias in the results, the sequence was preprocessed, and all retransmissions and non-data packets (e.g. beacons) removed. It was found that, in that particular sequence (approximately 29.3 million packets), 17.8% of the packets were immediately followed by one or more packets of the same length. As can be seen in Figure 4, sequences of up to 25 packets with the same length fall in the range of probability $p \geq 10^{-5}$.

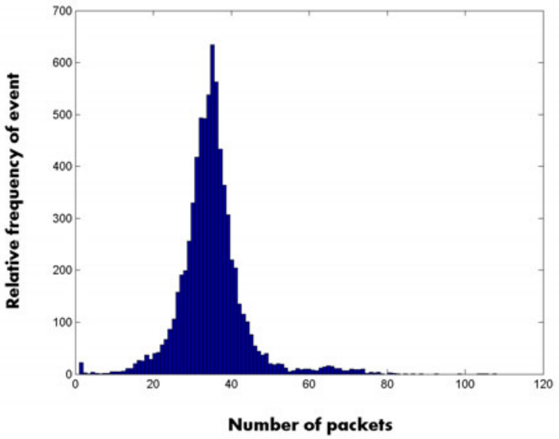


Fig. 5. Distribution of the number of intercalated packets

For confirmation purposes, we made several on campus recordings of 2-hour periods, in different operational situations (weekends and weekdays, morning and late afternoon). The obtained results were, in fact, similar, with the appearance of the same behaviour of repetitions.

This fact immediately rules out the hypothesis of choosing, as watermark, a sequence of packets with the same length, at least for small k. Even if that particular length has a low probability of random appearance, its conditional probability, given that one packet of the coincidental length appeared, is too high. Which, of course, means that the probability of random occurrence of the k-sized sequence may remain of the approximate order of magnitude of the probability of random occurrence of its first value (p_{ra}). We will therefore use as watermark a sequence of packets of different lengths, these lengths being chosen from the set of packets lengths with lower probability (resulting from a local ad-hoc pre-analysis). Since, typically, there will be many lengths of equivalent low probability to choose from, the choice does not pose any additional difficulties or constraints.

Thirdly, we must also consider the fact that, in a packet switching environment, it is possible that extraneous packets are inserted amongst the watermark sequence packets. In fact, for any reasonable traffic load conditions, it will be highly unlikely for the watermark sequence of packets to be transmitted from the legitimate to the evil AP end-to-start, without intercalated extraneous packets, belonging to other conversations. To appreciate this, we can see, in Figure 5, the histogram of the number of packets received in between packets of a 30 packet watermark sequence. Both the details of the test wireless network, and of the traffic conditions in all three profiles (low, medium and high traffic), are detailed in Section 5, more particularly, in Figure 6 and Table 7(a).

The observed mean number of extraneous packets inserted between each pair of packets belonging to the watermark sequence was 34.8. As can be seen in Figure 5, the distribution has a heavy right tail, and inserted sequences of more than 100 packets were observed. The situation becomes worse, of course, at high traffic levels (see

Section 5 for details), were the observed mean rose to 96.4, with the appearance of inserted sequences longer than 300 packets.

Finally, it is also possible for packets in the sequence to arrive at the legitimate AP in reversed order. This is thus the environment which the sequence detection scheme must be designed to cope with.

Detection of the watermark sequence will be made by progressing through the steps of a k-state finite state machine. The machine state progresses whenever a packet with the proper length is detected. If, for example, the watermark is chosen to be a sequence of three packets of lengths 110, 250 and 130, the three-state machine ($k = 3$) will terminate when all three lengths 110, 250, 130 are observed, in this relative order. It does not matter how many extraneous packets are interspersed between the water mark sequence, since the machine state never regresses. Due to the possibility of packet loss, we again have clients repeating the request c times, and the server repeating the watermark n times. That is, n sets of k packets will be transmitted for each request received by the server, each one of the k sets of packets being constituted by packets with the chosen lengths.

Even though the watermark to be detected is constituted by a sequence, we will not use a Sequential Likelihood Ratio Test - SLRT [22] to support the decision, as could be expected in such a type of sequence based detection (see, for example, [28]). The reason is threefold, and lies deep in the assumptions of such a test, which our particular setup fails to obey. To start with, we have the lack of statistical independence between samples of the stochastic process formed by the sequence of packet lengths (mainly resulting from the high probability of repetitions of the same length, as discussed above). But there are two eventually deeper and more structural reasons: the lack of stationarity and lack of ergodicity of the underlying stochastic process (both of which are in the basis of the SLRT). To appreciate this, let us consider a simple example, where an SLRT is being used in a medium traffic situation, and we have just received a packet with the correct length, but we have not yet crossed one of the decision thresholds. This means that we will look at the following packets to further refine the associated probabilities and, hence, hopefully progress towards one of the decision boundaries (see, for example, [15]). The problem is now: is the probabilistic density of the next packet unchanged by the knowledge that we've just detected one packet of the proper length? And what about the density of, say, the 35^{th} packet to arrive after the just detected packet? Is its probabilistic density unchanged? And is it the same density of the next packet? The answer to all these questions is no (which, of course, implies that there is no stationarity, and, hence, no ergodicity). Since, in these traffic conditions, the distribution of the number of packets inserted between watermark packets has a mean of approximately 35 (see Figure 5 and the related discussion), the 35^{th} packet is much more likely to belong to the watermark than the next packet. Those two packets will, therefore, have different associated probability distributions, which immediately rules out stationarity. The assumption of ergodicity cannot thus be maintained (meaning that we cannot reflect the temporal behaviour of the sequence on the statistics of each packet). All in all, the statistical basis for the use of an SLRT is completely compromised.

The chosen decision rule is therefore, the following: if the machine terminates (reaches its final state) within the time period allocated to the test, the watermark is considered detected (and, therefore, the service providing AP classified as an evil twin); otherwise, it is decided that no watermark was transmitted on that channel, and the test is repeated for the remaining channels.

Statistical Performance. In this case of encrypted networks, the major difference will be observed on p_{FA}, since, as discussed, the probability of extraneous packets having the same length than the marking packets (p_{ra}) is not a negligible quantity. n sets of k packets will be sent, each set having the chosen packet length sequence. Detection occurs when the sniffer has seen all the required packet lengths, in the proper sequence.

We will assume that a false negative due to the loss of the watermark sequence will happen if packet losses occur in all transmitted sets. In fact, this is a conservative assumption, since a partial length sequence, observed on, say, the first set, may be completed by length observations belonging to the second set. Also, it is possible that a lost packet may be substituted by an extraneous packet with the proper length, thus leading to a correct detection of the sequence on that set.

For small values of n, the assumption error is small. In any case, it always leads to a conservative evaluation. With this assumption, the associated probabilities thus become:

- **Probability of miss** - p_M.

$$p_{M|r} = (1 - (1 - p_{pli})^k)^n. \tag{4}$$

  The overall probability of a false negative is, again, given by:

$$p_M = \sum_{i=0}^{c} \binom{c}{i} (p_{plo})^{(c-i)} (1 - p_{plo})^i (p_{M|r})^i. \tag{5}$$

- **Probability of detection** - p_D. The probability of correctly detecting the length sequence is given by: $p_D \geq 1 - p_M$.
- **Probability of false alarm** - p_{FA}. An error of the first kind (spurious detection of the watermark) will occur if we have k extraneous occurrences of appropriate length. Designating by λ the rate of such an extraneous occurrence, we may use a Poisson process to majorate the probability of false alarm:

$$p_{FA} = 1 - \sum_{i=0}^{(k-1)} \frac{(\lambda\, t)^i}{i!} e^{-\lambda\, t}, \tag{6}$$

  where t is the observation period.

This implies that both p_{FA} and P_D can be independently controlled. Increasing k will decrease p_{FA} as desired. This will also reduce p_D, but p_D can then be brought to the desired level by properly choosing n or c. Denoting the desired probabilities by p^*_{FA} and p^*_D, and noting that the previously threshold effect for c is also applicable for the Covert WiFiHop case, we must choose k in such a way as to guarantee that

$$\sum_{i=0}^{(k-1)} \frac{(\lambda\, t)^i}{i!} e^{-\lambda\, t} \geq 1 - p^*_{FA}, \tag{7}$$

and

$$c \geq \frac{log(1 - p_D^*)}{log(p_{plo})} \tag{8}$$

The observed particular values of the parameters, and a description of our particular implementation of WiFiHop, will be presented next. We again note that, in the trade-off to be made between c and n, we kept c as small as possible, to minimize execution time.

5 Implementation

To evaluate the performance of the proposed schemes in a real life situation, a test setup was put together. We deployed an access point in our university campus, and configured it to provide access to the internet. The objective was to test both algorithms under real traffic and environmental conditions, by having several other wireless networks coexisting with our own. An *evil twin* AP was also deployed and configured to act as a rogue access point. Finally, we set up a client with wifihop-ng, and an extra computer that served as a wireless monitor and captured all the wireless packets being sent, for later examination. A representation of this network is shown in Figure 6.

Both the legitimate and the evil twin APs are FON2201, supporting IEEE 802.11b/g. The client used was an Intel desktop, running a Linux flavoured operating system, with a Proxim ComboCard Gold that supports IEEE 802.11a/b/g. The monitor was an iMac running OSX Snow Leopard. Note that all of these off-the-shelf equipment supports monitor mode, and therefore, meets the requirements to run WiFiHop. We tested several different network configurations. In all the tests, the client accessed the evil twin AP through an open network[1] (no encryption), but we varied the encryption available between the evil twin and the legitimate APs. The encryptions used were: WEP with a 128 bit key; WPA2 using AES; and OpenVPN using blowfish. We note here a limitation of our system: the inability of currently dealing with security systems that use random padding on their cryptography algorithm. However, we note that the IEEE 802.11 standard does not offer this feature [3], and the use of VPNs in Hotspots is very uncommon. This leaves WiFiHop capable of detecting the very large majority of present day attacks, as claimed.

To be able to infer the influence of network traffic on the performance of WiFiHop, we added two extra clients to the network, connected directly to the legitimate AP, that generated constant TCP and UDP traffic. The generation of traffic was done using HTTP downloads with rate-limiting, and iperf, a traffic generation tool, used to generate constant UDP traffic streams. The traffic profiles generated by these two extra clients are presented in Table 7(a).

The first task was the characterisation of the parameters relevant to the statistical configuration of the tests. Namely, the following probabilities had to be estimated: the probabilities of packet loss p_{pli} and p_{plo}, the rate of extraneous appearance of packets with lengths coincidental with the length of the next expected watermark packet λ, and the statistical distribution of the roundtrip delay in the user/echo-server channel.

[1] The use of an open network between clients and the evil twin AP is just a particularity of our testing environment, and not a limitation of WiFiHop itself.

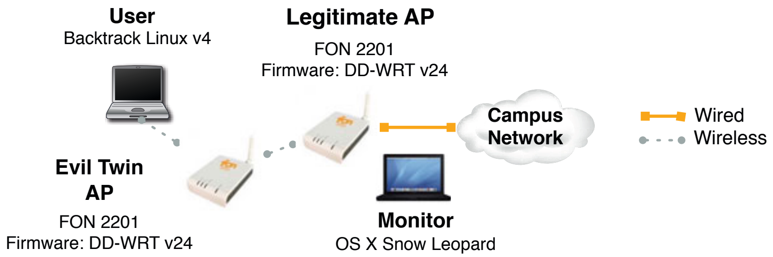


Fig. 6. Illustration of the network in which WifiHop was tested

Profile	DL rate (Mbps)	UL rate (Mbps)
Low	2	1
Medium	8	5
High	16	12

(a) Traffic generated by the extra clients.

	Low Traffic	Medium Traffic	High Traffic
p_{pli} (tcpdump)	0.0217	0.0239	0.0805
p_{pli} (scapy)	0.2574	0.6721	0.8911
p_{plo}	$1.333 \cdot 10^{-4}$	$2.667 \cdot 10^{-4}$	0.16

(b) Probability of packet loss.

Fig. 7. Traffic profiles and probability of packet loss

Determination of λ*.* We used a six second window to determine the least probable packet lengths. That is: the choice of packet lengths to be used in the watermark sequence (encrypted networks case) is obtained by monitoring network traffic during six seconds, and choosing the k least observed lengths. If there are more than k lengths with the same minimal length, which turns out to be invariably the case, the choice is randomly made between the minimal length candidates. From Figure 3, we see that this choice implies $p_{ra} = 6.4E-4$. Since this probability was obtained for 5 minute blocks, with a mean number of 25606 packets, this translates, in terms of Poisson arrival rate to $\lambda = 0.0533$ arrivals per second.

Determination of p_{pli} *and* p_{plo}*.* The probabilities of packet loss were experimentally determined. Runs of 10 mins were made, in which control packets were inserted in the network, along with the random traffic generated by the two extra clients (see Table 7(a)). To measure the p_{pli} we divided the number of control packets captured by the sniffers, and divided them by the number of control packets sent by the remote server, effectively obtaining $1-p_{pli}$. For p_{plo}, we simply divided the number of control packets sent from the client, by the number of control packets received at the server, obtaining $1-p_{plo}$. In the case of p_{pli}, and since the sniffer will be the predominant cause of packet loss, we repeated the test with two different sniffers: TCPdump [5] and Scapy [4].

As can be seen in Table 7(b), there is a very big difference in performance between TCPdump and Scapy. Therefore, scapy was used in all the implementation tests, to obtain worst case evaluations.

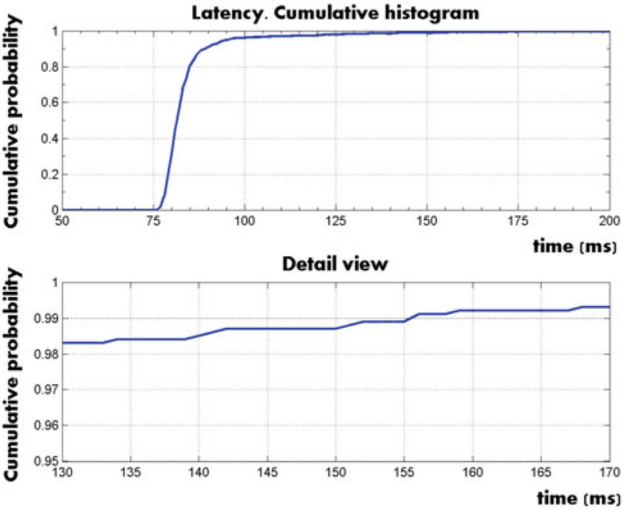


Fig. 8. Cumulative histogram of the round-trip delay

		Low Traffic	Medium Traffic	High Traffic
Open	c	1	1	3
	n	6	19	18
Covert	c	1	1	3
	n	6	19	18
	k	1	1	1

(a) Required values for c, n and k.

Traffic Profile	WiFiHop	Attacks detected
Low Traffic	Open	100%
	Covert	100%
Medium Traffic	Open	100%
	Covert	100%
High Traffic	Open	98.44%
	Covert	98.05%

(b) Effectiveness of WifiHop.

Fig. 9. Parameters and results of WiFiHop's evaluation

Round-trip delay distribution. To evaluate the round-trip delay distribution, a simple experiment was setup: a request was transmitted from the wireless client to the echo-server; upon reception, the echo server replied with a sequence of 30 packets, addressed to the wireless client. The time between the wireless transmission of the request and the wireless reception of the last packet in the sequence was recorded. The experiment was repeated 1000 times, in medium traffic conditions. The cumulative histogram of the recorded values can be seen in Figure 8.

As can be seen in this figure, we only need 156 ms to ensure a 0.99 probability that all the sequence is received. However, the user needs some additional time to be able to change channels, and enter monitor mode. Hence, the echo-server will be made to delay transmissions by an extra 500 ms. This means that the attempt to detect the watermark transmission should not terminate until $t \geq 656$ ms after the transmission of the request (or watermark) to the echo-server. In all tests, t was set to 1 s.

Determination of c, n and k. With the above values of λ, p_{plo} and p_{pli}, it is easy to obtain the needed values of c, n and k, by using the formulas of Section 4. Imposing, as objectives, $p_D^* \geq 0.999$ in the medium and low traffic conditions, $p_D^* \geq 0.98$ in the high traffic case (due to the extremely heavy conditions of the test, well above the limits of reasonable operation), and $p_{FA}^* \leq 0.001$ in all cases, the required values of c, n and k are, for both (Open and Covert) cases, the ones represented in Table 9(a). These were, therefore, the parameter values used in our implementation of WiFiHop, for the performance tests of Section 6.

Since, in the Covert WiFiHop case, k turned out to be one in all traffic conditions (due to the low probability of random appearance of coincidental lengths), the values of c and n are the same for both Open and Covert WiFiHop.

Echo server. The echo-server can deployed through the use of a simple script on any public hosting service (we used a 10-line Python script). This simplicity, opposed to, for example, the deployment and configuration of an end-to-end VPN, enables users to easily create private echo-servers in which they can trust. In Open WiFiHop, the server, after receiving a UDP packet, transmits n replies, each one of them containing the watermark, but delaying the replies by d seconds. In our experiments, clients were always able to switch channels in less than $500ms$. Therefore, $d = 0.5$ was used. The Covert WiFiHop receives, as payload of the UDP request packet, the set of packet sizes to be used in the watermark sequence. The packet sizes, as mentioned in the previous section, are chosen by the user, after testing the wireless medium for six seconds. As in the previous case, the server waits d seconds before transmitting the watermark.

WiFiHop. A command-line tool was developed, which implements both Open and Covert WiFiHop: wifihop-ng. When ran, wifihop-ng puts the wireless device in monitor mode, allowing the wireless device to receive every transmitted packet, regardless of origin or destination. While not all wireless devices support this specific mode, an increasing number of them do [7].

After association with the AP, wifihop-ng verifies internet connectivity by sending a heartbeat to our remote echo-server. Then, it sends the watermark to the AP and immediately switches to one of the other available radio channels, listening all transmitted packets. In Open WiFiHop, we chose a small UDP packet, with a total of 44 bytes, that contains a random 128 bit string in the payload. The choice of UDP over TCP was made to avoid the TCP three-way handshake. However, any kind of packet can be used. In Covert WiFiHop, the watermark is also UDP packet, containing the packet sizes to be used on the watermark (see Section 4 for more details on this).

To distinguish between both mechanisms wifihop-ng receives an input flag. If the watermark is not detected within the time period t, the procedure is repeated with a different wireless channel. Conversely, if the watermark is detected, wifihop-ng returns the origin and destination MAC addresses of the APs involved in the communication. The user is then asked if he wishes to connect directly to the legitimate AP.

6 Results

Regarding the effectiveness of WiFiHop, we show in Table 9(b) the results of a total of 6000 different trials, using the previously calculated parameters (Table 9(a)). We made 1000 tests for each one of the traffic profiles present in Table 7(a), for both Open and Covert WiFiHop. The immediate conclusion is that WifiHop had no false negatives for both the low and medium traffic profiles. This was expected, since we had imposed $p_D \geq 0.999$ (and, hence, $p_M \leq 0.0001$) in both these settings. In the high traffic case, the observed detection probabilities were $p_D = 0.9844$ and $p_D = 0.9805$ (for Open and Covert WiFiHop, respectively), which again are within the required range. In none of the tests false positives occurred. That is why this parameter is not shown in Table 9(b).

The user does not need to test every available wireless channel, since wifihop-ng can search for beacons, and tests can safely be made only on channels in which networks are being advertised. However, we will consider the worst case scenario (testing all 11 channels). Each test requires six seconds to choose the less probable packet lengths (in the case of Covert WiFiHop), an additional $t = 1$ seconds for each wireless channel to be tested, and approximately 0.5 seconds to change the channel back to the network operation channel. This means that a full set of tests will last a minimum of approximately 22.5 seconds. In fact, all the tests performed with wifihop-ng, required approximately 30 seconds. Such a full set of tests will only be done once, when the user joins a new network, and therefore, this time-frame does not seem to be excessive or impractical.

7 Conclusions

User-sided evil twin attack detection is viable. It can be done in useful time, and is statistically high effective in the range of normal network operations. These detection mechanisms can operate in both open and encrypted networks (e.g. WEP, WPA and some VPNs). Also, they avoid many of the difficulties associated with some server-sided detection mechanisms, such as the need for several wireless sniffers, the high false positive rate, and the non-real time detection. We have shown that WiFiHop can be implemented in off-the-shelf equipment, giving wireless hotspot users the capability of individually detecting a evil twin attack in the networks to which they access, without having to trust the network operator. In our implementation of WiFiHop, this detection was always done in less than one minute, with virtually no false positives, and a very low rate of false negatives.

References

1. Airdefense - tire of rogues? solutions for detecting and eliminating rogue wireless networks, `http://www.airdefense.net/whitepapers/roguewatch_request2.php`
2. Netstumbler, `http://www.netstumbler.com/`
3. Nist guide to securing legacy ieee 802.11 wireless networks, `http://csrc.nist.gov/publications/nistpubs/800-48-rev1/SP800-48r1.pdf`
4. Scapy project, `http://www.secdev.org/projects/scapy/`
5. Tcpdump, `http://www.tcpdump.org/`

6. Wavelink, http://www.wavelink.com
7. Wireless card compatibility list, http://www.aircrack-ng.org/doku.php?id=compatibility_drivers
8. Wisentry - wireless access point detection system, http://www.wimetrics.com/Products/WAPD.htm
9. Abdollah, T.: Ensnared on the wireless web, http://articles.latimes.com/2007/mar/16/local/me-wifihack16
10. Adya, A., Bahl, P., Chandra, R., Qiu, L.: Architecture and techniques for diagnosing faults in ieee 802.11 infrastructure networks. In: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking, MobiCom 2004, pp. 30–44. ACM, New York (2004), http://doi.acm.org/10.1145/1023720.1023724
11. Bahl, P., Chandra, R., Padhye, J., Ravindranath, L., Singh, M., Wolman, A., Zill, B.: Enhancing the security of corporate wi-fi networks using dair. In: Proceedings of the 4th International Conference on Mobile Systems, Applications and Services, MobiSys 2006, pp. 1–14. ACM, New York (2006), http://doi.acm.org/10.1145/1134680.1134682
12. Baiamonte, V., Papagiannaki, K., Iannaccone, G.: Detecting 802.11 wireless hosts from remote passive observations. In: Akyildiz, I.F., Sivakumar, R., Ekici, E., Oliveira, J.C.d., McNair, J. (eds.) NETWORKING 2007. LNCS, vol. 4479, pp. 356–367. Springer, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1772322.1772361
13. Bellardo, J., Savage, S.: 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In: Proceedings of the 12th Conference on USENIX Security Symposium, vol. 12, p. 2. USENIX Association, Berkeley (2003), http://portal.acm.org/citation.cfm?id=1251353.1251355
14. Beyah, R., Kangude, S., Yu, G., Strickland, B., Copeland, J.: Rogue access point detection using temporal traffic characteristics. In: Global Telecommunications Conference, GLOBECOM 2004, November-December 3, vol. 4, pp. 2271–2275. IEEE, Los Alamitos (2004)
15. Hippenstiel, R.D.: Detection Theory: Applications and Digital Signal Processing, 2nd edn. CRC Press, Boca Raton (2002)
16. Kao, K.F., Liao, I.E., Li, Y.C.: Detecting rogue access points using client-side bottleneck bandwidth analysis. Computers and Security 28(3-4), 144–152 (2009), http://www.sciencedirect.com/science/article/B6V8G-4V353XY-1/2/0e2cd909933fa11ae60a0417d16d0faa
17. Ma, L., Teymorian, A.Y., Cheng, X.: A Hybrid Rogue Access Point Protection Framework for Commodity Wi-Fi Networks. In: 2008 IEEE INFOCOM - The 27th Conference on Computer Communications, pp. 1220–1228. IEEE, Los Alamitos (2008), http://dx.doi.org/10.1109/INFOCOM.2008.178
18. Mano, C.D., Blaich, A., Liao, Q., Jiang, Y., Cieslak, D.A., Salyers, D.C., Striegel, A.: Ripps: Rogue identifying packet payload slicer detecting unauthorized wireless hosts through network traffic conditioning. ACM Trans. Inf. Syst. Secur. 11, 2:1–2:23 (2008), http://doi.acm.org/10.1145/1330332.1330334
19. Schulman, A., Levin, D., Spring, N.: CRAWDAD data set umd/sigcomm2008 (March 2, 2009), crawdad.cs.dartmouth.edu/umd/sigcomm2008 (March 2009)
20. Shetty, S., Song, M., Ma, L.: Rogue access point detection by analyzing network traffic characteristics. In: Military Communications Conference, MILCOM 2007, pp. 1–7. IEEE, Los Alamitos (2007)
21. Song, Y., Yang, C., Gu, G.: Who is peeping at your passwords at starbucks?; to catch an evil twin access point. In: 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 28- July 1, pp. 323–332 (2010)
22. Wald, A.: Sequential Analysis. Wiley, Chichester (1959)

23. Watkins, L., Beyah, R., Corbett, C.: A passive approach to rogue access point detection. In: Global Telecommunications Conference, GLOBECOM 2007, pp. 355–360. IEEE, Los Alamitos (2007)
24. Wei, W., Wang, B., Zhang, C., Kurose, J., Towsley, D.: Classification of access network types: Ethernet wireless lan, adsl, cable modem or dialup? In: Proceedings IEEE of INFOCOM 2005 24th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 1060–1071 (March 2005)
25. Wei, W., Jaiswal, S., Kurose, J., Towsley, D.: Identifying 802.11 traffic from passive measurements using iterative bayesian inference. In: Proc. IEEE INFOCOM (2006)
26. Wei, W., Suh, K., Wang, B., Gu, Y., Kurose, J., Towsley, D.: Passive online rogue access point detection using sequential hypothesis testing with tcp ack-pairs. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC 2007, pp. 365–378. ACM, New York (2007), `http://doi.acm.org/10.1145/1298306.1298357`
27. Xie, G., He, T., Zhang, G.: Rogue access point detection using segmental tcp jitter. In: Proceeding of the 17th International Conference on World Wide Web, WWW 2008, pp. 1249–1250. ACM, New York (2008), `http://doi.acm.org/10.1145/1367497.1367750`
28. Yin, H., Chen, G., Wang, J.: Detecting protected layer-3 rogue aps. In: Fourth International Conference on Broadband Communications, Networks and Systems, BROADNETS 2007, pp. 449–458 (September 2007)

Investigation of Signal and Message Manipulations on the Wireless Channel

Christina Pöpper, Nils Ole Tippenhauer, Boris Danev, and Srdjan Capkun

Department of Computer Science, ETH Zurich, Switzerland
{poepperc,tinils,bdanev,capkuns}@inf.ethz.ch

Abstract. We explore the suitability of Dolev-Yao-based attacker models for the security analysis of wireless communication. The Dolev-Yao model is commonly used for wireline and wireless networks. It is defined on abstract messages exchanged between entities and includes arbitrary, real-time modification of messages by the attacker. In this work, we aim at understanding and evaluating the conditions under which these real-time, covert low-energy signal modifications can be successful. In particular, we focus on the following signal and message manipulation techniques: symbol flipping and signal annihilation. We analyze these techniques theoretically, by simulations, and experiments and show their feasibility for particular wireless channels and scenarios.

Keywords: Wireless Security, Adversarial Interference, Signal Manipulation.

1 Introduction

In wireless radio communications, message transmissions from a sender to one or several receivers take place over the wireless channel. Given that this channel is an open and shared medium, the communication is inherently exposed to threats related to eavesdropping and intentional interference. The security analysis of wireless systems usually evaluates these intrinsic threats with respect to specific application and system properties (e.g., mobility, device complexity). As a result, a range of attacker models and corresponding assumptions arise in practical evaluations.

Certain attacker models only consider passive (eavesdropping) attacks [36,10]. Others are restricted to denial-of-service (DoS) jamming attacks in which the receiver is precluded from retrieving and decoding the signal transmitted by the sender, e. g., in military [19,20] and increasingly in civilian [14,38,35] contexts. In stronger attacker models, the attacker does not only have the ability to jam (i. e., block) the original transmission, but she can also insert her own self-composed or replayed signals (insertion/ pollution attack) [1,32,28,26]. The attacker usually achieves this by either transmitting a signal with significantly more power, which "overshadows" the original transmission (this was, e. g., reported for GPS signals in [32] and for wireless access points in [26]) or by blocking the legitimate signal by jamming and then inserting her own signal at another time or on another frequency channel (e. g., demonstrated for WLAN in [28]). In both cases, in a successful attack, the receivers get deceived into receiving the inserted signal of the attacker instead of the original signal.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 40–59, 2011.

The strongest attacker models (e. g., in [24, 25, 11, 16]) adhere to a Dolev-Yao [5] model, in which the attacker has the capability to eavesdrop, modify, compose, and (re)play any messages transmitted and received by authorized devices. In this model, in addition to eavesdropping and insertion, the attacker can fully [24] or partially [25, 11, 16] modify and annihilate signals at the receiver's antenna.

Since attacker models are the foundation of the security analysis of any system, they should be based on a realistic assessment of the system vulnerabilities and attacker capabilities. Weaker attacker models usually underestimate the threats because they do not consider the full set of techniques that may be available to a determined attacker. For example, any jamming detection based on the energy observed on the channel could be circumvented if the attacker is using low-energy signals that corrupt only the message preamble; many standard receivers would not be able to decode the message, although the data part of the message would remain unchanged. On the other hand, the strongest attacker models are often not motivated by practical considerations. For example, Dolev-Yao based models will allow the attacker to transfer information to remote locations instantaneously while this is not realistic [22]. Although designing a system with an overestimation of the attacker capabilities does not harm the security of the system, it may complicate the proposed solutions and create unnecessary overhead on the communication or make the hardware setup more costly.

To investigate the suitability of different attacker models for wireless communication, in this work we explore the basic techniques for wireless signal (message) manipulations and investigate their assumptions and practical realization. We first categorize physical-layer techniques available to strong attackers and show how they affect the received message at the logical layer. We then focus on techniques that allow the attacker to achieve covert, low-energy manipulations during the signal transmission. More specifically, we investigate *symbol flipping* attacks, by which the attacker can change symbols of the transmitted message and thus attack the message integrity, and *signal annihilation* attacks, by which the attacker suppresses the sender's signal at the receiver.

In short, our main contributions are as follows:

- We categorize adversarial interference in wireless transmissions and compare it to the capabilities of a Dolev-Yao attacker.
- We present a theoretical model to describe symbol flipping attacks.
- We explore the effectiveness of symbol flipping and signal strength manipulation (annihilation) attacks in simulations and validate our findings experimentally using USRP [6] devices.

The remainder of this paper is organized as follows: In Section 2, we describe related work and state the problem that we tackle. In Section 3, we define and classify adversarial interference in wireless communications and analyze its mapping to the Dolev-Yao attacker model. We analyze symbol flipping attacks and the conditions for their success theoretically in Section 4. In Section 5, we evaluate the feasibility of symbol flipping and signal strength manipulation attacks by simulations and experiments. We discuss implications of our findings in Section 6 and conclude the paper in Section 7.

2 Related Work and Problem Statement

2.1 Related Work on Signal Manipulations

Wireless communication jammers have been widely analyzed and categorized in terms of their capabilities (e.g., broadband, narrowband, tone) and behavior (e.g., constant, sweep, random, reactive) [13,19,38]. Jammer models used in prior works [13,31,38] cover the interference with transmissions by signal jamming and dummy packet or preamble insertions. The authors of [30,25] explicitly consider signal modification, overshadowing, and symbol flipping in their respective attacker models and propose solutions that achieve jamming- (and overshadowing-)resistant communication. However, neither of the mentioned works investigates the *feasibility* of such attacks.

When signals collide, the stronger one may survive regardless of the kind of signal. Whitehouse et al. [33] propose a technique for sensor networks to detect and recover packets from (unintended) collisions taking advantage of the *capture* effect, whereby the packet with the stronger signal strength can be received in spite of a collision. [23] quantifies the SINR conditions under which the capture effect can be observed. Another example is GPS tampering by *overriding* [3]; the success of the attack is based on the fact that GPS receivers tune in to the strongest (four) GPS signals available. The authors of [9] point out that GPS signals can also be subject to spoofing and flipping attacks that succeed with a certain probability. While they do not derive these probabilities, our findings in the experimental evaluation are conform to their numbers.

The authors of [1,21] show that for low-power wireless devices (sensor motes) predictable and deterministic symbol corruptions (flippings) are hard to achieve by mote-class attackers. In these papers, the authors describe the effect of intentional interference with a signal transmission in terms of the predictability of bit and packet corruptions. Our work is related to this, however, we do not restrict our investigations to customary sensor mote attackers but explore the underlying principles and conditions under which message manipulations and signal annihilation can be successful.

2.2 Problem Statement

In this paper, we address the following problem: *How can an attacker actively interfere with ongoing wireless transmissions and which success rates can be achieved?* This question aims at exploring the feasibility of real-time manipulations of signals (messages) in which the attacker tampers with the signals *while they are being transmitted*.

In particular, we will practically investigate two types of attacks that may allow the attacker to (i) modify signals and the data content of messages during their transmission or (ii) disrupt the communication in a covert, hard-to-detect manner. We briefly outline these two types of attacks:

Symbol flipping targets the data payload or the packet preamble, trying to modify the packets at the receivers. Flipped symbols in the preamble prevent both the decoding of the data payload and the detection of the jamming attack on standard devices because they do not allow the receiver to detect the beginning of the message header or result in a misinterpretation of the constellation diagram. Successful preamble corruption does not require that *specific* symbols are flipped. Although integrity measures (e. g., checksums and CRCs) may identify symbol flippings, they will not succeed if the attacker can deterministically change bits of the CRC to conceal her modifications. We note that a number of wireless protocols do not employ integrity protection measures or do not enforce them cryptographically (such as WEP 802.11, civilian GPS, or the RFID-M1 communication protocol).

Signal annihilation can be achieved when the attacker's signal creates destructive interference with the sender's signal at the receiver (similar to multipath interference [29]). In this case, the sender's signal gets attenuated and may be annihilated at the receiver; hence the receiver cannot detect an ongoing transmission. This attack can be performed without prior knowledge of the message content and is difficult to prevent without resorting to hardware modifications of the transceivers. Signal attenuation and amplification attacks are also crucial to the security of RSSI-based localization [8].

The investigation of the research question above examines realistic attacker capabilities that are assumed in a number of works on wireless communication without exploration, e. g. in [24,25,11,16]. We therefore see our work as an important building block for constructing realistic threat models and appropriate countermeasures. This is specially relevant in view of the recent development of tools that practically interfere with ongoing transmissions and show the feasibility of real-time reactive radio interference, such as [34].

3 Classifying Wireless Attacks

Attacker models used in the security analysis of wireless protocols are often defined on an abstract layer. They usually consider effects—such as deletion and modification—that an attacker can have on the reception of *messages* at the receiver. We will explain such an attacker model in more details in Section 3.1.

In the context of wireless systems, message-based attacker models have been adopted in a number of works, e. g., in [25,24,22,11,16,15]. In these works, the attacker is usually assumed to be able to eavesdrop, insert, modify, replay, delay, or delete any *signal* being transmitted on the wireless channel. Since messages are defined on the abstract, logical level of bits and signals comprise also the physical characteristics of the transmission, it is not clear that abstract network protocol attacker models can be applied directly to wireless communications.

In the following, we summarize message-layer effects commonly used in abstract attacker models and identify signal-layer effects which cause them (Section 3.1). To model these effects, we define *adversarial interference* as attacks in which the attacker transmits her own signals to the channel and we investigate how this can be captured in existing physical-layer reception models (Section 3.2). We then formally classify attacks based on adversarial interference (Section 3.3).

3.1 Signal Manipulations and Effects on Messages

In attacker models such as the Dolev-Yao (DY) model [5], the attacker's capabilities include eavesdropping and the arbitrary modification and deletion of messages transmitted by legitimate entities as well as the composition and insertion of the attacker's

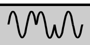
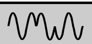
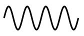
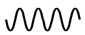
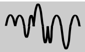
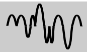
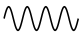
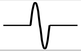
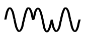
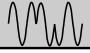
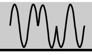
Signals			Effects	
target	attacker's	resulting	signal layer	message layer
—			signal creation/replay	insertion
		—	**annihilation**	deletion
			noise jamming	
			**symbol flipping**	modification
			overshadowing	

Fig. 1. Examples of signal-layer manipulations and their effects on the message layer. Signals can, e. g., be annihilated or jammed, their signal strength can be modified, and their amplitude, phase or frequency can be changed to influence their demodulation. Message-layer effects can in general be caused by multiple signal-layer effects. Signal-layer effects in bold will be investigated in Section 4.

own messages at the receivers. In the following, we list the effects that a DY-like attacker is assumed to be capable of achieving at the victim's receiver and give examples of how a wireless attacker can cause these effects on the signal layer (see Figure 1).

Message Eavesdropping: The attacker can observe all messages sent to one or more receivers. In a wireless network, on the signal layer, an attacker can observe the channel and record all signals with own antennas. The interpretation of the received signals as messages may require secrets such as the used spreading codes, which might not be available to the attacker. In some scenarios, the attacker can be restricted in the number of channels that she can simultaneously monitor [4,37,25].

Message Insertion and Replay: The attacker acts like a legitimate member of the network, and as such she can insert messages or replay previously received messages. In wireless networks, this is a reasonable assumption on both the message and signal layer because the attacker can construct own messages and transmit the corresponding signals and she can also replay previously received signals and messages. Restrictions on this can exist, e. g., in spread spectrum communication using secret sequences shared between the sender and receivers [19].

Message Deletion: The attacker is in control of the network and can prevent the reception of messages. To achieve this effect on a wireless channel, several methods can be used on the signal layer. These methods include jamming of complete messages using higher energy noise signals as well as jamming only the message preamble to hide it from the receiver. A more covert attack is to annihilate the signal by sending inverse signals to the receiver. While these methods all have the same effect on the message layer, i. e., the deletion of the message, in each method the receiver will capture different signals on the (physical) signal layer.

Message Modification: The attacker can modify the messages obtained by the receivers. To modify wireless messages, the attacker can either change the signals during their transmission by adding own signals—thus influencing the demodulation of single symbols (*symbol flipping*)—or prevent the receiver from obtaining the original message (message deletion) and then insert a modified version of the message.

Signal-layer manipulations such as attenuation and amplification are not directly reflected in abstract attacker models. If the signal amplitude of the message is increased or decreased (within a certain threshold), the data content on the message layer will remain unchanged with most modulation schemes. However, the amplitude change can be relevant for a number of wireless protocols, e. g., RSSI-based localization [8] for which signal strength amplification and attenuation constitute an attack.

3.2 Model of Adversarial Interference

In this section, we present a model to describe the possible effects that signal-layer manipulations can have on the message layer.

We start with a brief system description and introduction of the notation used. We consider a sender A and a particular receiver B that are able to communicate over a wireless radio link. Wireless transmissions are characterized by the messages (data) being transmitted and the physical signals used to transmit the data. The physical signals are determined by the used modulation scheme, power levels, etc. Let $s(t)$ be the signal transmitted by A; $s(t)$ is the result of the encoding process at A that packages, error-encodes, and modulates a data sequence $\mathbf{S}_A$. Let $\hat{s}(t)$ be the signal that B receives under unintentional interference (including noise and signal attenuation). In order to receive the message, B applies a function $d(\cdot)$ to demodulate $\hat{s}(t)$; it outputs the demodulated symbol sequence $\mathbf{S}$. If B does not detect the message on the channel[1], the demodulation results in the empty symbol sequence $\emptyset$.

Let $j(t)$ be the signal transmitted by an attacker J and $\hat{j}(t)$ be the corresponding signal received at B. The demodulation of $\hat{j}(t)$ at B results in $d(\hat{j}(t)) = \mathbf{S}_J$. We now define adversarial interference as follows:

Definition 1. *Let $\hat{o}(t)$ be the superposition of two signals $\hat{s}(t)$ and $\hat{j}(t)$ at B. Let $\mathbf{S}_A = d(\hat{s}(t))$, $\mathbf{S}_A \neq \emptyset$. Let $\mathbf{S}_B = d(\hat{o}(t))$ at B. The transmission of $j(t)$ is an* interference attack *if $\mathbf{S}_B \neq \mathbf{S}_A$ or if $P_{\hat{o}}(t) \neq P_{\hat{s}}(t)$, where $P_{\hat{o}}(t)$ and $P_{\hat{s}}(t)$ are power metrics for $\hat{o}(t)$ and $\hat{s}(t)$.*

This definition implies that, in a successful interference attack, the attacker changes the message symbols and/or the signal power of the original signal $\hat{s}(t)$. We note that $\hat{s}(t)$ and $\hat{j}(t)$ must overlap in time and frequency band at B for the attack to succeed.

The defined signal-layer manipulations can be integrated in existing physical reception models for wireless communications, see Appendix A. This integration supports and facilitates the identification of different types of attacks.

3.3 Classification

Given the considerations above, we can identify the following types of attacks based on adversarial interference. We also map them to message-layer effects, see Figure 1. We use the notation as introduced in Definition 1.

[1] The detection of a signal may, e.g., not be triggered if the signals power lies below a threshold or if its preamble does not match the used protocol.

- **Symbol flipping:** One or more symbols of $\mathbf{S}_A$ are flipped. $\hat{o}(t)$ gets demodulated into a valid sequence $\mathbf{S}_B$, $\mathbf{S}_B \neq \mathbf{S}_A$ and $\mathbf{S}_B \neq \mathbf{S}_J$. $P_{\hat{o}}(t) \approx P_{\hat{s}}(t)$ for the message duration.
- **Amplification:** $\hat{j}(t)$ amplifies $\hat{s}(t)$ at B. $\mathbf{S}_B = \mathbf{S}_A$. $P_{\hat{o}}(t) > P_{\hat{s}}(t)$ for the entire signal $\hat{o}(t)$.
- **Attenuation:** $\hat{j}(t)$ attenuates $\hat{s}(t)$ at B. $\mathbf{S}_B = \mathbf{S}_A$. $P_{\hat{o}(t)} < P_{\hat{s}(t)}$ for the entire signal $\hat{o}(t)$.
- **Annihilation:** $\hat{o}(t)$ falls below the noise level. $\hat{s}(t)$ is removed at B by a (sufficiently close) inverse jamming signal $\hat{j}(t) \approx \hat{s}^{-1}(t)$. $\mathbf{S}_B = \emptyset$. $P_{\hat{o}}(t) \ll P_{\hat{s}}(t)$ for the entire signal $\hat{o}(t)$.
- **Overshadowing:** $\hat{s}(t)$ appears as noise in the much stronger signal $\hat{j}(t)$. $\mathbf{S}_B = \mathbf{S}_J$. $P_{\hat{o}}(t) \gg P_{\hat{s}}(t)$ for the entire signal $\hat{o}(t)$.
- **Noise jamming:** $\hat{j}(t)$ is noise to prevent B from detecting the message, thus blocking its reception. $\mathbf{S}_B = \emptyset$. $P_{\hat{o}}(t) \gg P_{\hat{s}}(t)$ for the entire signal $\hat{o}(t)$.

Amplification, attenuation, and annihilation can be denoted as *signal strength modification* attacks. From the attacker's point of view, a similar action is performed in all attack cases listed above, namely the transmission of a signal $j(t)$. What differs are the type and strength of $j(t)$ and its dependency on $s(t)$: While $j(t)$ is independent of $s(t)$ in overshadowing and noise jamming attacks, the attacker uses $s(t)$ to construct $j(t)$ in signal strength modification attacks and both $s(t)$ and $o(t)$ in symbol modification attacks, where $o(t)$ is the signal that the attacker wants B to receive.

We note that, according to Definition 1, attacks in which the attacker jams the original signal and inserts an adversarial signal with a shift in time or frequency band (e.g., exploiting the channel structure of WLAN 802.11 signals by transmitting on separate frequencies [28]) are a combination of adversarial interference and a parallel insertion/pollution attack [25, 12].

4 Theoretical Analysis of Symbol Flipping

In this section, we focus on symbol modification attacks and present our model of symbol flipping. We restrict our considerations to single carrier modulations and reason about flipping on the level of symbols. We distinguish symbol flipping attacks according to the attacker's goal. $\mathbf{S}_A$, $\mathbf{S}_B$, and $j(t)$ are as in Definition 1.

Definition 2. *A* deterministic symbol flipping attack *has the goal to make B demodulate $\mathbf{S}_B = \mathbf{S}_T$, where the symbol sequence $\mathbf{S}_T \neq \mathbf{S}_A$ has been defined by the attacker before the transmission of $j(t)$. A* random symbol flipping attack *targets at modifying any symbol(s) of $\mathbf{S}_A$ such that $\mathbf{S}_B \neq \mathbf{S}_A$.*

In the following, we denote the symbols of the sequence $\mathbf{S}_A$ also as *target symbols*. Deterministic symbol flipping requires a-priori knowledge about the target symbols, i.e., about the parts of a message that are to be flipped. We next investigate how to achieve successful symbol flipping.

The way multiple signals get superimposed depends on their modulations (including signal power, phase shifts, etc.). We consider linear digital modulation schemes such

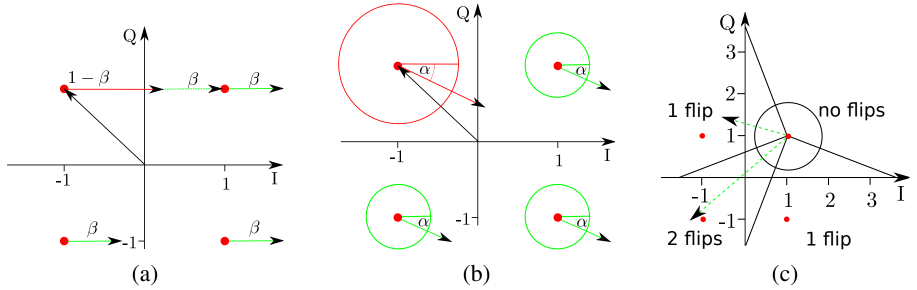


Fig. 2. (a) Effect of imperfect baseband alignment of the flipping symbol w.r.t. the target QPSK symbol. Given a delay βT_s, the fraction β of the energy will be added to the next symbol. (b) Effect of the relative carrier phase offset α between the target and the flipping signals. The phase offset rotates the energy contribution of the flipping signal. As all flipping symbols have the same carrier phase offset, all energy contributions get rotated. (c) Depending on the signal energy and rotation, different constellation regions can be reached by symbol flipping.

as 2-PAM, 4-QAM (QPSK), and 16-QAM, which divide the constellation space into decision regions with varying sizes and shapes. For QPSK (see Figure 2a), the decision regions are separated by the axes of the IQ-plane. Given a modulation scheme and the received signal vector $\hat{s}$, the decision element in the receiver's decoder outputs the constellation point with the minimum Euclidean distance (ML detection) [20]. Moving a signal vector $\hat{s}$ in the constellation implies a change in signal power (distance from the origin of the constellation diagram) and/or a changed angular phase of the signal. For QPSK, we define two ways of flipping a symbol (this will later matter for our simulations):

Definition 3. *For QPSK, a* short transition *denotes the shift of a symbol vector into an adjacent constellation region (ideally parallel to the I- or Q-axis). A* long transition *denotes a diagonal shift into the opposite constellation region.*

In Gray-encoded constellations, a short transition changes one bit of a symbol and a long transition both bits of the symbol. Such transitions can be caused by adding a QPSK symbol with modified carrier phase alignment and enough power. If this symbol temporally overlaps with one or more target symbols, we call it *flipping symbol*.

In practice, three factors influence the result of a symbol flipping attack: (i) the baseband alignment of the sender's and attacker's symbols, (ii) the relative carrier phase offset of the attacker's signal, and (iii) the energy of the attacker's symbol.

(i) The *baseband alignment* of the flipping symbols determines the amount of energy that will not be contributed to the target but to the *neighboring* symbols in the message. Here, we assume a sequence of flipping symbols that are all delayed by the same time βT_s, where T_s is the symbol duration. Then, a fraction β of the energy will influence the decoding of the following symbol. Figure 2a visualizes the effect of the baseband symbol alignment and shows the effect on the next target symbol: the misaligned flipping symbol, represented by the vector (2,0), will affect the current symbol $(-1{,}1)$ with $1 - \beta$ and the following symbol with β. A similar effect may occur to the current symbol due the prior flipping symbol. We will analyze the required baseband alignment by simulations and experiments in Section 5.

(ii) In addition to the effect of the baseband alignment, the relative *carrier phase offset* α of the flipping signal with respect to the target signal will rotate the energy contribution of the signal. As all flipping symbols have the same carrier phase offset, all energy contributions get rotated in the same way, see Figure 2b.

(iii) For short transitions, the minimum required *signal energy* (for exact carrier phase and baseband alignment) is a factor $1/\sqrt{2}$ of the energy of the target signal; for long transitions, at least as much energy as in the target signal is required. Figure 2c gives an example of a short transition (one bit changed) and a long transition (two bits changed). Based on our model, we can predict the probability of successful symbol flipping for a random carrier phase offset. Figure 3 displays the analytical flipping probabilities depending on the relative signal energy, derived using trigonometrical functions.

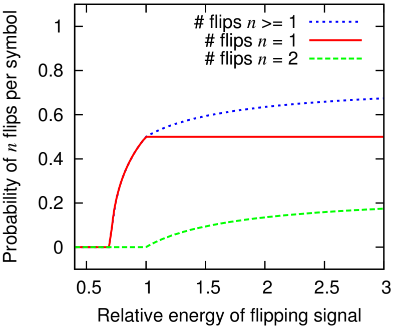


Fig. 3. Analytical probability of successful symbol flipping for random carrier phase alignment and perfect baseband alignment, depending on the relative signal energy

5 Simulation and Experimental Evaluation

In this section, we explore the conditions for successful symbol flipping and signal annihilation (as defined in Section 3.3) under an attacker as presented in Section 5.1. We verify our theoretical symbol flipping model of Section 4 by simulations in Matlab [27] in Section 5.2. The main results are then validated using signals captured from recorded wireless communications in Section 5.3. We also explore signal annihilation and attenuation by experiments with wireless devices in Section 5.4.

5.1 Simulation Setup and Attacker Model

Simulation Setup. For our simulation and experimental evaluation of symbol flipping and annihilation, we focus on QPSK modulation due to its widespread use (e.g., in 802.11 and Bluetooth 3.0). We implemented an 802.11 digital QPSK modem with an AWGN channel. The matched filter g(t) was implemented by a root raised cosine filter. The carrier frequency was fixed to $f_c = 2.4$ GHz with $\phi_1(t) = \cos(2\pi f_c t)$ and $\phi_2(t) = -\sin(2\pi f_c t)$ for the I and Q channels, respectively. Figure 7 in Appendix B displays the simulation setup.

Our simulations are based on 1000 random QPSK symbols that we use to create the flipping symbols. For long transitions, we invert each symbol and double its amplitude; for short transitions we combine the inverted symbol with its complex-conjugate. We use the following notations: The original (target) symbol is denoted by $\mathcal{T}$, the short transition flipping symbol by $\mathcal{S}$, and the long transition flipping symbol by $\mathcal{L}$. $\mathcal{R}$ is a flipping symbol with random carrier phase offset and same power as $\mathcal{L}$.

Attacker Model. In our simulations, we focus on two attacker types: (a) a *strong attacker* with perfect carrier phase alignment, able to predict which symbols are going to be sent, and therefore using perfect flipping signals; (b) a *weak attacker* without carrier phase alignment and therefore random flipping signals. The goal of the strong attacker is to perform a *deterministic* symbol flipping attack, while the weak attacker tries to perform a *random* symbol flipping attack (see Definition 2). In order to achieve their goals, the attackers follow these strategies:

- The strong attacker uses a short transition flipping signal $\mathcal{S}$ to flip a specific bit of a target symbol. To flip both bits of the symbol, she uses a (more powerful) long transition flipping signal $\mathcal{L}$. In both cases, the flipping signals have *perfect carrier phase alignment* with the target signal.
- The weak attacker uses flipping symbols $\mathcal{R}$ with the same power as $\mathcal{L}$ but with *random carrier phase* (rotating the signal vector in the IQ-plane) with respect to the target signal.

We note that a short transition by a strong attacker is successful only if the *intended* bit was flipped, while for a weak attacker the flipping of any of the two bits (or both bits) of the symbol are considered a success.

5.2 Simulated Modification of Modulated Signals

Following our model from Section 4, we will now predict the effects of varying power, carrier phase offset, and baseband offset of the flipping signal. Finally, we will predict their impact on annihilation attacks.

Power of the Flipping Signal. According to our model, the power of the flipping signal needs to be greater than a fraction $1/\sqrt{2}$ of the target signal. Flipping in this case is only successful if the flipping signal has the optimal phase (e.g., shifts the symbol (1,1) into the direction of (1,-1)). For random phases, the power of the flipping signal must be higher.

Figure 4a displays the influence of the relative power of the flipping signal on the probability to flip QPSK symbols (for random carrier phases of the flipping signal and perfect baseband symbol alignment). The plot shows the probability of a random symbol flip for a weak attacker and a deterministic flip for a strong attacker, for an SNR level of 20 dB. The weak attacker has no carrier phase synchronization and thus no control over the angle of the flipping signal. The strong attacker uses a flipping signal with perfect phase synchronization.

The simulation confirms that, for a low noise level (high SNR), the power $P_{\mathcal{S}}$ of a short transition symbol must satisfy $P_{\mathcal{S}} \geq \frac{P_{\mathcal{T}}}{\sqrt{2}}$, where $P_{\mathcal{T}}$ is the power of the target symbol, in order to change a single bit of the symbol. The weak attacker's probability to flip a single bit converges towards 50 % for $P_{\mathcal{R}} \geq P_{\mathcal{T}}$ and her chance to flip both bits of a symbol towards 25 % for $P_{\mathcal{R}} \to \infty$ (not shown in Figure 4a).

Carrier Phase Offset for Symbol Flipping. The carrier phase offset between the target signals and the flipping signals at the receiver is hard to control for the attacker. This

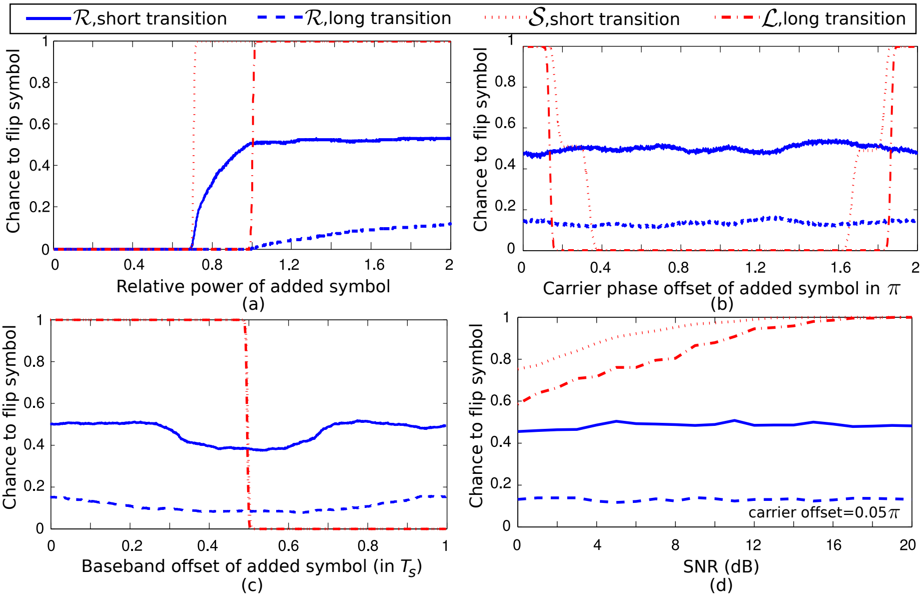


Fig. 4. Influence of the flipping symbol on the probability to change a QPSK symbol using a random-phase flipping symbol $\mathcal{R}$ (weak attacker) or a perfect short/long flipping symbol ($\mathcal{S}/\mathcal{L}$) (strong attacker). (a) Influence of the relative power of the flipping symbol. (b) Influence of the carrier phase offset of the flipping signal. (c) Influence of the baseband offset (relative to the symbol duration T_s) of the flipping symbol. (d) Influence of the SNR for a fixed carrier phase offset of 0.05π.

is the main reason why symbol modification attacks are difficult to conduct even with perfect advance knowledge of the data to be sent. The effect of a constant carrier phase offset under noise is displayed in Figure 4b for $P_\mathcal{R} = P_\mathcal{L} = 2P_\mathcal{T}$, $P_\mathcal{S} = \sqrt{2}P_\mathcal{T}$, and 20 dB SNR.

Simulations without noise show that a strong attacker must hit the carrier phase within about $13.5\,\%$ of the carrier phase duration to flip both bits of the target symbol (long transition). Short transitions for the strong attacker require less carrier phase precision, the tolerance is $25\,\%$. The carrier phase offset has no impact for a weak attacker because she uses flipping signals with random phase; the carrier phase offset does therefore not influence her probability to flip bits.

If the attacker does not synchronize correctly to the sender's carrier *frequency*, this will make it almost impossible for her to predict the optimal carrier phase alignment for the flipping symbols. However, the attacker must synchronize the carrier frequency of her flipping signals only once to a target transmission, which will then result in the same carrier phase offset for all flipping signals with respect to the target transmission.

Baseband Offset for Symbol Flipping. A weak attacker might have problems aligning the flipping symbols correctly to the target symbols. This has the effect that the energy of the flipping symbol will not only contribute to the target symbol but also influence neighboring symbols (see Section 4). We evaluated the impact of this baseband offset by simulations, see Figure 4c. We set $P_{\mathcal{R}} = P_{\mathcal{L}} = 2P_{\mathcal{T}}$ and $P_{\mathcal{S}} = \sqrt{2}P_{\mathcal{T}}$ as before for the power of the flipping signals and 20 dB SNR. The simulation results show that the probability for a *weak* attacker to flip a bit degrades smoothly. In Figure 4c, her probability does not converge to zero for a baseband misalignment of one symbol duration (T_s) because the following symbol is flipped (which is a success for the weak attacker). The strong attacker has a probability of 1 to flip both bits of a symbol if the baseband offset is smaller than 50 % (with sufficiently high SNR).

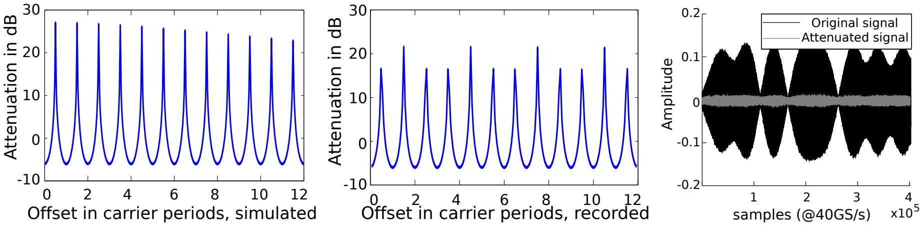
Similarly to the carrier frequency offset, an offset in the baseband symbol *rate* between the attacker and the sender will lead to changing baseband offsets for a sequence of flipping symbols, which will not influence the weak attacker but make deterministic attacks for the strong attacker almost impossible.

Influence of the SNR. We next investigate the influence of the Signal-to-Noise-Ratio on the attacker's probability to perform successful symbol flipping. Intuitively, the higher the SNR at the receiver, the better a strong attacker can predict the effects of the flipping attack. To demonstrate the effect of the SNR on the attacker's success probability, we ran a simulation with $P_{\mathcal{R}} = P_{\mathcal{L}} = 2P_{\mathcal{T}}$, $P_{\mathcal{S}} = \sqrt{2}P_{\mathcal{T}}$, carrier phase offset 0.05π, and perfect baseband alignment. The results in Figure 4d show that the SNR does not influence the weak attacker, but lower SNR values require the strong attacker to have a more accurate carrier phase synchronization to flip the target.

Simulation of Signal Strength Modification. We now investigate *signal annihilation* attacks (cp. Section 3.3). For this purpose, we use the legitimate signal of the sender to attenuate the sender's signal at the receiver by destructive interference, similar to worst-case effects in multipath environments. The attacker's goal is to attenuate the overall power of the signal so that it is not detected at the receiver (instead of changing the message content). Since this attack repeats the signals transmitted by the sender, it is agnostic to the actual data content of the message; the attacker does not need to know it in advance. The repeated signal will also have the same carrier frequency as the original signal, eliminating this possible source of randomness for the attacker. To fully annihilate the original signal, the attacker's signal needs to have the same power as the sender's signal at the receiver.

Figure 5a shows the simulated signal attenuation at the receiver for variable delays between the transmitted (original) and the repeated (adversarial) signal using the simulation setup in Section 5.1 with an SNR of 30 dB. The highest attenuation of approximately 28 dB is achieved only when shifting by a delay of π and high attenuation is reached every 2π of the carrier delay. This high attenuation slightly decreases for higher offsets in carrier periods due to the resulting larger time offset between the two signals. We refer to this attack as a π-shift-attack. We note that the original signal can also be amplified instead of attenuated. This would occur when shifting by a delay of 2π and multiples of it. The original signal could be amplified by up to 6 dB.

Given that the π-shift-attack does not require demodulation or complex logic at the attacker, it can be implemented using only directional antennas and possibly an amplifier. In Section 5.4, we present a practical implementation of this attack and show that high attenuation is also possible in practice.



(a) (b) (c)

Fig. 5. Signal annihilation attack. Figures (a) and (b) depict the signal attenuation obtained by adding the same signal delayed with different carrier offsets. (a) shows the results using signals simulated in Matlab (with an SNR of 30 dB), (b) uses recorded signals (measured SNR of around 30 dB). (c) shows the practical signal attenuation obtained using our experimental carrier.

5.3 Simulated Modification of Recorded Signals

We continue our evaluation with signals transmitted over the air and recorded by an oscilloscope. This allows us to validate the simulation results of symbol flipping and signal attenuation (Section 5.2) with a non-ideal transceiver and lossy communication channel. In our experiment, we combine our digital QPSK modem with the capabilities of a universal software radio peripheral (USRP [6]). We use fully modulated messages in a frame that closely resembles the 802.11b frame specification [2] with a preamble for carrier frequency offset estimation and synchronization [17]. Figure 8 in Appendix C displays our setup for the experimental investigations in Sections 5.3–5.4.

Symbol Flipping of Recorded Signals. Our main goal of this experiment is to validate our predicted probabilities for an attacker using optimal $\mathcal{S}/\mathcal{L}$ flipping symbols to reach her goal with random carrier phase synchronization. In addition, we are interested in the chance of a weak attacker flipping any (neighboring) bits. We simulated the addition of the recorded flipping symbol with varying baseband offsets of 0, $0.25T_s$, and $0.5T_s$ and averaged carrier phase offsets between 0 and 2π. The power of the flipping symbols is $P_{\mathcal{R}} = P_{\mathcal{L}} = 2P_{\mathcal{T}}$, $P_{\mathcal{S}} = \sqrt{2}P_{\mathcal{T}}$ as in the previous simulations.

Table 1 compares the chances for successful attacks on the target symbol (T) and (unwanted) flipping of neighboring symbols (N) between the results of simulation without noise (Sim) and the findings based on our recorded signals (Recorded). We observe that the predicted probabilities for long and short transitions closely follow the probabilities computed from the recorded signals for baseband offsets of 0 and $0.25T_s$. The influence on the target and neighboring symbols only differ for an offset of $0.5T_s$. This is most likely due to the fact that the probabilities to symbol flipping at $0.5T_s$ occupy a transition region (Figure 4c) and thus can take different values in the presence of noise. Nevertheless,

Table 1. Probability of modifications of the target (T) and neighboring (N) symbols in simulated vs. recorded signals for random carrier phase offset (%)

	Baseband Offset					
	0		$0.25 \times T_s$		$0.5 \times T_s$	
Sim	T	N	T	N	T	N
$\mathcal{R}$, short	25	0	25	0	0	0
$\mathcal{R}$, long	13.5	0	9.3	0	0	49.96
$\mathcal{R}$, *any*	63.5		59.3		74.82	
Recorded	T	N	T	N	T	N
$\mathcal{R}$, short	24.3	0	25.0	0	21.5	9.7
$\mathcal{R}$, long	11.1	0	11.1	0	2.8	27.8
$\mathcal{R}$, *any*	58.3		58.3		70.8	

our main result is confirmed by the experimental evaluation: about 13 % flipping chance for long transitions and about 25 % for a short transition, both with random carrier phase offset and small baseband offset.

Signal Annihilation of Recorded Signals. We used recorded messages as described in 5.3 to simulate the effect of signal annihilation by adding time-shifted copies of the signal. The lower plot in Figure 5 shows the obtained attenuation. In comparison to the simulation with ideal signals (i.e., upper plot in Figure 5), the achieved highest attenuation was lower by few decibels. Correct demodulation at the receiver was still not possible with our implementation, hence the signal was successfully annihilated. We also observe that there are several possible carrier offsets at which this high attenuation can be achieved.

5.4 Experimental Evaluation of Signal Annihilation

The main goal of this evaluation is to estimate how accurately the carrier phase offset can be controlled and what attenuation could be achieved in real multipath environments. For this purpose, we built the experimental signal annihilation setup shown in Figure 8 (Appendix C). The setup consists of a transmitter (USRP), a receiver (oscilloscope), and two directional antennas (with a gain of 15 dBi) connected by a cable. One antenna is directed at the transmitter and the second antenna repeats the received signal towards the receiver. The USRP sends periodic signals, which are simultaneously repeated by the antennas, received at the oscilloscope, and demodulated in Matlab. To achieve signal annihilation, the amplitude and carrier phase delay of the attacker's signal must closely match the legitimate signal at the receiver. We controlled the carrier phase offset between the transmitted and repeated signals by changing the distance between the antennas. Since we used high gain directional antennas, we could also adapt the power of the repeated signal by directing the antenna away from the receiver by some degrees. For a distance of 2 m between the USRP and the receiver and an appropriate positioning of the directional antennas (approximately 1 m away from the line of sight), we achieved the predicted signal attenuation down to the noise level. Figure 5c shows the signals received at the oscilloscope with and without the two directional antennas. Our results show an attenuation of approximately 23 dB. By using a longer (1 m) cable between the directional antennas, we also verified that the resulting higher baseband offset between the transmitted and repeated signals does have a significant impact on the achieved attenuation. We note that for longer distances, the same setup would require additional amplification between the directional antennas.

5.5 Summary of Results

We evaluated the influence of carrier and baseband offsets, amplitude mismatches, and the SNR on symbol flipping, first theoretically in Section 4 and then by simulations and experiments. Our findings show that, given accurate carrier phase and baseband synchronization, deterministic symbol flipping is feasible for strong attackers.

If the attacker cannot adapt to the sender's carrier phase offset, a random offset will allow her to achieve long transitions causing deterministic symbol flippings in around 13.5 % of the cases; for a short transition, this chance reaches up to 25 % (see Table 1).

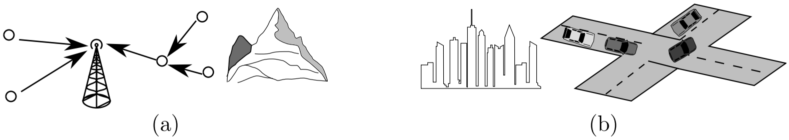


Fig. 6. Examples for wireless networks. (a) Static networks in quasi-static, quasi-free-space environments allow a strong attacker to perform deterministic signal manipulations; we thus confirm the Dolev-Yao model as an appropriate worst-case attacker model. (b) Environments with multipath effects and networks with mobile nodes suggest that deterministic, covert signal manipulations are hard to achieve—a probabilistic attacker model is more realistic.

The weak attacker aiming at changing one bit of any symbol will achieve this with a chance of 50 % (see Figure 4 and Table 1) per flipping symbol as long as her signal has enough power, regardless of the carrier phase offset and baseband offset. Since the carrier phase offset is influenced by the channel and the geometric setup of the sender, attacker, and receiver, it might be hard to exactly match the target offset in practice. We discuss the impact of this on deterministic message manipulations in Section 6.

We also predicted an attenuation of the original signal to the noise level by adding the same signal shifted by a certain carrier phase offset for realistic SNR levels (e.g., 20 dB). We reproduced the attenuation with recorded signal traces in Matlab and showed its practical feasibility in a lab environment using two directional antennas.

We discussed the use of rotated and scaled QPSK symbols as flipping signals. The use of alternative, e.g., shorter symbols of higher bandwidth, is left for future work.

6 Implications

In the previous sections, we have investigated the practicability of low-energy symbol flipping and signal annihilation attacks through simulations and experiments. We will now discuss the implications of our results in selected scenarios.

In a first scenario, we consider a wireless network with static wireless nodes and quasi-static, quasi-free-space channel properties. An example of such a network could be wireless sensor nodes deployed in rural areas, see Figure 6a. If an attacker with strong signal manipulation capabilities is allowed to access any location, she can measure distances and estimate the channel with high precision to any target node. The attacker would thus be able to achieve carrier phase synchronization and control the signal amplitude levels at the target receiver in order to flip symbols and/or annihilate transmitted signals with very high probability (for our system with non-coherent receivers). This corresponds to the model of our strong attacker (Section 5.1).

In a number of scenarios that are typical for wireless network deployments at least one of the assumptions in the above case is violated. Examples include static wireless networks in dynamic environments (e.g., urban areas) or mobile wireless networks, see Figure 6b. In both examples, wireless nodes communicate over time-varying fading channels [29]. This channel makes carrier phase synchronization and amplitude control at the target receiver very difficult (if not infeasible) for the attacker as it requires her to know the state information of the sender-receiver channels. Given that feedback signaling is typically needed for channel state information (CSI) estimation [18], it is hard to launch deterministic attacks without receiver cooperation. Failing to do so significantly reduces the probability of a strong attacker to perform deterministic short and long symbol flipping (Definitions 2 and 3) to 25% and 12.5%, respectively (in our scenario using QPSK modulation).

Furthermore, our results show that an attacker without a priori knowledge of the transmitted data has a chance of up to 75 % (see Table 1) to change *any* symbol (flip one or two bits) by adding a flipping symbol with twice the signal power. Depending on the error-correcting mechanisms employed at the receiver, this can allow the attacker to jam messages (or message preambles) in an energy-efficient way.

In summary, we draw the following conclusions: We conclude that the attacker models selected for the security analysis of wireless communication need to be chosen in accordance with the deployed network and scenario. In the worst case, the attacker can covertly and deterministically delete and manipulate messages if the wireless network deployment cannot guarantee that the channel is dynamic. These attacks would not be detected by existing energy-based jamming detection countermeasures, as they do not add significantly more energy on the channel. In this aspect, the attacker's capabilities become very close to those of the Dolev-Yao model. If a dynamic channel can be assumed, even the strongest attacker can only probabilistically delete and modify messages without risking detection by energy-based jamming detection techniques. Such a probabilistic attacker model captures dynamic time-varying channels in the sense that the carrier phase offset is likely to change between individual messages. We note that the probability with which the attacker will be successful depends on a number of system parameters, including coherency or non-coherency of the reception process of the receiver, multipath effects, etc. We leave the investigation of these settings open for future work.

7 Conclusion

In this paper, we investigated the applicability of abstract attacker models of wireline protocols in the security analysis of wireless protocols. We first categorized different types of signal-layer attacks and mapped them to the Dolev-Yao attacker model. Then we explored the feasibility of basic techniques for manipulating wireless signals and messages. We focused on symbol flipping and signal annihilation attacks that both allow covert, low-energy manipulations of signals during their transmission. Our theoretical analysis, simulations, and experiments identified their conditions for success for QPSK-modulated signals and showed their practical feasibility given quasi-static, quasi-free-space channels. Our findings confirm the need of strong attacker models (similar to Dolev-Yao's model) in specific static scenarios, but they also suggest to construct alternative, probabilistic attacker models for a number of common wireless communication scenarios.

Acknowledgments. This work was partially supported by the Zurich Information Security Center. It represents the views of the authors.

References

1. Arora, A., Sang, L.: Capabilities of low-power wireless jammers. In: IEEE Infocom Mini-conference (2009)
2. IEEE Standards Association. IEEE Standard 802.11b-1999: Wireless LAN MAC and PHY Specifications (1999), http://standards.ieee.org
3. Davidoff, S.: GPS spoofing (2008), http://philosecurity.org/2008/09/07/gps-spoofing
4. Desmedt, Y., Safavi-Naini, R., Wang, H., Charnes, C., Pieprzyk, J.: Broadcast anti-jamming systems. In: Proceedings of the IEEE International Conference on Networks, ICON (1999)
5. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
6. Ettus. Universal software radio peripheral (USRP), http://www.ettus.com
7. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory 46(2) (2000)
8. Hightower, J., Borriello, G., Want, R.: SpotON: An indoor 3D location sensing technology based on RF signal strength. Technical Report 2000-02-02, University of Washington (2000)
9. Humphreys, T.E., Ledvina, B.M., Psiaki, M.L., O'Hanlon, B.W., Kintner Jr., P.M.: Assessing the spoofing threat: Development of a portable GPS civilian spoofer. In: Proceedings of the ION GNSS International Technical Meeting of the Satellite Division (2008)
10. Jana, S., Premnath, S.N., Clark, M., Kasera, S.K., Patwari, N., Krishnamurthy, S.V.: On the effectiveness of secret key extraction from wireless signal strength in real environments. In: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom (2009)
11. Jin, T., Noubir, G., Thapa, B.: Zero pre-shared secret key establishment in the presence of jammers. In: Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). ACM Press, New York (2009)
12. Karlof, C., Sastry, N., Li, Y., Perrig, A., Tygar, D.: Distillation codes and applications to DoS resistant multicast authentication. In: Proceedings of the Network and Distributed Systems Security Symposium, NDSS (2004)
13. Li, M., Koutsopoulos, I., Poovendran, R.: Optimal jamming attacks and network defense policies in wireless sensor networks. In: Proceedings of the IEEE Conference on Computer Communications, InfoCom (2007)
14. Lin, G., Noubir, G.: On link layer denial of service in data wireless LANs: Research articles. Wireless Communications & Mobile Computing 5(3), 273–284 (2005)
15. Liu, A., Ning, P., Dai, H., Liu, Y.: Defending DSSS-based broadcast communication against insider jammers via delayed seed-disclosure. In: Proceedings of Annual Computer Security Applications Conference, ACSAC (2010)
16. Liu, Y., Ning, P., Dai, H., Liu, A.: Randomized differential DSSS: Jamming-resistant wireless broadcast communication. In: Proceedings of the IEEE Conference on Computer Communications, InfoCom (2010)
17. Oppenheim, A.V., Schafer, R.W., Buck, J.R.: Discrete-Time Signal Processing, 2nd edn. Prentice-Hall Signal Processing Series (1998)
18. Iserte, A.P.: Channel state Information and joint transmitter-receiver design in multi-antenna systems. PhD thesis, Polytechnic University of Catalonia (2005)
19. Poisel, R.A.: Modern Communications Jamming Principles and Techniques. Artech House Publishers, Boston (2006)
20. Poisel, R.A.: Foundations of Communications Electronic Warfare. Artech House Publishers, Boston (2008)

21. Sang, L., Arora, A.: Capabilities of low-power wireless jammers. Technical Report OSU-CISRC-5/08-TR24, The Ohio State University (2008)
22. Schaller, P., Schmidt, B., Basin, D., Čapkun, S.: Modeling and verifying physical properties of security protocols for wireless networks. In: Proceedings of the IEEE Computer Security Foundations Symposium (2009)
23. Son, D., Krishnamachari, B., Heidemann, J.: Experimental study of concurrent transmission in wireless sensor networks. In: Proceedings of the ACM Conference on Networked Sensor Systems, SenSys (2006)
24. Strasser, M., Danev, B., Čapkun, S.: Detection of reactive jamming in sensor networks. ACM Transactions on Sensor Networks 7, 16:1–16:29 (2010)
25. Strasser, M., Pöpper, C., Čapkun, S., Čagalj, M.: Jamming-resistant Key Establishment using Uncoordinated Frequency Hopping. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, S&P (2008)
26. Symantec. Securing enterprise wireless networks. White Paper (2003)
27. The MathWorks, Inc. Matlab – a numerical computing environment, www.mathworks.com
28. Tippenhauer, N.O., Rasmussen, K.B., Pöpper, C., Čapkun, S.: Attacks on Public WLAN-based Positioning. In: Proceedings of the ACM Conference on Mobile Systems, Applications and Services, MobiSys (2009)
29. Tse, D., Viswanath, P.: Fundamentals of wireless communication. Cambridge University Press, Cambridge (2005)
30. Čagalj, M., Hubaux, J.-P., Čapkun, S., Rengaswamy, R., Tsigkogiannis, I., Srivastava, M.: Integrity (I) Codes: Message Integrity Protection and Authentication Over Insecure Channels. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, S&P (2006)
31. Čagalj, M., Čapkun, S., Hubaux, J.-P.: Wormhole-based antijamming techniques in sensor networks. IEEE Transactions on Mobile Computing 6(1), 100–114 (2007)
32. Warner, J.S., Johnston, R.G.: Think GPS Cargo Tracking = High Security? Think Again. Technical report, Los Alamos National Laboratory (2003)
33. Whitehouse, K., Woo, A., Jiang, F., Polastre, J., Culler, D.: Exploiting the capture effect for collision detection and recovery. In: Proceedings of the IEEE workshop on Embedded Networked Sensors (EmNets) (2005)
34. Wilhelm, M., Martinovic, I., Schmitt, J., Lenders, V.: Reactive jamming in wireless networks: How realistic is the threat? In: Proceedings of the forth ACM Conference on Wireless Network Security, WiSec (2011)
35. Wood, A.D., Stankovic, J.A.: Denial of service in sensor networks. IEEE Computer 35(10), 54–62 (2002)
36. Xiao, L., Greenstein, L., Mandayam, N., Trappe, W.: Fingerprints in the ether: Using the physical layer for wireless authentication. In: Proceedings of the IEEE International Conference on Communications, ICC (2007)
37. Xu, W., Trappe, W., Zhang, Y.: Channel surfing: defending wireless sensor networks from jamming and interference. In: Proceedings of the ACM Conference on Networked Sensor Systems, SenSys (2006)
38. Xu, W., Trappe, W., Zhang, Y., Wood, T.: The feasibility of launching and detecting jamming attacks in wireless networks. In: Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc (2005)

A Integration into the SINR Model

In the physical SINR model [7], the transmission from a node A is successfully received by node B under simultaneous transmissions from a set $\{I_i\}$ of transmitters if

$$\frac{P_{AB}}{N + \sum_i P_{iB}} \geq \beta_B, \tag{1}$$

where $P_{AB} = P_{\hat{s}(t)}$ and P_{iB} are the sender's and the transmitters' signal powers at B, respectively, N is the ambient noise level, and β_B is the minimum SINR (Signal to Interference plus Noise Ratio) required for successful message reception at B. The SINR model represents the reception of the original transmission $s(t)$ under concurrent signals of sufficient or insufficient power.

In order to capture adversarial interference in the SINR model, we split the overall interference into legitimate (neighboring) transmissions and interference from an attacker J. Let $P_{JB} = P_{\hat{j}(t)}$ denote J's signal power at B (originating from one or multiple collaborating attackers). In order to reflect different types of adversarial interference, we distinguish constructive and destructive interference. We denote by P^c_{JB} the fraction of P_{JB} that creates constructive interference with $\hat{s}(t)$, by P^d_{JB} the fraction of P_{JB} that creates destructive interference with $\hat{s}(t)$, and by P^n_{JB} the fraction of P_{JB} that appears as noise at B; $P^c_{JB} + P^d_{JB} + P^n_{JB} = P_{JB}$. B receives a signal of sufficient power to enable demodulation for $P_{AB} + P^c_{JB} - P^d_{JB} > 0$ if

$$\frac{P_{AB} + P^c_{JB} - P^d_{JB}}{N + \sum_i P_{iB} + P^n_{JB}} \geq \beta_B. \tag{2}$$

The left-hand side of Equation 2 is the power of the signal $\hat{o}(t)$ at B. Based on this equation, we can distinguish the following cases:

- $P^d_{JB} = P_{AB} + P^c_{JB}$:
  This attack annihilates the signal with $d(\hat{o}(t)) = \emptyset$.
- $P^n_{JB} \gg P_{AB} + P^c_{JB}$:
  This results in noise jamming with $d(\hat{o}(t)) = \emptyset$.
- $P^c_{JB} - P^d_{JB}$ is in the order of P_{AB} and P^n_{JB} does not cause a blocked message at B:
  This can modify (flip) bits in the message and we get $d(\hat{o}(t)) \neq \emptyset$ and $d(\hat{o}(t)) \neq \mathbf{S}_A$. If this happens in the packet preamble we get $d(\hat{o}(t)) = \emptyset$.
- P^c_{JB} and P^d_{JB} do not modify the demodulation result and P^n_{JB} does not block the reception at B:
  In this case, we get $d(\hat{o}(t)) = d(\hat{s}(t)) = \mathbf{S}_A$, possibly under an amplified (with $P^c_{JB} > P^d_{JB}$) or attenuated (with $P^c_{JB} < P^d_{JB}$) signal.
- $P^c_{JB} - P^d_{JB} \gg P_{AB}$ and P^n_{JB} does not cause a blocked message at B:
  In this case, B will demodulate $d(\hat{o}(t)) = d(\hat{j}(t)) = \mathbf{S}_J$, hence the attacker's message is overshadowing the message from A.

B Simulation Setup

Figure 7 shows the simulation setup used for the Matlab simulations. The modulated data symbols are passed through a matched filter g(t) (root raised cosine) and up-converted to the carrier frequency (2.4 GHz band) (ϕ(t)). The channel is simulated by adding Gaussian noise (AWGN). After sampling with rate kT_s, a Maximum Likelihood (ML) decoder outputs the decoded symbols.

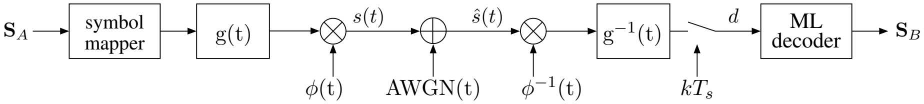


Fig. 7. Simulation setup used for the Matlab simulations

C Experimental Setup

Figure 8 shows the setup we used for our practical experiments. Symbols are generated by a QPSK modulator and form the input to a USRP that transmits them over the air. We capture the original or manipulated transmissions using an oscilloscope. We then demodulate and analyze the data.

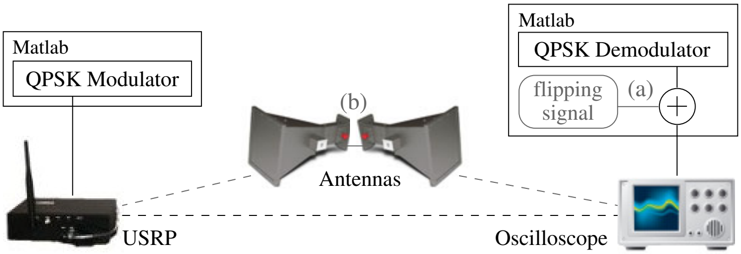


Fig. 8. Experimental setup. (a) For simulated symbol flipping of recorded signals, we add the flipping signals to the captured signals in Matlab. (b) For the experiments on signal attenuation, two antennas capture and repeat the signals.

Protecting Private Web Content from Embedded Scripts

Yuchen Zhou and David Evans

University of Virginia
{yuchen,evans}@virginia.edu

Abstract. Many web pages display personal information provided by users. The goal of this work is to protect that content from untrusted scripts that are embedded in host pages. We present a browser modification that provides fine-grained control over what parts of a document are visible to different scripts, and executes untrusted scripts in isolated environments where private information is not accessible. To ease deployment, we present a method for automatically inferring what nodes in a web page contain private content. This paper describes how we modify the Chromium browser to enforce newly defined security policies, presents our automatic policy generation method, and reports on experiments inferring and enforcing privacy policies for a variety of web applications.

1 Introduction

Web applications can provide better services and more targeted information by customizing content for individual users. Those customizations, however, may leak personal information to third parties whose scripts are embedded in the web page. Current web browsers grant embedded scripts full access to all content on the page, including the ability to access any personal profile information, photos, email addresses, or other private content that is displayed in the web page. Many commonly used scripts require host pages to directly embed their scripts into the host page. Scripts that must be directly embedded include popular ad networks (including Google AdSense and Yahoo! Advertising), analytics scripts (including Google Analytics), and Facebook's recently released comments API [18]. One example of how this privilege could be abused is an advertisement embedded in Facebook pages last year that offended privacy expectations by incorporating images of the user's friends in an advertisement [17].

The easiest way to isolate untrusted scripts from the host page is to put them in an iframe. Since a script included using an iframe comes from a different origin, that script cannot access any resource in the host domain. This isolation is complete—the included script cannot interact with any other part of the page. To avoid the all or nothing model, several researchers have proposed alternatives that provide untrusted scripts with limited access to the host. As we discuss further in Section 6, though, none of these solutions satisfactorily address the security, functionality, and usability requirements necessary for a solution to be widely deployed.

Threat Model. We focus on the scenario where a content provider wants to embed content from untrusted third-party scripts such as advertising, analytics scripts, and gadgets in its output pages that contain private user information. The adversary controls one or

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 60–79, 2011.

more of the scripts embedded in the target page. To obtain private content, the embedded script may use any means provided by JavaScript to get the text or attribute of a confidential node including directly calling DOM APIs or probing values of variables in host scripts. We assume a one-way trust model since our goal is to protect user content from untrusted scripts rather than to protect embedded scripts from the host page or each other. Host scripts should be able to access the full functionality of third-party scripts, but third-party scripts should not be able to access or modify host scripts. Hence, we provide a form of one-way access from host to guest scripts. In summary, our goal is to provide third-party scripts with limited access to the DOM and no access to host scripts, while granting host scripts full access to third-party scripts and the DOM.

We do not target JavaScript frameworks such as jQuery that require rich, bi-directional interactions with the host's content. In these cases, we assume the developers fully trust the third-party libraries. We also do not consider other attack vectors such as cross-site scripting attacks or web browser vulnerabilities. Many other projects have focused on mitigating these risks, and we concentrate on the scenario where the host page developer deliberately embeds untrusted scripts.

Contributions. Our approach has three main advantages over previous approaches:

Fine-grained access control. Polices can be specified at a per-node, per-script granularity. For example, we allow the host page to set DOM node *A* to be invisible to script *X*, while DOM node *B* is read-only to script *Y* and fully accessible to script *X*. Developers can explicitly allow scripts to collaborate and execute in the same context while isolating them from other scripts on the page. This provides greater flexibility and expressiveness than previous solutions. For example, MashupOS [22] and Jayaraman et al. [10] base their policies on node locations on the page. We also provide a mechanism that gives developers one-way access to untrusted JavaScript code without exposing trusted scripts. Section 2 explains the policies enabled by our mechanisms.

Compatibility. Previous approaches place restrictions on what embedded scripts may contain, often placing limits on dynamic script execution. For example, AdJail [12] and Stamm et al. [19] do not support script node insertion; Caja [14] and AdSafe [3] do not support eval. AdJail [12] also does not fully support document.write(). We avoid JavaScript source code transformations to ensure maximum compatibility and performance. Our approach allows embedded scripts to use all of JavaScript with no restrictions, except those imposed by the actual access control policy. Section 3 explains how our implementation achieves this.

Easy deployment. One of our goals is to enable developers to painlessly incorporate our protection into legacy web applications, so our approach minimizes the effort required from the developer. All developers need to do is identify untrusted scripts (which can usually be done automatically based on their origin) and annotate nodes that contain private information by adding an attribute to that node. To further reduce deployment effort, we developed a method for automatically identifying all the nodes in a web page that may contain private information (Section 4). For this, we consider any content that varies depending on whether the page is requested with or without the user's credentials as private. We envision automatic policy learning as part of a third-party or ISP service, enabling our protections to be provided without any cooperation from sites.

We evaluate our design by implementing it as a modification to the Chromium browser, and conducting both security and functionality experiments on a range of websites. Section 5 reports on our experiments that show it is possible to automatically learn effective privacy policies for most tested sites, and to enforce our isolation and privacy mechanisms without requiring developer modifications or breaking website functionality.

2 Protecting Private Data

We provide two types of protection policies: JavaScript execution isolation and DOM access control.

2.1 Execution Isolation

One of our primary goals is to let web developers easily group third-party scripts so that some of them may collaborate with each other while still remaining separated from other third-party scripts and host page scripts. To facilitate this we add a new attribute to the script tag: worldID=*string*. This idea originates from Barth et al.'s *isolated world* concept [1] which was developed to isolate browser extensions. Each world with a unique worldID is isolated from all other worlds. The worldID attribute also serves as the principal for scripts for controlling access to DOM nodes (Section 2.2).

Figure 1 illustrates the semantics of the worldID attribute. The custom and native objects of the first script (in worldID="1") are isolated from the second script because they have different worldIDs. This means the variable a, defined in the first script, is not visible in the second script, and the second script only sees the original toString method.

```
<script worldID = "1">
  var a = 3;
  function f() {}
  Boolean.prototype.toString = f;
</script>
<script worldID = "2">
  var b = a; // error: a undefined
  f(); // error: f undefined
  new Boolean(0).toString();
      // calls original toString
</script>
<script worldID = "1">
  var b = a; // OK
  new Boolean(0).toString(); // f()
</script>
```

Fig. 1. Execution context separation

```
<div id="a" RACL="1,2" WACL="1">
  User: Alice
</div>
<script worldID = "1">
  var b=document.getElementById('a'); // OK
  b.innerHTML = 'changed'; // OK
</script>
<script worldID = "2">
  var b=document.getElementById('a'); // OK
  b.innerHTML = 'changed'; // disallowed write
</script>
<script worldID = "3">
  var b=document.getElementById('a');
      // error: a not readable
</script>
```

Fig. 2. DOM access mediation

Since the third script has worldID="1", it executes within the same context as the first script and can access all the objects the first script can.

Shared Libraries. Full isolation of embedded scripts would break the functionality of many host pages. To support embedded scripts that are used as libraries, we added two new attributes to script tags: sharedLibId and useLibId. All objects inside a script tagged with a sharedLibId attribute can be accessed by the host execution context as well as all other worlds that have the corresponding useLibId attribute. The third-party scripts, however, cannot access the privileged scripts and are still bound by the DOM access policies.

For example, *Google Analytics* users can use _gaq to track business transactions. The host script pushes transaction information into the array _gaq which is later processed by Google Analytics. Now that we have isolated the context, the _gaq variable would not normally be visible in other worlds. To support this, the sharedLibId attribute is defined to identify when an embedded script is a shared library:

```
<script src="google.com/GA.js" worldID="1" SharedLibId="GA">
```

Then, other scripts can use the useLibId attribute to access objects defined in the shared library. To prevent pollution of other script objects, objects in the shared library are prefixed with the library identifier. For example,

```
<script useLibId = "GA">
    GA._gaq.push(['_addTrans', '1234', '11.99']);
</script>
```

2.2 DOM Node Access Control

In addition to isolating objects in scripts, we provide fine-grained access control over host objects at the granularity of DOM nodes. We introduce two additional tags for all nodes in the DOM tree: RACL for specifying read access, and WACL for specifying write access. Each access control list is a comma-separated list of worldIDs. Only scripts running in the worlds listed in the RACL list are permitted to read the node, and only scripts listed in WACL are permitted to modify the node. For example, if a third-party script wants to remove a node, it must have the privileges of modifying both that node and its parent (this is consistent with the JavaScript syntax for removing a node which requires two node handles: parentNode.removeChild(thisNode)). On the other hand, to append a node to an existing node, a script only needs to have write privileges for the parent node since it already has access to the node to be inserted. The ACLs a node has do not depend on its parent or children.

As shown in Figure 2, a script can only access a particular div element if it is present in the corresponding access control list of that element. This is more flexible than previous works like Adjail [12] and MashupOS [22]. Table 1 summarizes the customizable policies for providing fine-grained mediation of host objects together with the control of sharing and isolation of custom and native objects.

Special Root Properties. In addition to specific DOM nodes, we also provide a way to hide selected APIs from certain scripts. These special host objects may provide scripts

Table 1. Summary of Policy Attributes

Context	Policy syntax	Semantics
script	worldID="s"	WorldID of the script context is s
script	sharedLibId="s"	This is script from s library
script	useLibId="s"	This script requires to use s library
DOM node	RACL="$d_1, d_2, \ldots$"	Worlds that may access this
DOM node	WACL="$d_1, d_2, \ldots$"	Worlds that may modify this

with access to private information. For example, document.cookie returns authentication tokens. Since cookie is a special property of the document it is not associated with any specific node. Other examples include document.location, document.URL and document.title as well as powerful APIs such as document.write() and document.open(). Therefore, we add a set of new attributes for the <html> tag to allow developers to specify these per-API/per-script policies. These privileges are disallowed for untrusted scripts unless explicitly permitted.

3 Implementation

Our implementation is built on Google's open source Chromium project (revision 57642 on Windows 7). Approximately 1500 lines of code were added or modified, mostly in the WebKit DOM implementation and the bindings of V8 JavaScript interpreter and WebKit DOM. We did not modify V8. Hence, our implementation could be adapted to other browsers that use WebKit as well with the effort of adding *isolated world* support.

Figure 3 illustrates how a DOM API call is executed in our system. In step 1, the WebKit parser parses a raw HTML file from a remote server and passes each script node to the *ScriptController* in WebKit/V8 bindings to set up the execution environment. If the context associated with the current worldID is already created, *ScriptController* tells V8 to enter, otherwise it creates a new one. In step 2, the *ScriptController* sends the script to V8 to start script execution. At some point, V8 encounters a DOM API call and invokes a callback to the corresponding function using the WebKit/V8 bindings (step 3).

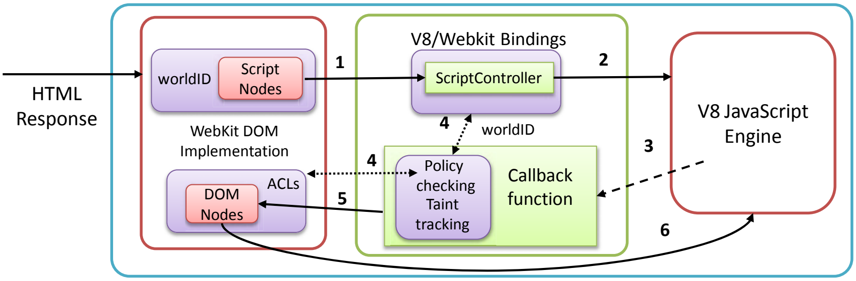


Fig. 3. Execution flow of a DOM API call

In step 4, that callback function is modified to include policy checking code that checks the worldID against the ACLs of the node. After passing the policy checking, the call is forwarded to the WebKit DOM implementation (step 5). In cases where modification happens, the target node is also tainted according to rules explained in Section 3.3. Finally, the result is returned from the WebKit DOM back to V8 (step 6). Next, we provide details on how we enforce script isolation and mediate access to the DOM. Section 3.4 discusses some special issues for handling dynamically-generated scripts.

3.1 Script Execution Isolation

Isolating any two scripts by putting them into different execution contexts allows us to specify per-script policies. We adopt Barth et al.'s *isolated world* mechanism [1]. This is used in Chrome to separate the execution context of different browser extensions, so a security compromise of one extension does not compromise the host page or other extensions. The isolated world mechanism replaces the one-to-one DOM-to-JS execution context mapping with a one-to-many map where each context maintains a mapping table to the DOM elements of the host page. This ensures that only host objects are shared among all worlds, but not native or custom objects. If a script in world 1 declares a local variable or modifies the toString prototype function, it is not visible to other worlds. If that script changes host page DOM elements, though, the changes are propagated to all other worlds (our policy mechanisms can disallow such modifications).

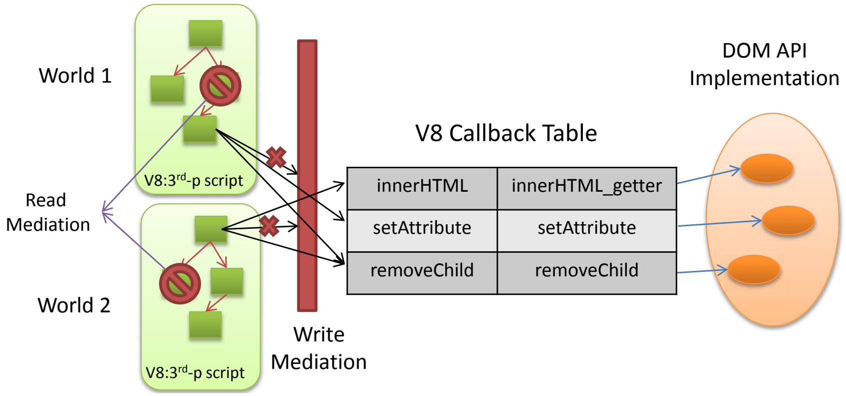
We extend this mechanism to apply to embedded scripts instead of just extension content scripts. We modified Chrome to recognize a new attribute worldID so that the WebCore ScriptController can support different JavaScript execution contexts according to a scripts' worldID. A hashmap of all the execution contexts is instantiated on a per-page basis to enable scripts to execute in the correct context. Two scripts with different worldIDs run in completely different contexts and cannot access each other's objects.

Host Script Access. For compatibility, we also need an asymmetric way for host scripts to access third-party objects. We take advantage of two properties: (1) all objects defined in the script are children of a global object, DOMWindow; and (2) it is possible to inject arbitrary objects into another context using Google V8 JavaScript engine APIs. We modified the browser to automatically grab the handle of the global object of that context and inject it into the host context as soon as a third-party script execution context with a SharedLibId is created. As long as the global objects of trusted contexts are never passed to untrusted contexts, third-party scripts are never able to access objects in trusted scripts. Here, developers need to be careful not to pass any confidential host objects to untrusted scripts.

3.2 DOM Access Control

Fine-grained policies allow different scripts to be granted different access permissions. We do this by either hiding inaccessible nodes from scripts based on their worldID, or in cases where more expressive policies are needed, by mediating access requests.

Completely Hidden Nodes. Unreadable nodes can be completely hidden from scripts. In our implementation, attempts to request a reference of a hidden node or any API on a hidden node instead receive a fabricated result. We return the v8::null() object for



innerHTML	innerHTML_getter
setAttribute	setAttribute
removeChild	removeChild

Fig. 4. JavaScript to DOM API Mediation

functions that would normally return a DOM node wrapper; we return an empty string object for functions that would normally return a string object. The null results avoid leaking any information, but should enable a well-written script to continue (we confirm this in our experiments, as reported in Section 5.2).

Our implementation mediates all DOM API getter functions to check the ACL of target node as shown in Figure 4. The upperleft and the lowerleft squares indicate two different execution worlds. As each world tries to grab handles of different nodes or call getterAPIs on those nodes, some of them are thwarted by our mediation according to respective policies; the ones that get through are executed normally.

Mediated functions include all the node handler getters as well as APIs that can be called after a node handler is held, such as getAttribute(). One of the trickier APIs to deal with is the innerHTML getter, as well as similar APIs. These APIs are designed to return the text/HTML markup of all children of this node. AdJail[12] does not have to worry about this since their policies require that the parent of a subtree cannot be assigned more privileges than the intersection of its children's privileges. Since in our case some of the children may have been marked private while the root node is marked public, calling the innerHTML getter on parent nodes may reveal confidential information in its children. To remedy this, we modify the implementation of the innerHTML callback and other similar APIs to filter out private nodes from the result.

Read-Only Access. Providing read-only or other restricted access is more complex since it requires giving the script a handle to the node. There are five ways a script may modify a node: 1) directly changing a node property (Chrome calls the internal setter function), 2) modifying the style of that node, 3) modifying the children of that node, 4) modifying the attribute of that node by calling node-specific JS-DOM APIs (e.g., setAttribute(), textContent), or 5) attaching or removing any event handlers to that node (e.g. addeventhandler()). Each of these is handled in a completely different fashion in Chromium, so it is necessary to address all of them.

We modified all related JS-DOM binding functions and made sure that if a script's worldID does not appear in the WACL of a node it cannot do any of these actions. Special caution has to be used when coping with textNode because the browser exposes a quite different set of APIs. The security attributes, WACL, RACL, and worldID should never be changed by scripts other than the host since this would allow untrusted scripts to change the policy. We therefore modified the attribute setters to check attribute names and the script's worldID to prevent unauthorized modifications to these attributes.

3.3 Taint-Tracking

Since a node may initially contain public information, but later be modified by a script to contain private information. This use of Ajax/XHR to dynamically authenticate users and update respective content is not uncommon among the sites we have tested (for example, `cnn.com` uses JavaScript to update the username box on the upper right corner of the page after the entire page is loaded). Thus, it is important to update the privacy status of a node when it is modified by a script. We do this using a conservative taint-tracking technique that marks a DOM node as private whenever any host script modifies it. Nodes that are modified by a script with worldID=*a* are only visible to scripts in world *a* as well as the host scripts.

We implemented a simple taint-tracking design that automatically marks a node as private when it is changed by a host script. Since our experiments show that this tainting policy occasionally leads to compatibility problems when too many nodes are tainted, we relaxed tainting by adding a heuristic to only taint nodes whose text content or source attributes are changed by the script. This lowers the false positive rate by ignoring the CSS and location changes of the nodes. In case this policy is too relaxed for certain websites, developers can manually mark these nodes as private using the WACL or RACL attributes. This heuristic does not pose a privacy risk, but enables side-channels between scripts that could otherwise not communicate since they may be able to modify a node that can be read by the other script. We do not consider this a serious security risk since private data is only exposed to a third-party when explicitly allowed by the policy, so although that script can now leak the data to a different third-party script it could also misuse the data directly.

3.4 Dynamic Scripting

Many previous works feared the consequences of allowing dynamically-generated code and simply excluded dynamic parts of JavaScript such as eval. This fear is justified for any rewriting-based approach since dynamically-generated code circumvents the rewriting protections. Since we enforce policies at run-time, we can fully support dynamic scripts but need to be careful to assign the appropriate policies to generated scripts. In particular, generated scripts may execute in different contexts from the scripts that created them. This may break functionality since variables and functions that should be shared are now isolated. More seriously, it may also lead to privilege escalations if less privileged scripts are able to dynamically create a higher privileged script.

We solve both problems by propagating worldIDs. Dynamically-generated scripts inherit the worldID from their creator, thus executing within the same context. We mediate all four ways to dynamically evaluate a script: 1) calling eval() or setTimeOut(), etc.; 2) defining an anchor element with JavaScript pseudo-protocol (i.e., javascript:code;); 3) creating a script node with arbitrary code; or 4) embedding a new script node by calling document.write(). The first two cases are handled by modifying respective script initialization functions in the V8ScheduledAction and ScriptController class. For the third case, we strip any worldID attributes from created node and add the creator's worldID attribute. This is done automatically inside the browser. The fourth situation is most complex, and discussed next.

Injected Scripts. In the fourth situation, scripts may be added to the page dynamically using document.write or document.writeln. These functions can dynamically create scripts by injecting raw HTML code into the page. These interfaces are very powerful, but it is necessary to support them to maintain compatibility with many existing web applications. To address this, we inject several lines of code in the HTML parser to ensure the parser correctly interprets the current execution context and then adds the appropriate worldID attribute to dynamically-created script nodes.

Event Handlers. Third-party scripts may also insert code in the context of host scripts by adding that code as an event handler of another DOM node, assuming the event can be triggered (e.g., using the onload event). There are four possible ways to attach an event handler: 1) direct assignment (e.g., someNode.onclick = 'somefunction()'), 2) setAttribute, 3) addEventListener, or 4) creating an attribute node and attaching it to a node (e.g., <div onclick = 'somefunction()'>). To preserve policy enforcement and execution context, an event handler should execute in the same context as the script that created it. For each of the four ways of attaching event handlers, we propagate the worldID to make sure that the event handler executes in the correct context. Note that after the host script registers an event handler, third-party scripts can try to call that event handler even if the event is not triggered. Hence, we associate all event handlers with their creator's worldID and mediate all the getters of event handlers to make sure the caller's worldID is identical to the callee's.

4 Automatic Policy Generation

To protect private information in host pages, we need some way to identify what nodes in the host page contain private information. This could be done by web application developers manually annotating nodes as public or private. Manual annotation, however, is probably too tedious for most web applications and unlikely to happen until a protection system is widely deployed. If we had access to the server, one strategy would be to use information flow techniques at the server to track private content and mark nodes containing private content when they are output. Since we do not assume server access, however, here we instead present a dynamic technique for inferring private content solely based on the pages returned from different requests.

We define *private content* as any content that varies depending on user credentials. Thus, any content that is different in an authenticated session from what would be retrieved for the same request in an unauthenticated session is deemed private. Nodes that directly contain private information should be marked private, but not the parent of that node. For example, if <div><span>*Username*</span></div> appears, only the inner span element is private, but not the outer div. The fine-grained nature of our policy enforcement supports this. We automate learning policies by submitting multiple requests to the server with different credentials, and identifying the differences as potentially private content.

One of our design goals is to minimize the changes have to be made both on server side and on client side, so we use a proxy to add security policies. Figure 5 illustrates the structure of our policy learner. The proxy automatically identifies third-party scripts and generates the policies for the response when a request is captured. The resulting page, including the inferred policies, is passed on to browser client.

Our proxy is implemented using Squid, which supports the Internet Content Adaptation Protocol (ICAP) that allows us to modify web traffic on the fly. For convenience, we run the Squid server in the same machine as our modified browser, however it could be moved onto an intermediate node along the routing path for better centralized control. For the ICAP server implementation, we use GreasySpoon [15]. This design cannot deal with SSL web traffic since the proxy will only see the encrypted traffic. The Chromium development group is currently (as of June 2011) still working to implement webRequest and webNavigation as experimental extension APIs [6]. Once these are implemented, we can move our proxy server inside the browser thus making it work on SSL/TLS traffic and easing deployment.

The content adaptation is divided into two main functions: third-party script identification and public node marking.

Third-party Script Identification. The ICAP server examines the response header. For each script with a source tag we compare the script's source with the host domain. For scripts that come from different domains, we add worldID attributes that identify the origin and indicate that they are not trusted by the host.

Identifying Public Nodes. To identify public content, our proxy compares the responses from two requests, one with the user's credentials and one without, and denotes any content that is identical in both responses as public. For example, assume a user visits `nytimes.com` so the browser sends a request including the user's cookies as credentials to `nytimes.com` and stores this response as R_{priv}. Once our ICAP server sees the incoming response it sends the same request except without including the cookies, storing the response as R_{pub}.

Once it has both responses, the proxy executes a differencing algorithm. This is similar to a simple text diff, except it follows the node structure. Initially, all nodes

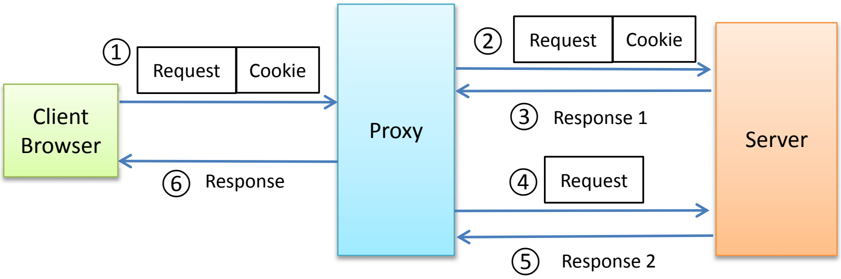


Fig. 5. Automatic Policy Generation

are assumed to be private. Then, any node in R_{priv} that appears identically in R_{pub} is marked as public. For write accesses, we make sure all children of a root node are the same before marking the root node public. Read access is slightly more relaxed than write access, since we already modified innerHTML function to conceal private nodes inside a subtree. As long as the attributes and immediate textnode children are the same in both responses we mark that node public.

State-Changing Requests. Our policy learning process requires sending duplicate requests to the server. This could have undesirable side effects if an unauthenticated request can alter server state. To limit this, POST requests are ignored since sending them twice could result in undesired state changes at the server. The entire response from a POST request is considered private. The HTML specification suggests GET methods should be idempotent [4], but many sites do make persistent state changes in response to GET requests. For example, a forum site might use a GET request for anonymous postings. If we submit the request twice the anonymous comment may be posted twice since no credentials are required for the posting.

We consider two possible solutions. The first is for the server to annotate non-idempotent pages. The first time a user visits a site the proxy has not seen before, it skips the request duplication and looks for idempotent field in the response header. Servers can send idempotent=**false** in the header to indicate that the browser should not to send duplicate requests for this page. If the idempotent field is not detected in the first response we resume the proxy behavior and submit the duplicate requests.

A second approach is to set up a third-party service like AdBlock and have users subscribe to this service. The centralized server collects information from users and correlates responses to mark private data. If we have an authority like this we do not necessarily need to send two requests since other users may have already submitted similar requests and the server should already have recorded the responses. This centralized server should be established at the ISP so that we do not introduce extra vulnerable point in the network path. Of course in this case the ISP server's identification accuracy would affect many more users than a local proxy, but it is also convenient to manually correct the mistakes as a center server. This approach also has a drawback that the requests may not necessarily happen near each other in terms of timing. For a highly active news site like `nytimes.com`, the structure or content might change fast enough that more false positives will appear.

5 Evaluation

We evaluated security or our implementation by manually testing a range of possible attacks, its compatibility with a sample of web applications, and the effectiveness of the automatic policy generator.

5.1 Security

We tested our implementation against all attack vectors we could identify from the W3C DOM [21] and ECMA specifications [9]. Table 2 lists the attack vectors and examples of the attacks we tested. For each attack vector, we created at least one test case for

Table 2. Attack Space Summary

Attack Type	Examples
Calling DOM to get nodes	document.getElementById(), nextSibling(), window.nodeID
Calling DOM to modify nodes	nodeHandler.setAttribute(), innerHTML=, nodeHandler.removeChild()
Probing host for private objects	reading host vars, calling host functions and event handlers
Accessing special properties	document.cookie, open(), document.location

each feature in the W3C DOM/ECMAscript specification and confirmed that the attack is thwarted by our implementation. Since most of these attack vectors are handled by a few functions in the Chromium implementation, this provides a reasonably high level of confidence that our implementation is not vulnerable to these attacks.

5.2 Compatibility

To evaluate how much our defense mechanisms disrupts benign functionality of typical web applications, we conducted experiments on a proof-of-concept website we built ourselves and on a broad sampling of existing websites.

The first experiment uses a constructed webpage that contains all the required annotations and third-party scripts. This page functions well in our modified browser. Both advertising networks we tested (Google Adsense and Clicksor) behave normally during testing with no errors even while hiding as much user information as possible from those scripts by marking content nodes as private. Security properties verified previously ensure that embedded third-party scripts cannot access the private content.

For real-world web applications, there is no easy way to automate testing because of the need to create and log into accounts, as well as to interact with the site. This limits the number of sites we can test. We picked 60 sites to test, sampling a range of sites based on popularity. We chose the top 20 US sites according to Alexa.com, 20 sites from sites ranked 80-300 (primarily from 80-100 with some other sites randomly picked from 100-300 to substitute for sites with inappropriate content, e.g. porn sites), and 20 sites from below the rank of 1000 (randomly selected from sites ranked from 1000-10000). For each site, we tested basic functionalities such as login and site-specific operations. These sites contained a variety of third-party scripts including advertising networks (Doubleclick, Adsonar, Ad4game, etc.) and Google Analytics. We isolated the third-party scripts and added the privacy policies on nodes that carry user data. We did not modify the embedded scripts. Policies for these pages are automatically generated by our policy learner which we evaluate more extensively in the following section. Here, we ensure the third-party script identifications are correct. In cases where a compatibility issue arises due to errors in the automatic third-party script identification, we manually correct the policies and test the functionality again. We discuss situations where the policy learner produces an incorrect policy in Section 5.3.

We relaxed our policy learner to always give the <head> tag's write access to third-party scripts. This was necessary since some analytics and ad network scripts inject script nodes in the head region. This does not compromise confidentiality due to the fine-grained nature of our policy: user-sensitive data is never revealed from children nodes of the <head> tag as long as the tags that directly containing the private information are marked private.

With the assistance of our automatic policy generator and minimum manual annotation effort (mainly helping proxy server to recognize important library scripts as host scripts such as jQuery), 46 out of 60 sites functioned without a problem. Of these, 23 sites do require manual identification of third-party scripts. For example, we added aolcdn.com to aol.com's whitelist as trusted domain.

Of the 14 sites that have problems, four are due to our HTML parser, Nokogiri [16], crashing on the site's HTML. Two sites do not contain login functions, another two sites use only SSL traffic which our current implementation of policy learner cannot tackle. Three sites show significant JavaScript console errors, all due to host script trying to access many guest objects (e.g., _gaq as mentioned before) but our policy learner cannot automatically add the global window object before these accesses. This problem also happened in some other sites, but the access is simple and we can manually add the object easily. For more complicated cases, they can be addressed by either web developer's effort or dynamic modification within JavaScript Engine. Three sites have third-party scripts that try to access a private node and therefore crashed in the process. After a closer look at all these accesses, we found that the private nodes identified by our policy learner are actually all false positives. Appendix A provides more details on the results for each site.

5.3 Policy Learning

To evaluate our automatic policy generator tested requests to sample websites and evaluated the accuracy of third-party script identification and node visibility marking.

Untrusted Script Identification. Our approach marks embedded scripts as untrusted when their origin is different from the page origin. The only false positives we observed resulted from websites that host scripts on another domain. This is fairly common with larger websites. For example, nytimes.com embeds some scripts from nyt.com which interact with host scripts closely, including accessing variables or functions from host scripts. There is no way to safely infer that scripts from other domains are trusted, so for this situation we resort to requiring developers to manually specify a list of additional trusted domains in the response headers. Scripts from a trusted domain are treated as if they are from host domain and execute in the same world as host scripts.

There are also situations where scripts in the host page appear to come from the host, but should not actually be trusted. This occurs when the host includes a third-party script using cut-and-paste. For instance, Google AdSense and Google Analytics require host pages to include an inlined code snippet. This is safer than an embedded script loaded from the remote site, since at least the host has the opportunity to see the script and knows that it is not vulnerable to a compromise of the remote server, but inlined scripts should still not have access to protected data on the page. Our policy generator has no way to tell whether an inlined script came from an untrusted source. This causes certain functions to break due to the isolated execution environment of two mutually dependent scripts. Our ad-hoc solution is to use heuristics to identify specific patterns in inlined scripts that correspond to commonly inlined scripts. For example, we look for _gat or _gaq in an inlined script and mark scripts that contain them with the same worldID as the embedded Google Analytics script. Since other scripts may now intentionally add such tokens in their scripts to impersonate *Google Analytics*, this is only a ad-hoc solution. Ideally, service providers would add an appropriate worldID tag in their inlined snippets.

Private Node Marking. To test the marking accuracy of our private node identification approach, we tested the basic functionalities such as login and site-specific operations on the sample sites used for the compatibility experiment. The traffic is redirected to go through the proxy server where modifications are made to the responses including adding ACLs and worldIDs. We recorded the total number of nodes, total number of nodes marked public before login and after login, total number of third-party scripts existing on the page, and the trusted domains needed to be manually added (e.g., scripts stored in Content Distribution Networks and libraries).

Table 3 summarizes the results of our policy generation experiments. Appendix A provides the full details for each tested site. The sample size and ranking denotes the total number of sites we selected from that range of ranking at Alexa.com. $Priv_{NoCred}$ is the percentage of nodes that are marked private based in two identical requests, neither with credentials. Since none of these responses actually contain any personal information, $Priv_{NoCred}$ gives a rough measure of the nodes that are marked as private because they vary between requests even though they do not contain sensitive information. $Priv_{Cred}$ is the percentage of nodes that are marked as private based on normal operation of the proxy. That is, based on the differences between two successive requests, one with and one without login credentials. The last two columns show the total number of third-party scripts on the host page and the number of trusted domains that need to be added to maintain functionality.

There is a reasonable drop in the fraction of nodes that are public after we login to the page, which is exactly what we are expecting. We can also see an increase in public content share after login as the ranking of sites goes lower, which indicates less important sites have less private information.

Statistically, the average number of third-party scripts on a page grows as the sites become less popular. This indicates that less popular sites are more likely to embed untrusted scripts than more popular sites. Finally, the number of trusted domains that have to be added averages less than one per site. This is lower for less popular sites, consistent with the expectation that hosting scripts on alternate domains is more common with popular sites. This result indicates that the effort required for developers to denote trusted sites is minimal.

We also inspected the nodes that were marked as private. Most of them do contain information that most people would consider private such as usernames, email

Table 3. Summary of Automatic Policy Generation Results

Size	Alexa Ranking	$Priv_{NoCred}$	$Priv_{Cred}$	3rd-p scripts	Trusted Domain
13	1-20	28.4%	47.4%	0.8	0.7
11	80-100+	4.3%	21.6%	2.6	0.5
18	1000+	2.0%	17.0%	2.2	0.5

addresses, personal recommendations and preferences. In addition, some nodes that contain session-related advertisements and tags are also marked private, due to values in those tags that vary across requests. These false positives are more frequently seen on the more popular sites, as these sites are more dynamic and complex.

6 Related Work

Much previous work has targeted the challenge of safely executing scripts from untrusted sources in a web page. The two main approaches are to either rewrite the JavaScript code or to modify the browser. An alternative to restricting the private information third-party scripts can access is to move the content-related computation inside the browser, therefore leaking no information at all. This approach is taken by Adnostic [20] and RePriv [5], but since it requires re-architecting the entire web infrastructure we do not consider it further. Here, we categorize previous works by their major mechanisms.

Isolating Execution Environments. Barth et al.'s *isolated worlds* mechanism is designed to protect browsers from extension vulnerabilities [1]. The mechanism they introduced isolates extensions from each other by dividing the JavaScript execution context into several independent ones. We adopt this mechanism to isolate webpage scripts. Since this work do not target privacy, it does not consider mediating DOM accesses.

Adjail [12] is the most similar work to ours. It puts third-party scripts into a shadow iframe with a different domain name, using the browser's same origin policy to isolate that frame and sets up a restricted communication channel between the shadow iframe and host page. This approach does not require any browser modification, but has several limitations including inflexible policies (sub-tree root can only have intersection of children's ACLs), difficulty to accommodate two or more embedded collaborating ad scripts and a complicated, and complex maneuvers needed to preserve impression and clickthrough results.

The HTML5 standard provides a way to execute JavaScript in different threads using *Web Workers* [8]. The goal of this is mainly to improve JavaScript performance by preventing misbehaving scripts from consuming too many resources. Another tag proposed by HTML5 that related to our goals is sandbox [7]. This provides some isolation, however the allowed policy is very coarse-grained.

Rewriting JavaScript. ADsafe [3] restricts the power of advertising scripts by using a static verifier to limit them to a safe subset of JavaScript that excludes most global variables and dangerous statements such as eval. Caja [14] also restricts JavaScript, but uses an automatic code transformation tool. The rewriting procedure of Caja is very complicated and cannot always preserve original script functionality. Conscript [13] uses aspect-oriented programming to enforce various policies. Its advising functions provide a broad range of policies such as forbidding inline scripts and enforcing Http-only cookies.

Extending Browsers. Jim et al. proposed a per-script policy to defend against XSS attacks [11]. The basic idea is to create a whitelist of the hash of all scripts that are allowed to run on the page. MashupOS addresses the integrator-provider security gap by introducing several new tags that can be used to restrict embedded scripts in different ways [22]. However, it cannot support the policies needed to handle current ad networks since MashupOS requires the third-party content to be in embedded in a particular way. Following this work, Crites et al. proposed a policy that abandons the same-origin policy by allowing the integrator to specify public and private data including DOM accesses [2]. Completely abandoning SOP would require significant changes to websites. Jayaraman et al. introduced OS protection ring idea to DOM access control [10]. Each node is assigned a privilege level and only scripts within appropriate rings can access that DOM element. Compared to these works, our work has the most expressive policies and easiest deployment.

7 Availability

Our implementation is available under an open source license. The code for our modified Chromium is at `https://github.com/Treeeater/Chromium_on_windows`. The proxy server implementation is at `https://github.com/Treeeater/GreasySpoon-proxy-script`.

Acknowledgments. This work was partly supported by grants from the National Science Foundation and the Air Force Office of Scientific Research under MURI award FA9550-09-1-0539. The authors thank Adam Barth for helpful advice on the isolated worlds implementation. We thank Jonathan Burket, Peter Chapman, Jack Davidson, Yan Huang, and Tianhao Tong for helpful comments on this work.

References

1. Barth, A., Felt, A.P., Saxena, P., Boodman, A.: Protecting Browsers from Extension Vulnerabilities. In: 17th Network and Distributed System Security Symposium (2010)
2. Crites, S., Hsu, F., Chen, H.: OMash: Enabling Secure Web Mashups via Object Abstractions. In: 15th ACM Conference on Computer and Communications Security (2008)
3. Crockford, D.: ADsafe: Making JavaScript Safe for Advertising (2007), `www.adsafe.org`
4. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: RFC2616: Hypertext Transfer Protocol - HTTP/1.1, `http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.1`
5. Fredrikson, M., Livshits, B.: RePriv: Re-Envisioning In-Browser Privacy. In: IEEE Symposium on Security and Privacy (2011)
6. The Chromium Development Group. The Chromium Projects: Notifications of Web Request and Navigation, `https://sites.google.com/a/chromium.org/dev/developers/design-documents/extensions/notifications-of-webequest-and-navigation`
7. Hickson, I.: HTML5 specification adding Sandbox attribute, `http://www.whatwg.org/specs/web-apps/current-work/#attr-iframe-sandbox`
8. Hickson, I.: Web Workers in HTML5 standard, `http://www.whatwg.org/specs/web-workers/current-work/`

9. ECMA International. ECMA JavaScript specification, http://www.ecma-international.org/publications/standards/Ecma-262.htm
10. Jayaraman, K., Du, W., Rajagopalan, B., Chapin, S.J.: ESCUDO: A Fine-Grained Protection Model for Web Browsers. In: 30th IEEE International Conference on Distributed Computing Systems (2010)
11. Jim, T., Swamy, N., Hicks, M.: Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. In: 16th International Conference on World Wide Web (2007)
12. Louw, M.T., Ganesh, K.T., Venkatakrishnan, V.N.: AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In: 19th USENIX Security Symposium (2010)
13. Meyerovich, L.A., Livshits, B.: ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. In: IEEE Symposium on Security and Privacy (2010)
14. Miller, M.S., Samuel, M., Laurie, B., Awad, I., Stay, M.: Caja: Safe Active Content in Sanitized Javascript (2007), google-caja.googlecode.com/files/caja-spec-2008-01-15.pdf (revised 2008)
15. Karel Mittig. GreasySpoon, Scripting Factory for Core Network Services, http://greasyspoon.sourceforge.net/
16. Patterson, A.: Nokogiri - An HTML, XML, SAX and Reader parser with the ability to search documents via XPath or CSS3 selectors and much more, http://nokogiri.org/
17. Rogers, M.: Facebook Advertisements Displayed Pictures of User's Friends and Families (2009), http://endofweb.co.uk/2009/07/facebook_ads_2/
18. Singel, R.: Singel-Minded: Facebook comments are another 'Good News, Bad News' proposition, http://www.wired.com/epicenter/2011/03/singel-facebook-empire/
19. Stamm, S., Sterne, B., Markham, G.: Reining in the Web with Content Security Policy. In: 19th International Conference on World Wide Web. ACM, New York (2010)
20. Toubiana, V., Nissenbaum, H., Narayanan, A., Barocas, S., Boneh, D.: Adnostic: Privacy Preserving Targeted Advertising. In: 17th Network and Distributed System Security Symposium (2010)
21. W3C. W3C Document Object Model Level 3 Core Specification, http://www.w3.org/TR/DOM-Level-3-Core/
22. Wang, H.J., Fan, X., Howell, J., Jackson, C.: Protection and Communication Abstractions for Web Browsers in MashupOS. In: 21st ACM SIGOPS Symposium on Operating Systems Principles (2007)

A Automatic Policy Generation Results

Here we present the results from our automatic policy generation experiments. For each site we report:

- $Priv_{NoCred}$ — the number of nodes marked private based on two requests, both without any user credentials (cookies).
- $Priv_{Cred}$ — the number of nodes marked private based on normal proxy operation (two requests, one with and one without credentials)
- 3rd-p scripts — the number of scripts from different origins included in the site.
- Compatibility — any compatibility problems due to our protections.

We group the results into three tables, corresponding to the top (1-50), middle (50-300) and lower (1000+) ranking sites respectively, according to Alexa.com. The result for some sites are excluded due to non-applicability (SSL traffic, no login approaches, e.g.).

Policy Learning Results for Top-Ranked Sites

Sample sites	$Priv_{NoCred}$		$Priv_{Cred}$		3rd-p scripts	Compatibility	Trusted Domain
Google	188/243	78%	209/265	79%	none	none	none
Facebook	25/421	6%	122/195	63%	none	none	fbcdn.net
Yahoo	3417/8273	34%	3600/8221	35%	keywordblocks, s0.2mdn.com	none	yahooapis yimg
Youtube	32/739	4%	494/1195	41%	GA[4]	inline access[2]	none
Amazon	295/1049	28%	640/1490	43%	none	none	images-amazon
Twitter	399/752	53%	100/110	91%	GA	policy violation[3]	twimg,jQuery
Craigslist	0/1045	0%	0/1051	0%	none	none	none
Linkedin	5/262	2%	794/876	91%	GA	inline access	none
MSN	189/676	28%	36/1083	4%	none	none	s-msn.com
Bing	40/187	22%	46/188	24%	none	none	none
Aol	29/703	4%	75/706	11%	player.it	none	aolcdn.com
CNN	4/1416	1%	N/A[1]	N/A	Adsense dl-rms.com questionmarket insideexpressai	none	turner.com
wordpress	26/300	9%	122/349	35%	quantserve gravatar scorecardresearch	inline access	wp.com
Flickr	23/143	16%	663/699	95%	doubleverify s0.2mdn.net	none	yimg.com yahooapis.com

[1] These use scripts to change existing nodes to display user information. No significant number of public to private node changes were detected after login.

[2] Some of these pages' host script try to access vars/functions in third-party scripts and therefore encountered errors. Some of them can be corrected rather easily while others requires web developers effort or dynamic modification in the JavaScript engine.

[3] Some third-party scripts in these pages try to access private nodes. These errors should happen as the scripts violated the policies.

[4] GA stands for Google Analytics.

Policy Learning Results for Middle-Ranked Sites

Sample sites	$Priv_{NoCred}$		$Priv_{Cred}$		3rd-p scripts	Compatibility	Trusted Domain
Twitpic	18/107	17%	43/193	23%	crowdscience scorecardresearch quantserve fmpub gstatic	inline access policy violation	googleapis.com twitter
washingtonpost	1/1722	1%	192/1975	10%	facebook	inline access	none
Digg	33/967	3%	348/1000	35%	diggstatic.com	none	facebook scrorecardresearch
Expedia	66/814	8%	68/814	8%	intentmedia	none	none
vimeo	13/413	3%	229/431	53%	GA,quantserve	none	vimeocdn
statcounter	0/457	0%	53/190	28%	doubleverify	none	none
tmz.com	9/1682	1%	N/A[1]	N/A	quantserve adsonar revsci.net gumgum nexac.com s0.2mdn.net doubleverify	inline access	none
bit.ly	3/105	3%	35/121	29%	twitter,GA	inline access[2]	none
newegg.com	8/1212	1%	10/1212	1%	GA	inline access[2]	none
indeed.com	2/128	2%	9/129	7%	jobsearch,GA scrorecardresearch	policy violation[3]	none
wikia.com	2/417	1%	17/364	4%	GA vimeo quantserve	inline access	none
yelp.com	12/794	2%	115/848	14%	GA	none	yelpcdn
articlebase.com	1/1058	1%	563/729	77%	GA	none	googleapis.com
skyrock.com	427/804	53%	473/865	55%	CDN[4]	none	skyrock.net
btjunkie.com	4/349	1%	1759/2564	68%	Adbrite,GA	none	none
duckload.com	3/158	2%	134/233	58%	GA statcounter	none	googleapis.com

[1] These sites use scripts to change existing nodes to display user information. No significant number of public to private node changes were detected after login.

[2] bit.ly combine jQuery code with Google Analytics, this will not work unless a separate jQuery library is included in the page which is marked as third-party script.

[3] Some third-party scripts in these pages try to access private nodes. These errors should happen as the scripts violated the policies.

[4] Skyrock.com puts many third-party scripts onto their CDN. This is very rare and the scripts are considered first-party because CDNs need to be trsuted.

Policy Learning Results for Lowly-Ranked Sites

Sample sites	$Priv_{NoCred}$		$Priv_{Cred}$		3rd-p scripts	Compatibility	Trusted Domain
gamefaqs	12/451	3%	25/450	6%	i.i.com.com cbsinteractive	inline access	none
timeanddate	1/333	1%	3/333	1%	exponential.com tribalfusion.com	none	none
Armorgames	4/1138	1%	12/1134	1%	GA,ad4game, quantserve	inline access	cpmstar.com
Fantasyleague	13/594	2%	33/580	4%	adtech twitter sumworld	inline access	none
9gag.com	8/529	1%	125/585	21%	Adsense GA	inline access	cloudfront.net googleapis.com
Blinklist	1/321	1%	28/246	11%	GA	none	none
modcloth.com	9/316	3%	17/314	5%	GA	policy violation[1]	none
Getcloudapp	2/51	4%	86/124	69%	GA	none	none
imtalk	10/2096	1%	559/2488	22%	Adsense, addthis statcounter, GA	policy violation	none
change.org	19/355	5%	33/367	9%	GA google map simplegeo quantserve	none	googleapis
url.com	1/262	1%	28/279	10%	GA	none	none

[1] Some of these pages have Google Analytics initialization scripts that does document.getElementsByTagName('script')[0].parent.insertBefore(). Since the first script of the page is not necessarily public, this snippet would fail. Other version of Analytics use document.write therefore does not have this issue.

Preventing Web Application Injections with Complementary Character Coding

Raymond Mui and Phyllis Frankl

Polytechnic Institute of NYU
6 Metrotech Center
Brooklyn, NY, 11201, USA
wmui01@students.poly.edu,pfrankl@poly.edu

Abstract. Web application injection attacks, such as SQL injection and cross-site scripting (XSS) are major threats to the security of the Internet. Several recent research efforts have investigated the use of dynamic tainting to mitigate these threats. This paper presents complementary character coding, a new approach to character level dynamic tainting which allows efficient and precise taint propagation across the boundaries of server components, and also between servers and clients over HTTP. In this approach, each character has two encodings, which can be used to distinguish trusted and untrusted data. Small modifications to the lexical analyzers in components, such as the application code interpreter, the database management system, and (optionally) the web browser, allow them to become complement aware components, capable of using this alternative character coding scheme to enforce security policies aimed at preventing injection attacks, while continuing to function normally in other respects. This approach overcomes some weaknesses of previous dynamic tainting approaches. Notably, it offers a precise protection against persistent cross-site scripting attacks, as taint information is maintained when data is passed to a database and later retrieved by the application program. A prototype implementation with LAMP and Firefox is described. An empirical evaluation shows that the technique is effective on a group of vulnerable benchmarks and has low overhead.

1 Introduction

Web applications have become an essential part of our every day lives. As web applications become more complex, the number of programming errors and security holes in them increases, putting users at increasing risk. Injection vulnerabilities, such as cross site scripting and SQL injection, rank as the top two of the most critical web application security flaws in the OWASP (Open Web Application Security Project) top ten list [25].

Web applications typically involve interaction of several components, each of which processes a language. For example, an application may generate SQL queries that are sent to a database management system and generate HTML code with embedded Javascript that is sent to a browser, from which the scripts are sent to a Javascript interpreter. Throughout this paper we will use the term

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 80–99, 2011.

component languages to refer to the languages of various web application technologies such as PHP, SQL, HTML, Javascript, etc. We will also use the term *components* to denote the software dealing with the parsing and execution of code written in these languages from both server side and client side, such as a PHP interpreter, a database management system, a web browser, etc.

Web application injection attacks occur when user inputs are crafted to cause execution of some component language code that is not intended by the application developer. There are different classes of injection attacks depending on which component language is targeted. For example, SQL injection targets the application's SQL statements, while cross site scripting targets the application's HTML and Javascript code. These vulnerabilities exist because web applications construct statements in these component languages by mixing untrusted user inputs and trusted developer code. Best application development practice demands the inclusion of proper input validation code to remove these vulnerabilities. However, it is hard to do this because proper input validation is context sensitive. That is, the input validation routine required is different depending on the component language for which the user input is used to construct statements. For example, the input validation required for the construction of SQL statements is different from the one required for the construction of HTML, and that is different from the one required for the construction of Javascript statements inside HTML. Because of this and the increasing complexity of web applications, manual applications of input validation are becoming impractical. Just a single mistake could lead to dire consequences.

Researchers have proposed many techniques to guard against injection vulnerabilities. Several approaches use dynamic tainting techniques [6,10,12,23,24,26,27,37]. Current implementations of dynamic tainting involve instrumenting application code or modifying the application language interpreter to keep track of which memory locations contain values that are affected by user inputs. Such values are considered "tainted", or untrusted. At runtime, locations storing user inputs are marked as tainted, the taint markings are propagated so that variables that are affected by inputs (through data flow and/or control flow) can be identified and the taint status of variables is checked at "sinks" where sensitive operations are performed.

Dynamic tainting techniques are effective at preventing many classes of injection attacks, but there are a number of drawbacks to current approaches to implementing dynamic tainting. Perhaps the most limiting of these arises when applications store and/or retrieve persistent data (e.g. using a database). Current approaches to dynamic tainting do not provide a clean way to preserve the taint status of such data. Viewing the entire database as tainted, when retrieving data, is overly conservative. But viewing it as untainted leaves applications vulnerable to persistent attacks, such as stored XSS attacks.

This paper presents a new approach to dynamic tainting, in which taint marks are seamlessly carried with the data as it crosses boundaries between components. In particular, data stored in a database carries its taint status with it, allowing it to be treated appropriately when it is subsequently processed by other application code. The approach is based on *complementary character coding*, in which each character has two encodings, one used to represent untainted data and the other used to represent tainted data. Characters can be compared with *full comparison*, in which the two representations are treated differently, or *value comparison*, in which they are treated as equivalent. With fairly small modifications, components (e.g. the application language interpreter, DBMS, and optionally client-side components) can become *complement aware components (CACs)*, which use full comparison for recognizing (most) tokens of their component language, while using value comparison in other contexts. When component language code entered by a user (an attempted injection attack) is processed by the CAC under attack, the component does not recognize the component language tokens, therefore does not execute the attack. Meanwhile, trusted component language code is treated normally. This allows secure execution of code without the need of input sanitization. Ideally, the approach will be deployed with complement aware components on both the server side and the client side, but we also demonstrate a server side only approach that still protects current web browsers against XSS attacks. This allows for a gradual migration strategy through the use of server side HTTP content negotiation, supporting both current web browsers and complement aware browsers at once.

In addition to offering protection against stored attacks, our approach has several other attractive features. Existing dynamic tainting approaches require the processing at sinks to embody detailed knowledge of the component language with which the application is interacting at the sink (e.g. SQL dialect, HTML) and to parse the strings accordingly. Our technique delegates this checking to the components, which need to parse the strings the application is passing to them anyway. This provides increased efficiency and, potentially, increased accuracy. Taint propagation is also very efficient with complementary character coding, because taint propagation via data flow occurs automatically, without the need for auxilliary data structures. The main contributions of this work are:

- The concept of *complementary character coding*, a character encoding scheme where each character is encoded with two code points instead of one. Two forms of complementary character coding, *Complementary ASCII* and *complementary Unicode*, are presented.
- A new approach to dynamic tainting with complementary character coding, which allows transparent taint propagation across component boundaries.
- The concept of *complement aware components (CAC)*, which use complementary character coding to prevent a number of web application input injection attacks, including SQL injection and cross site scripting.
- A proof of concept implementation of LAMP (Linux Apache MySQL PHP) and Firefox using the technique in complementary ASCII. The prototype LAMP implementation is also backwards compatible with current web browsers.
- An experimental evaluation of the prototype, demonstrating that the approach effectively prevents SQL injection, reflected, and stored XSS attacks without causing defects for legitimate requests, and has low overhead.

The remainder of this section presents a motivating example. Section 2 introduces complementary character coding. Section 3 describes how complementary character coding is used to implement dynamic tainting and how complement aware components prevent injection attacks, without the need for sanitization. Section 4 illustrates how the example attacks are prevented with our technique. Section 5 describes the prototype implementation, Section 6 shows the results of the experimental evaluation, Section 7 discusses related work, and Section 8 concludes.

```
<?php

//connect to database
connectdb();

//unsanitized user inputs
$message = $_POST['message'];
$username = $_POST['username'];

//html header
echo '<html>
  <head> <title>Blog</title> </head>
  <body>';

//welcome the user
if(isset($username)) {
  echo "Welcome $username <br />";
}

//insert new message
if(isset($message)) {
$query = "insert into messages values ('$username', '$message')";
  $result = mysql_query($query);
}

//display messages except those from this user or admin
$query = "select * from messages where username != '" + $username + "'";
$result = mysql_query($query);
echo '<br /><b>Your messages:</b>';
while($row=mysql_fetch_assoc($result)){
  if($row['username'] != "admin") {
    echo "<br />{$row['username']} wrote: <br />{$row['message']}<br />";
  }
}

//display the rest of html...
```

Fig. 1. Motivating Example

Figure 1 contains the code of an example web application written in PHP. The database contains a single table, called *messages* with attributes *username* and *message*, both stored as strings. Four input cases are shown in Figure 2. Case one is an example of a normal execution. Case two is a SQL injection attack. Case three is a reflected cross site scripting attack. Case four is a persistent cross site scripting attack. Please refer to appendix A for detailed descriptions of each case. In Section 4 below, we will show how our technique prevents these attacks.

```
Case 1:
username = user          message = hello

Case 2:
username = user          message = hello');drop table messages;--

Case 3:
username = <script>document.location="http://poly.edu"</script>          message = hello

Case 4:
username = user          message = <script>document.location="http://poly.edu"</script>
```

Fig. 2. Input Cases for Example in Fig. 1

2 Complementary Character Coding

In complementary character coding, each character is encoded with two code points instead of one. That is, we have two versions of every character. This section introduces *complementary ASCII* and *complementary Unicode*, two forms of complementary character coding, as well as the concepts of *value comparison* and *full comparison* which are used to compare characters in complementary character coding.

2.1 Complementary ASCII

In complementary character coding, there are two versions of each character. Standard ASCII uses 7 bits per character (with values 0–127), while each byte is 8 bits (with values 0–256). Complementary ASCII is encoded as follows: The lowest seven bits are called the *data bits*, which corresponds to standard ASCII characters 0–127. The eighth bit is called the *sign bit*, a sign bit of 0 corresponds to a *standard character* and a sign bit of 1 corresponds to a *complement character*. In other words, for every standard character c in $\{0...127\}$ from standard ASCII, there exists a complement character $c' = c + 128$ that is its complement. The conversion between standard and complement characters in complementary ASCII can be done in a single instruction by flipping the sign bit.

2.2 Value Comparison and Full Comparison

Since there are two versions of every character in complementary character coding, there must be certain rules to establish how characters are compared. In complementary character coding there are two different ways to compare characters, *value comparison* and *full comparison*. Under value comparison, a standard character is equivalent to its complement version. A simple way to implement value comparison is to compute the standard forms of the characters and compare them. Full comparison, however, compares all bits of a character. Therefore under full comparison the standard and complement versions of the same character are not equal. Note that all complement characters will be evaluated as greater than all standard characters under full comparison regardless of the value of their data bits. This is not a problem because our technique only uses full comparison for *equals* and *not equals* comparisons.

2.3 Complementary Unicode

With the internationalization of the web, Unicode [32] schemes are becoming the standard character formats for displaying web content. Currently Unicode contains over a million code points and as of the current version of Unicode 6.0 less than 25 percent of this space is used or reserved. Due to the vast amount of available space, complementary Unicode can be implemented in different ways. One possible implementation of complementary Unicode can be done just like complementary ASCII through the use of the high order bit as the sign bit. Under this representation the operations of character conversion, value comparison and full comparison are implemented in nearly the same way as their counterparts in complementary ASCII. The extra space also allows the possibility of for having more than two versions of every character, which will be explored in future work.

3 Preventing Injections with Complementary Coding

This section describes how complementary character coding is used to implement dynamic tainting and how complement aware components prevent injection attacks, without the need for sanitization. The use of HTTP content negotiation to ensure backwards compatibility with non-complement aware web browsers is also discussed. Assumptions about the application code and environment are noted in section 3.4.

3.1 Dynamic Tainting with Complementary Coding

Prior dynamic tainting techniques maintain a data structure indicating which variables or memory locations are tainted. This data structure is initialized and updated either by instrumenting the application code or by modifying the application language interpreter. Points where the application sends code to other components (e.g. SQL statements to the DBMS, HTML to the browser, etc.) are called “sinks”. Custom checks are performed at sinks to check whether tainted data is used inappropriately.

In contrast, our approach does not require an additional data structure to track taint status. Trusted application code is encoded in standard characters as not tainted. When character strings from an untrusted source enter the system, they are tainted as the web server converts them into complement characters. Value comparison is used to compare characters during execution of the application code, thus the program continues to function normally in spite of the fact that extra information (taint status) is carried along with each character. Since a character and its taint status reside in the same piece of data, taint propagation via dataflow occurs automatically.

If the component C to which a string is being sent is complement aware, checking of whether tainted data is being used appropriately is delegated to C, as discussed in section 3.2, so no code instrumentation is needed at the taint sink. If C is a legacy component that is not complement aware, taint sink processing similar to that of existing dynamic tainting techniques can be used.

Complementary character coding has the following advantages over prior dynamic tainting techniques: First it allows for free taint storage and implicit taint propagation through normal execution, removing the need for code instrumentation and the resulting overhead of existing dynamic tainting techniques. Second, under the guise of a character encoding, our technique allows for seamless taint propagation between different server-side components, and also between servers and clients over HTTP. This approach is particularly useful against persistent cross site scripting attacks, as taint status of every character is automatically stored in the database, along with the character. Data read in from the database carries detailed information about taint status. Thus, when such data becomes the web application output, it can be handled appropriately (either through a complement aware browser or through server-side filtering.)

3.2 Complement Aware Components

We now describe how a component can leverage complementary character coding to allow safe execution against injection attacks. A web application constructs statements of a component language by mixing trusted strings provided by the developers and untrusted user input data and sends these to other components.

Each component C takes inputs in a formal language $\mathcal{L}_\mathcal{C}$ with a well-defined lexical and grammatical structure (SQL, HTML, etc.). As in reference [30] each component language can have a security policy that stipulates where untrusted user inputs are permitted within elements of $\mathcal{L}_\mathcal{C}$. In general, a security policy could be expressed at the level of $\mathcal{L}_\mathcal{C}$'s context free grammar, but our technique focuses on security policies defined at the level of $\mathcal{L}_\mathcal{C}$'s lexical structure.

In our approach, complementary character coding is used to distinguish trusted (developer-generated) characters from untrusted (user-generated) characters throughout the system. Trusted characters are represented by standard characters while untrusted characters are converted to complement characters by the web server. By making small modifications to their parsers, components can be made *complement aware*, capable of safe execution against input injection attacks without need of sanitization through the enforcement of a default security policy, or other optional policies if the default policy is deemed too restrictive.

More formally, the security policy of a complement aware component C is defined in terms of the tokens of $\mathcal{L}_\mathcal{C}$. The *allowed tokens* are tokens which can include untrusted characters; all other tokens are designated as *sensitive tokens* where untrusted characters are not allowed.

We define a *Default Policy* for each component language as follows: *All tokens except literal strings and numbers are sensitive.* The Default Policy defines the allowed token set as numbers and literal strings, all other tokens are defined as sensitive tokens. For example, the Default Policy applied to SQL states that tokens representing numbers and literal strings are allowed tokens, while all other tokens representing SQL keywords, operators, attribute names, delimiters, etc. are sensitive tokens.

A component C with input language $\mathcal{L}_\mathcal{C}$ is *complement aware* with respect to a security policy P with allowed token set A_P if

- The character set includes all relevant standard and complement characters (e.g. complementary ASCII or complementary Unicode).
- Sensitive tokens, i.e., tokens that are not in A_P, only contain standard characters.
- $\mathcal{L}_\mathcal{C}$ has a default token d which is in A_P. Strings that do not match any other token match d. (Typically this would be the string literal token).
- During lexical analysis C uses value comparison while attempting to recognize tokens in A_P and uses full comparison for all other tokens.
- Aside from parsing, C uses value comparison (e.g. during execution).

The first four elements assure that complement aware components enforce their security policies and the last element allow the component to function normally after checking the security policy, so data values are compared as usual, preserving normal functionality.

Assume trusted developer code is encoded in standard characters and user inputs are translated into complement characters on entry to the system (e.g. by the web server). When the application sends a string s to component C, no substring of s that contains complement characters can match any sensitive token under full comparison. Therefore the following **Safety Property** is satisfied: If component C is complement aware with respect to security policy P then C enforces P, i.e., for any string s, consisting of trusted (standard) and untrusted (complement) characters that is input to C, parsing s with $\mathcal{L}_\mathcal{C}$'s grammar yields a parse tree in which every token (terminal symbol) that contains untrusted characters is in A_P. Consequently, when the parsed token stream is further interpreted (e.g. during execution of the input), no sensitive tokens will come from untrusted inputs.

Note that if C is complement aware with respect to the Default Policy and if s is an attempted injection attack in which characters that come from the user are encoded with complement characters, then C's lexical analyzer will treat any keywords, operators, delimiters, etc. in s that contain complement characters (i.e. that were entered by the user) as parts of the default token (string literal), and the attack string will be safely executed like normal inputs.

The Default Policy is a strong policy that is restrictive. It is designed to be a safe default that is applicable to a wide number of languages against both malicious and non-malicious types of injections. For example, the Default Policy would define the use of HTML boldface tags ($<b>$ and $</b>$) from user inputs as a form of HTML injection. Other less restrictive policies can be defined through the addition of more tokens to the allowed token set A_P. For example, if the developers of a web browser wish to allow users to enter boldface tags, they can modify the Default Policy by adding boldface tags to A_P, creating a less restrictive policy which allows the browser to interpret untrusted boldface tags.

3.3 Architecture, Backwards Compatibility and Migration Strategy

Figure 3 provides an architectural overview of our technique. User inputs are converted into complement characters by the server upon entry. We can ensure backwards compatibility between the complement aware server and legacy web

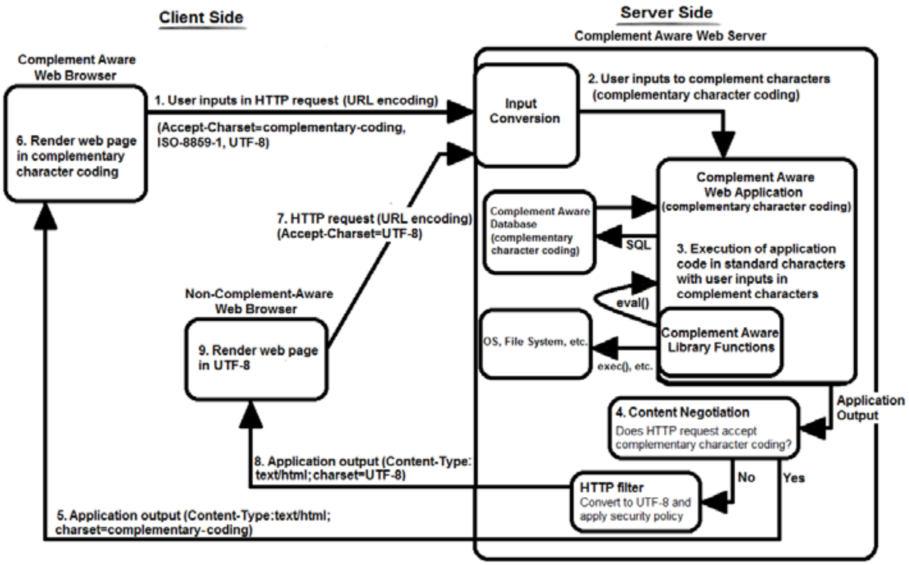

Client Side
Server Side
Complement Aware Web Server
Complement Aware Web Browser
6. Render web page in complementary character coding
1. User inputs in HTTP request (URL encoding)
(Accept-Charset=complementary-coding, ISO-8859-1, UTF-8)
Input Conversion
2. User inputs to complement characters (complementary character coding)
Complement Aware Web Application (complementary character coding)
3. Execution of application code in standard characters with user inputs in complement characters
Complement Aware Database (complementary character coding)
SQL
eval()
OS, File System, etc.
exec(), etc.
Complement Aware Library Functions
7. HTTP request (URL encoding) (Accept-Charset=UTF-8)
Non-Complement-Aware Web Browser
9. Render web page in UTF-8
Application Output
4. Content Negotiation
Does HTTP request accept complementary character coding?
No
Yes
HTTP filter
Convert to UTF-8 and apply security policy
8. Application output (Content-Type: text/html; charset=UTF-8)
5. Application output (Content-Type:text/html; charset=complementary-coding)

Fig. 3. Architecture of Our Technique

browsers with the use of HTTP content negotiation [36] with the Accept-Charset header. A content negotiation module, shown in step 4 of Figure 3, routes the application output in two ways. For a complement aware browser which specifies itself as complement aware in the Accept-Charset header, the content negotiation module sends the application output in complementary character coding over HTTP unchanged. For a web browser that does not support complementary character coding, the negotiation module routes the output to an HTTP filter. The filter performs the function of a complement aware web browser on the server side at the expense of server side overhead. It does so by applying the Default Policy for HTML and converting its character encoding to one that is readable by the client web browser, specified by the Accept-Charset header in the request. This modified output is then sent back to the client web browser.[1]

This architecture allows for a gradual migration strategy. Initially, deployment of complement aware servers would result in the usage of the HTTP filter for nearly all requests, resulting in extra server overhead. This extra server overhead would gradually decrease, as more and more users upgrade to complement aware web browsers, which do not require the filtering. Please refer to Appendix B for a step by step walk-through of the architecture.

3.4 Limitations

Complement aware components are mostly compatible with existing application code, with some exceptions. Since our technique is designed to execute code

[1] Server administrators who do not want clients to see which parts of the HTML come from user inputs might opt to apply server side filtering as well.

without the use of sanitization functions, existing sanitization in application code would have to be removed to avoid possible conflicts. While library sanitization functions (such as *mysql_real_escape_string* in PHP) can be made to do nothing by default, custom sanitization code would have to be changed manually. Also, application code that involves bit level operations on characters (e.g. shifting left) would not work with the technique. However in the context of web applications, we expect that developer code involving direct bit manipulations are rare. We did not encounter these during our experiments. The technique is also circumvented by applications that produce statements in component languages that are control-dependent, but not data dependent on inputs. The same problem occurs with other dynamic tainting techniques unless taint propagation via control dependence is implemented [7]. We also assume that the technique is not being used in an environment that is already compromised.

4 Example Revisited with CAC

We demonstrate how the four example cases from Section 1.1 will execute as complement aware components enforcing the Default Policy with complementary ASCII. Assume we are using a complement aware web browser. First, according to steps 1 and 2 on Figure 3, all user inputs are converted into complement characters by the server upon arrival. Developer code is encoded in standard characters. We describe each case as we begin step 3 on Figure 3, as the application begins to execute. In the example cases, we will show all complement characters with underlines.

In case one, first the application generates *Welcome user* as HTML at lines 16 to 18. At line 24, the application constructs the SQL query *insert into messages values ('user', 'hello')* and sends it to the DBMS to be executed. During parsing of the SQL query, the complement aware DBMS enforces the Default Policy by using full comparison to match all sensitive tokens in SQL. The tokens *user* and *hello* are recognized as literal strings (albeit with a non-standard character set). The values *user* and *hello* are stored in the database. When lines 27 to 34 are executed, the application generates HTML to display the contents of the database. A SQL query *select * from messages where username != 'user'* is generated at line 27 and the query is passed to the DBMS at line 28. Aside from the literal *user*, the SQL tokens in this query are encoded entirely in standard characters; each string representing a token matches the intended token using full comparison, so the query is executed. During the execution of the SQL query, value comparison is used to evaluate the WHERE clause. Since under value comparison, *user*, *user*, *user*, etc. are equal to *user*, rows containing any of those entries are not selected. Other rows with usernames not equal to *user* under value comparison are selected as desired. Similarly, when the PHP interpreter performs the comparison at line 31, it uses value comparison, which works correctly – the values *admin*, *admin*, *admin*, *admin*, etc. are all equivalent to each other under value comparison so messages posted by any of these variants are excluded from the output. The generated HTML is then sent to the complement aware web browser, which parses the HTML (Steps 4, 5 and 6 on Figure 3). To enforce the Default Policy, full comparison is used during parsing to match HTML tags. Since *user* and *hello* are in complement characters while HTML tags are in standard characters, they cannot be matched as any tag under full comparison during parsing and the Default Policy is enforced. After parsing, the characters are then rendered by the web browser, at this point value comparison is used in principle, so complement characters are made to look the same as their standard counterparts on the user's screen.

The executions of cases two, three, four or any other input proceed similarly as case one. We will briefly discuss how the attacks from each case are prevented.

In case two, the SQL query *insert into messages values ('user', 'hello');drop table messages;−−')* is constructed and sent to the database parser at line 24. Full comparison is used during parsing. No substring of *hello');drop table messages;−−* matches any sensitive tokens in SQL because under full comparison, ' is not equal to ′,) is not equal to), etc. Therefore the entire input string is recognized as the default token (string literal) and is stored in the database just like any other string the user provides. The injection attack fails.

In case three, value *Welcome <script>document.location="http://poly.edu" </script>* is generated as HTML at lines 16-18. When the page is parsed by the complement aware web browser, the HTML parser uses full comparison. No tags are matched by the parser because *<script>* is not equal to <*script*> under full comparison. So the browser does not interpret the injected tag as the beginning of a script. Instead, this string is rendered literally on the screen and the reflected XSS attack fails.

Case four is the same as case three except that the attack string is stored in the database as well. Like before, the input does not match any tokens in SQL or any HTML tags under full comparison during parsing. The string is stored literally in the database and is displayed literally in the client's web browser.

This example only shows the prevention of SQL injection and cross-site scripting, however it's important to note that our technique is designed to be general and it can be used against other types of web application injections as well. With complementary character coding, wherever user input is being used to construct statements in a language that is interpreted by other components (XML interpreters, eval, etc), security policies for those components can be defined and complement aware versions of the component can be implemented to prevent injection attacks.

5 Implementation

We describe our implementation of a complement aware web server with LAMP (Linux Apache MySQL PHP) and a complement aware web browser with Firefox. The implementation is done in complementary ASCII. Our implementation of LAMP enforces the Default Policy, while our implementation of Firefox is able to enforce customized security policies through specified allowed token sets. The complement aware LAMP server incorporates the use of HTTP content negotiation to be backwards compatible with unmodified browsers as well, as discussed in section 3.3. The key effort is implementing value comparison at the right places, since full comparison is already done by default.

We begin with an installation of LAMP with an 8 bit character encoding. For simplicity, we used the *Latin-1* character set. Latin-1's first 128 characters are exactly the same as the standard characters in complementary ASCII. We use the other 128 characters to represent complement characters and modify the way they are displayed. We modified PHP to encode the contents of GET and POST input arrays into complement characters at the point they are initialized. We modified the PHP interpreter so that the bytecode instructions for comparison used value comparison. The parser continues to use full comparison. PHP string functions are modified to support complementary ASCII. For MySQL, the query execution engine was modified to use value comparison, while the parser continued to use full comparison. The content negotiation module and HTTP filter are implemented with an Apache output filter. Since we are using the Default Policy, the filter simply converts all complement characters to a safe representation by encoding them using HTML numeric character references.

Our implementation of the server is sufficient for performing our experiments. A complete implementation would for example, convert everything that comes from the user (such as *$_SERVER['PATH_INFO']*) into complement characters.

We have modified Firefox 3.5 to be complement aware and enforce a customized security policy specified from an allowed token set, as described in section 3.2. If a tag name contains only standard characters, it is interpreted normally. If a tag name contains complement characters and it exists in the allowed token set, then the corresponding tag is interpreted normally. If the tag name contains complement characters and it does not exist in the allowed token set, then the tag is not interpreted and is rendered literally.

6 Evaluation

Our experimental evaluation has two objectives: 1) to confirm that the technique is effective against attacks without causing defects, and 2) to measure the runtime overhead resulting from using our implementation. Two sets of test data were used. The SQL Injection Application Testbed [29] has been used for evaluating various techniques developed by other researchers [2,11,12,28,30]. It consists of a large number of test cases on a series of open source applications. It contains two types of test cases: the ATTACK set which contains SQL injection attacks, and the LEGIT set which contains legitimate inputs. The second benchmark is from ARDILLA [17], a technique which generates cases of SQL injection and XSS attacks automatically. This test set contains cases of SQL injections, and both reflected and persistent cross site scripting attacks on a set of open source applications. The programs are LAMP applications. Links to the benchmark programs can be found in [17,29]. The experiments were performed on a dual core 2 GHz laptop with 3 GB of RAM running our LAMP implementation based on Ubuntu 9.04, Apache 2.2.13, MySQL 5.1.39, and PHP 5.2.11.

Table 1. Summary of SQL Injection Application Testbed and ARDILLA test set

	LOC	SQL Injection	Reflected XSS	Persistent XSS	Legit
SQLIA Testbed					
bookstore	16,959	5474	–	–	608
classifieds	10,949	5590	–	–	576
empldir	5658	6388	–	–	660
events	7,242	5606	–	–	900
portal	16,453	5686	–	–	1080
ARDILLA					
schoolmate	8,181	6	10	2	–
webchess	4,722	12	13	0	–
faqforge	1,712	1	4	0	–
geccbblite	326	2	0	4	–

Table 1 summarizes the benchmarks. The first column contains the names of the applications. The second column contains the number of lines of code (LOC) from each application. Columns 3–5 show the numbers of test cases targeting each type of attack (SQL injection, reflected cross-site scripting, persistent cross-site scripting.) Column 6 shows the number of test cases representing legitimate inputs (not attempted attacks.)

For the much larger SQL Injection Application Testbed, we manually identified the queries that the developer intended to execute for each web page. For example on a Login page the only query is of the form *SELECT * FROM members WHERE username = '?'AND password = '?'*. (Here question marks denote values dependent on user inputs.) A successful SQL injection attack would result in execution of a query with a different structure. We used scripts to execute the ATTACK test cases targeting each page using the CAC prototype, and to check that queries in the database logs matched the intended forms. All the queries were of the intended forms, which shows that the prototype prevented all of the attempted SQL injection attacks from the test suite.

We executed the LEGIT test cases on both the CAC prototype and a standard configuration with identical initial environments, and compared the generated HTML using value comparison. The results of the two implementations were identical by value comparison, except for current time-stamps generated by one page; this shows that the prototype handled these non-attack inputs normally without functionality defects.

We ran the ARDILLA test cases manually. For each test case we checked the database states, database logs and the output HTML for any signs of SQL injection or XSS. We again found no signs of injection attacks. We also found no functionality defects caused by our implementation.

We then measured the response times to determine server side overhead. We expected the overhead of the technique to be small, since the only sources of overhead are from the encoding of user inputs into complement characters and the use of value comparison, each of which was implemented in a few instructions. Our evaluation is done by comparing the difference in runtime between the original LAMP installation that our implementation is based on, and our CAC implementation both with and without the use of the HTTP filter to measure

Table 2. Result of Timing Evaluation

	Default LAMP (seconds)	CAC without filter (seconds)	Percent Overhead (without filter)	CAC with filter (seconds)	Percent Overhead (with filter)
bookstore	6.816 ± 0.055	6.867 ± 0.058	0.74%	6.935 ± 0.061	1.74%
classifieds	6.852 ± 0.057	6.873 ± 0.095	0.32%	6.915 ± 0.069	0.93%
empldir	10.166 ± 0.075	10.149 ± 0.066	-0.17%	10.183 ± 0.081	0.17%
events	17.745 ± 0.186	17.723 ± 0.181	-0.12%	17.760 ± 0.183	0.09%
portal	45.581 ± 0.202	45.905 ± 0.196	0.71%	45.794 ± 0.228	0.47%

the overhead of our content negotiation technique. We only use the LEGIT set from the SQL Injection Application Testbed for this, since successful attacks from the ATTACK set on the original installation would cause different paths of execution, and produce irrelevant timing results.

We ran this test set on each setup 100 times and computed the average run time and the 95% confidence interval. The results are shown in Table 2. The first column contains the names of the applications. The second column contains the average time of the original LAMP installation over 100 runs along with its 95% confidence interval. The third column contains the average time of our complement aware server implementation without passing through the HTTP filter (interacting with a complement aware web browser). The fourth column contains the percentage difference between columns two and three. The fifth column contains the average time of our complement aware server through the HTTP filter (interacting with a legacy web browser) to show the overhead of our backwards compatibility technique.

These results show a performance improvement of complementary character coding compared to existing dynamic tainting techniques. For example, the average overhead of WASP [12] over the same benchmark is listed as 6%, while the worst case overhead of our technique is no more than 2%. Since overhead were on the order of milliseconds per request, other factors such as database operations, network delay, etc. will easily dominate it when our technique is deployed for real world applications.

7 Related Work

Researchers have proposed many other techniques against web injection attacks. Dynamic tainting techniques [6,10,12,23,24,26,27,37] have the most similarity to our technique. Dynamic tainting techniques are runtime analyses that involve the marking of every string within a program with taint variables and propagating them across execution. Attacks are detected when a tainted string is used as a sensitive value. As discussed in section 3, the difference between our technique and traditional dynamic tainting techniques is that complementary character coding provides character level taint propagation across component boundaries of web applications without the need of code instrumentation and the overhead of maintaining extra data structures to propagate taint information. Another difference is that while previous dynamic tainting techniques implement taint sinks using code instrumentation to detect attacks, our technique delegates enforcement of the security policy to the parser of each component.

Sekar proposed a technique of black-box taint inference to address some of the limitations with dynamic tainting [28], where the input/output relations of components are observed and maintained to prevent attacks. Su and Wassermann provided a formal definition of input injection attacks and developed a technique to prevent them involving comparing parse trees with an augmented grammar [30]. Bandhakavi, Bisht, Madhusudan, Venkatakrishnan developed CANDID [2], a dynamic approach to detect SQL injection attacks where candidate clones of a SQL query, one with user inputs and one with benign values, are compared during parsing. Louw and Venkatakrishnan proposed a technique to prevent cross site scripting [20] where the application sends two copies of output HTML to a web browser for comparison, one with user inputs and one with benign values. Bisht and Venkatakrishnan proposed a technique called XSS-GUARD [3], in which shadow pages and their parse trees are being compared at the server. Buehrer, Weide, and Sivilotti developed a technique involved with comparing parse trees [5] to prevent SQL injection attacks.

Static techniques [1,11,14,16,19,31,34,35] employ the use of various static code analysis techniques to locate sources of injection vulnerabilities in code. The results are either reported as output or instrumented with monitors for runtime protection, others employ the use of machine learning [13,33]. Martin, Livshits, and Lam developed PQL [21], a program query language that developers can use to find answers about injection flaws in their applications and suggested that static and dynamic techniques can be developed to solve these queries.

Boyd and Keromytis developed a technique called SQLrand [4] to prevent SQL injection attacks based on instruction set randomization. SQL keywords are randomized at the database level so attacks from user input become syntactically incorrect SQL statements. A proxy is set up between the web server and the database to perform randomization of these keywords using a key. Van Gundy and Chen proposed a technique based on instruction set randomization called Noncespaces against cross site scripting [9]. Nadji, Saxena and Song developed a technique against cross site scripting called Document Structure Integrity [22] by incorporating dynamic tainting at the application and instruction set randomization at the web browser. Kirda, Kruegel, Vigna and Jovanovic developed Noxes [18], a client side firewall based approach to detect possibilities of a cross site scripting attack using special rules. Jim, Swamy, and Hicks proposed a cross site scripting prevention technique called browser-enforced embedded policies [15] where a web browser receives instructions from the server over what scripts it should or should not run.

Interpolique [8] is a framework for sanitizing sensitive inputs by replacing their values with Base64 encodings, along with calls to components' Base64 decoding functions. Like our technique, Interpolique converts sensitive inputs into forms that do not match any tokens of component languages. However, unlike our technique, Interpolique requires developers to identify variable uses that should be transformed.

8 Conclusion and Future Work

In this paper, we have presented complementary character coding and complement aware components, a new approach to dynamic tainting for preventing a wide variety of web application injection attacks. In our approach, two encodings are used for each character, standard characters and complement characters. Untrusted data coming from users is encoded with complement characters, while trusted developer code is encoded with standard characters. Complementary character coding allows taint information about each character to be propagated across component boundaries seamlessly. Components are modified to enforce security policies, which are characterized by sets of allowed tokens, for which user input characters should not be permitted. Each complement aware component enforces its policy by using full comparison to match sensitive tokens during parsing. Elsewhere they use value comparison to preserve functionality. This allows them to safely execute attempted injection attacks as normal inputs. While ideally, the technique would be used with complement aware components on both the server side and the client side, it is backward compatible with existing browsers through HTTP content negotiation and server-side filtering. Whether deployed with complement aware browser or with a legacy browser, it provides protection against stored XSS attacks.

We have implemented a prototype for LAMP and Firefox. An experimental evaluation on the prototype prevented all SQL injection, reflected and stored cross-site scripting injection attacks and executed legitimate inputs normally in the benchmarks studied with only small overhead. Directions of future work include extending the prototype to use complementary Unicode, incorporating techniques to deal with taint propagation via control flow, and exploring other applications of complementary character coding and its extended version through the use of multiple taint bits.

Acknowledgments. This research was partially supported by the US Department of Education GAANN grant P200A090157, National Science Foundation grant CCF 0541087, and the Center for Advanced Technology in Telecommunications sponsored by NYSTAR. We thank the reviewers for their helpful comments.

References

1. Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Saner: Composing static and dynamic analysis to validate sanitization in web applications. In: SP 2008: Proceedings of the 2008 IEEE Symposium on Security and Privacy, pp. 387–401. IEEE Computer Society, Washington, DC, USA (2008)
2. Bandhakavi, S., Bisht, P., Madhusudan, P., Venkatakrishnan, V.N.: Candid: preventing SQL injection attacks using dynamic candidate evaluations. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 12–24. ACM, New York (2007)

3. Bisht, P., Venkatakrishnan, V.N.: XSS-GUARD: Precise dynamic prevention of cross-site scripting attacks. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 23–43. Springer, Heidelberg (2008)
4. Boyd, S.W., Keromytis, A.D.: SQLrand: Preventing SQL injection attacks. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 292–302. Springer, Heidelberg (2004)
5. Buehrer, G., Weide, B.W., Sivilotti, P.A.G.: Using parse tree validation to prevent SQL injection attacks. In: SEM 2005: Proceedings of the 5th International Workshop on Software Engineering and Middleware, pp. 106–113. ACM, New York (2005)
6. Chin, E., Wagner, D.: Efficient character-level taint tracking for Java. In: Proceedings of the 2009 ACM Workshop on Secure Web Services, SWS 2009, pp. 3–12. ACM, New York (2009)
7. Clause, J., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: ISSTA 2007: Proceedings of the 2007 International Symposium on Software Testing and Analysis, pp. 196–206. ACM, New York (2007)
8. Kaminsky, D.: Interpolique, http://dankaminsky.com/interpolique/
9. Gundy, M.V., Chen, H.: Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. In: NDSS (2009)
10. Haldar, V., Chandra, D., Franz, M.: Dynamic taint propagation for Java. In: ACSAC 2005: Proceedings of the 21st Annual Computer Security Applications Conference, pp. 303–311. IEEE Computer Society, Washington, DC, USA (2005)
11. Halfond, W.G.J., Orso, A.: Amnesia: analysis and monitoring for neutralizing SQL-injection attacks. In: ASE 2005: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, pp. 174–183. ACM, New York (2005)
12. Halfond, W.G.J., Orso, A., Manolios, P.: Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In: SIGSOFT 2006/FSE-14: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 175–185. ACM, New York (2006)
13. Huang, Y.W., Huang, S.K., Lin, T.P., Tsai, C.H.: Web application security assessment by fault injection and behavior monitoring. In: WWW 2003: Proceedings of the 12th International Conference on World Wide Web, pp. 148–159. ACM, New York (2003)
14. Huang, Y.W., Yu, F., Hang, C., Tsai, C.H., Lee, D.T., Kuo, S.Y.: Securing web application code by static analysis and runtime protection. In: WWW 2004: Proceedings of the 13th International Conference on World Wide Web, pp. 40–52. ACM, New York (2004)
15. Jim, T., Swamy, N., Hicks, M.: Defeating script injection attacks with browser-enforced embedded policies. In: WWW 2007: Proceedings of the 16th International Conference on World Wide Web, pp. 601–610. ACM, New York (2007)
16. Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In: SP 2006: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 258–263. IEEE Computer Society, Washington, DC, USA (2006)
17. Kieyzun, A., Guo, P.J., Jayaraman, K., Ernst, M.D.: Automatic creation of SQL injection and cross-site scripting attacks. In: ICSE 2009: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, pp. 199–209. IEEE Computer Society, Washington, DC, USA (2009)

18. Kirda, E., Kruegel, C., Vigna, G., Jovanovic, N.: Noxes: a client-side solution for mitigating cross-site scripting attacks. In: SAC 2006: Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 330–337. ACM, New York (2006)
19. Livshits, V.B., Lam, M.S.: Finding security vulnerabilities in Java applications with static analysis. In: SSYM 2005: Proceedings of the 14th Conference on USENIX Security Symposium, pp. 18–18. USENIX Association, Berkeley (2005)
20. Louw, M.T., Venkatakrishnan, V.N.: Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In: SP 2009: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, pp. 331–346. IEEE Computer Society, Washington, DC, USA (2009)
21. Martin, M., Livshits, B., Lam, M.S.: Finding application errors and security flaws using PQL: a program query language. SIGPLAN Not. 40(10), 365–383 (2005)
22. Nadji, Y., Saxena, P., Song, D.: Document structure integrity: A robust basis for cross-site scripting defense. In: NDSS (2009)
23. Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Cross-site scripting prevention with dynamic data tainting and static analysis. In: Proceeding of the Network and Distributed System Security Symposium, NDSS 2007 (2007)
24. Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically hardening web applications using precise tainting. In: Sasaki, R., Qing, S., Okamoto, E., Yoshiura, H. (eds.) SEC, pp. 295–308. Springer, Heidelberg (2005)
25. OWASP Top Ten Project, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
26. Perl security: Taint mode, http://perldoc.perl.org/perlsec.html#Taint-mode
27. Pietraszek, T., Berghe, C.V., Chris, V., Berghe, E.: Defending against injection attacks through context-sensitive string evaluation. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 124–145. Springer, Heidelberg (2006)
28. Sekar, R.: An efficient black-box technique for defeating web application attacks. In: NDSS (2009)
29. SQL Injection Application Testbed, http://www.cc.gatech.edu/~whalfond/testbed.html
30. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: POPL 2006: Conference record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 372–382. ACM, New York (2006)
31. Tripp, O., Pistoia, M., Fink, S.J., Sridharan, M., Weisman, O.: Taj: effective taint analysis of web applications. In: PLDI 2009: Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 87–97. ACM, New York (2009)
32. Unicode Consortium, http://unicode.org/
33. Valeur, F., Mutz, D., Vigna, G.: A learning-based approach to the detection of SQL attacks. In: Julisch, K., Krügel, C. (eds.) DIMVA 2005. LNCS, vol. 3548, pp. 123–140. Springer, Heidelberg (2005)
34. Wassermann, G., Su, Z.: Sound and precise analysis of web applications for injection vulnerabilities. In: PLDI 2007: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 32–41. ACM, New York (2007)
35. Wassermann, G., Su, Z.: Static detection of cross-site scripting vulnerabilities. In: ICSE 2008: Proceedings of the 30th International Conference on Software Engineering, pp. 171–180. ACM, New York (2008)

36. World Wide Web Consortium: RFC 2616 Section 12: Content Negotiation, http://www.w3.org/Protocols/rfc2616/rfc2616-sec12.html
37. Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks. In: USENIX-SS 2006: Proceedings of the 15th Conference on USENIX Security Symposium. USENIX Association, Berkeley (2006)

Appendix A: Detailed Description of Figures 1 and 2. Case one is an example of a normal execution. Lines 7 and 8 get the user's inputs from the HTTP request for this page. Lines 10 to 13 begin generation of an HTML page that will eventually be sent to the user's browser. A greeting is generated as HTML at lines 16 to 18. At lines 21 to 24, an SQL insert statement is generated then sent to MySQL, which inserts data provided by the user into the database. Lines 27 to 34 generate a SQL query, send it to MySQL, then iterate through the result set, generating HTML to display the contents of the database (excluding messages from the admin and the user). The web server sends the generated HTML to the user's browser, which parses it and displays the welcome message and the table on the user's screen. We will assume the database is not compromised initially, so no attacks occurred.

Case two is an example of a SQL injection attack. The SQL code being executed at line 23 becomes *insert into messages values ('user', 'hello');drop table messages;−−')*, since there is no input validation. This results in the deletion of the table *messages* from the database. By modifying the attack string an attacker can construct and execute other malicious SQL code as well.

Case three is an example of a reflected cross site scripting attack. The unsanitized user input (a script) is included in the HTML at line 17. When the HTML is parsed by the browser, it will recognize the script tags and send the enclosed script to its Javascript engine, which will parse it and execute it. In this case the script redirects the user to another website. An attacker can exploit this by inducing users to provide inputs like case three, causing redirection to another malicious web page which steals personal information.

Case four is an example of a persistent cross site scripting attack. At line 23, the attack script is stored in the database. It is sent to any user visiting the application when lines 27 to 34 are executed. This is a more severe form of cross site scripting because it affects everyone visiting the web page.

Appendix B: Illustration of Architecture from Figure 3. We present two scenarios to the architecture from Figure 3 in detail. Scenario (1) uses a complement aware web browser. Scenario (2) uses a non-complement aware web browser to demonstrate our content negotiation mechanism for backwards compatibility. For both scenarios, we assume the complement aware components implement the Default Policy as their security policies.

Scenario 1: In step 1, a HTTP request along with standard URL encoded user inputs are sent to the server by a complement aware web browser. The request is URL encoded as specified by the HTTP protocol, identifying itself as complement aware with the Accept-Charset header. In step 2, the server converts the user input into complementary ASCII/Unicode as complement characters. The input conversion module returns complement characters for all possible inputs. In step 3, web application executes with user inputs in complement characters, while developer code is in standard characters. Value comparison is used within the application, so it functions normally. The application constructs the HTML output by mixing developer code, user inputs, and values obtained from the database. In step 4, this output is sent to the content negotiation module, which checks the Accept-Charset header of the HTTP request to see if the client browser is complement aware. Since the browser is complement aware in scenario (1), the application output is sent back to the client browser as the HTTP response, labeling the output character set as complementary ASCII/Unicode. In step 5, the complement aware browser receives the HTML output, recognizes the output character set as complementary ASCII/Unicode and parses the output accordingly. During parsing the browser's security policy is enforced. Because the Default Policy is used, full comparison is used to match all HTML tags, comments, etc. Consequently, any such tokens that are tainted, whether they came directly from this user's input or whether they'd been stored previously then retrieved from the database, are treated as default tokens, i.e. string literals. After parsing, the page is then rendered on the screen where value comparison is used in principle; this means that complement characters are made to look like their default counterparts on the screen.

Scenario 2: The browser does not support complementary character coding. Beginning in step 7, the browser sends an URL encoded HTTP request to the server, similar to step 1. However, the request does not identify itself as complement aware at the Accept-Charset header; it accepts UTF-8 instead. The input conversion in step 2 and execution of application code in step 3 are the same as in scenario (1). In step 4, the application output is sent to the content negotiation module, which checks the Accept-Charset header of the HTTP request to see if the client web browser is complement aware. Since the web browser in this scenario does not identify itself as complement aware, the output is sent to an HTTP filter, which applies the Default Policy for HTML, while converting its character encoding to UTF-8. For example, the filter can escape tainted characters occurring in HTML tags using HTML numeric character references. Finally, the new output is sent to the browser in step 8 and rendered normally in step 9.

Automatic and Precise Client-Side Protection against CSRF Attacks

Philippe De Ryck, Lieven Desmet, Wouter Joosen, and Frank Piessens

IBBT-DistriNet
Katholieke Universiteit Leuven
3001 Leuven, Belgium
{philippe.deryck,lieven.desmet}@cs.kuleuven.be

Abstract. A common client-side countermeasure against Cross Site Request Forgery (CSRF) is to strip session and authentication information from malicious requests. The difficulty however is in determining when a request is malicious. Existing client-side countermeasures are typically too strict, thus breaking many existing websites that rely on authenticated cross-origin requests, such as sites that use third-party payment or single sign-on solutions.

The contribution of this paper is the design, implementation and evaluation of a request filtering algorithm that automatically and precisely identifies *expected* cross-origin requests, based on whether they are preceded by certain indicators of collaboration between sites. We formally show through bounded-scope model checking that our algorithm protects against CSRF attacks under one specific assumption about the way in which good sites collaborate cross-origin. We provide experimental evidence that this assumption is realistic: in a data set of 4.7 million HTTP requests involving over 20.000 origins, we only found 10 origins that violate the assumption. Hence, the remaining attack surface for CSRF attacks is very small. In addition, we show that our filtering does not break typical non-malicious cross-origin collaboration scenarios such as payment and single sign-on.

Keywords: CSRF, web security, browser security.

1 Introduction

From a security perspective, web browsers are a key component of today's software infrastructure. A browser user might have a session with a trusted site A (e.g. a bank, or a webmail provider) open in one tab, and a session with a potentially dangerous site B (e.g. a site offering cracks for games) open in another tab. Hence, the browser enforces some form of isolation between these two origins A and B through a heterogeneous collection of security controls collectively known as the *same-origin-policy* [18]. An *origin* is a (protocol, domain name, port) triple, and restrictions are imposed on the way in which code and data from different origins can interact. This includes for instance restrictions that prevent scripts from origin B to access content from origin A.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 100–116, 2011.

An important known vulnerability in this isolation is the fact that content from origin B can initiate requests to origin A, and that the browser will treat these requests as being part of the ongoing session with A. In particular, if the session with A was authenticated, the injected requests will appear to A as part of this authenticated session. This enables an attack known as *Cross Site Request Forgery (CSRF)*: B can initiate effectful requests to A (e.g. a bank transaction, or manipulations of the victim's mailbox or address book) without the user being involved.

CSRF has been recognized since several years as one of the most important web vulnerabilities [3], and many countermeasures have been proposed. Several authors have proposed server-side countermeasures [3,4,10]. However, an important disadvantage of server-side countermeasures is that they require modifications of server-side programs, have a direct operational impact (e.g. on performance or maintenance), and it will take many years before a substantial fraction of the web has been updated.

Alternatively, countermeasures can be applied on the client-side, as browser extensions. The basic idea is simple: the browser can strip session and authentication information from malicious requests, or it can block such requests. The difficulty however is in determining when a request is malicious. Existing client-side countermeasures [9,5,12,13,16,19] are typically too strict: they block or strip all cross-origin requests of a specific type (e.g. GET, POST, any). This effectively protects against CSRF attacks, but it unfortunately also breaks many existing websites that rely on authenticated cross-origin requests. Two important examples are sites that use third-party payment (such as PayPal) or single sign-on solutions (such as OpenID). Hence, these existing client-side countermeasures require extensive help from the user, for instance by asking the user to define white-lists of trusted sites or by popping up user confirmation dialogs. This is suboptimal, as it is well-known that the average web user can not be expected to make accurate security decisions.

This paper proposes a novel client-side CSRF countermeasure, that includes an automatic and precise filtering algorithm for cross-origin requests. It is *automatic* in the sense that no user interaction or configuration is required. It is *precise* in the sense that it distinguishes well between malicious and non-malicious requests. More specifically, through a systematic analysis of logs of web traffic, we identify a characteristic of non-malicious cross-origin requests that we call the *trusted-delegation assumption*: a request from B to A can be considered non-malicious if, earlier in the session, A explicitly delegated control to B in some specific ways. Our filtering algorithm relies on this assumption: it will strip session and authentication information from cross-origin requests, unless it can determine that such explicit delegation has happened.

We validate our proposed countermeasure in several ways. First, we formalize the algorithm and the trusted-delegation assumption in Alloy, building on the formal model of the web proposed by [1], and we show through bounded-scope model checking that our algorithm protects against CSRF attacks under this assumption. Next, we provide experimental evidence that this assumption is realistic: through a detailed analysis of logs of web traffic, we quantify how often the trusted-delegation assumption holds, and show that the remaining attack surface for CSRF attacks is very small. Finally, we have implemented our filtering algorithm as an extension of an existing client-side CSRF protection mechanism, and we show that our filtering does not break typical non-malicious cross-origin collaboration scenarios such as payment and single sign-on.

In summary, the contributions of this paper are:

- The design of a novel client-side CSRF protection mechanism based on request filtering.
- A formalization of the algorithm, and formal evidence of the security of the algorithm under one specific assumption, the trusted-delegation assumption.
- An implementation of the countermeasure, and a validation of its compatibility with important web scenarios broken by other state-of-the-art countermeasures.
- An experimental evaluation of the validity of the trusted-delegation assumption.

The remainder of the paper is structured as follows. Section 2 explains the problem using both malicious and non-malicious scenarios. Section 3 discusses our request filtering mechanism. Section 4 introduces the formalization and results, followed by the implementation in Section 5. Section 6 experimentally evaluates the trusted-delegation assumption. Finally, Section 7 extensively discusses related work, followed by a brief conclusion (Section 8).

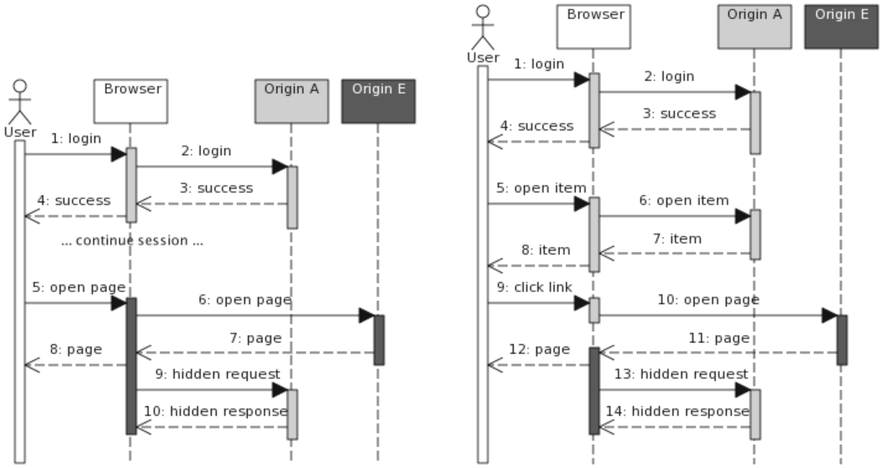
2 Cross-Origin HTTP Requests

The key challenge for a client-side CSRF prevention mechanism is to distinguish malicious from non-malicious cross-origin requests. This section illustrates the difficulty of this distinction by describing some attack scenarios and some important non-malicious scenarios that intrinsically rely on cross-origin requests.

2.1 Attack Scenarios

A1. Classic CSRF Figure 1(a) shows a classic CSRF attack. In steps 1–4, the user establishes an authenticated session with site A, and later (steps 5–8) the user opens the malicious site E in another tab of the browser. The malicious page from E triggers a request to A (step 9), the browser considers this request to be part of the ongoing session with A and automatically adds the necessary authentication and session information. The browser internally maintains different *browsing contexts* for each origin it is interacting with. The shade of the browser-lifeline in the figure indicates the origin associated with the browsing context from which the outgoing request originates (also known as *the referrer*). Since the attack request originates from an E browsing context and goes to origin A, it is *cross-origin*.

For the attack to be successful, an authenticated session with A must exist when the user surfs to the malicious site E. The likelihood of success can be



(a) Classic CSRF

(b) Link injection

Fig. 1. CSRF attack scenarios

increased by making E content-related to A, for instance to attack a banking site, the attacker poses as a site offering financial advice.

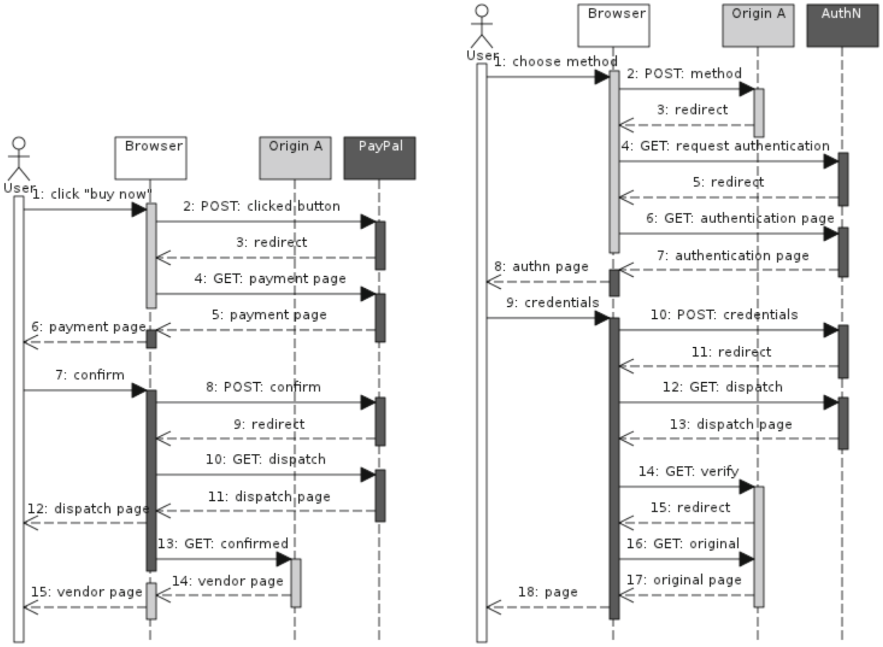
A2. Link Injection. To further increase the likelihood of success, the attacker can inject links to E into the site A. Many sites, for instance social networking sites, allow users to generate content which is displayed to other users. For such a site A, the attacker creates a content item which contains a link to E. Figure 1(b) shows the resulting CSRF scenario, where A is a social networking site and E is the malicious site. The user logs into A (steps 1–4), opens the attacker injected content (steps 5–8), and clicks on the link to E (step 9) which launches the CSRF attack (step 13). Again, the attack request is cross-origin.

2.2 Non-malicious Cross-Origin Scenarios

CSRF attack requests are cross-origin requests in an authenticated session. Hence, forbidding such requests is a secure countermeasure. Unfortunately, this also breaks many useful non-malicious scenarios. We illustrate two important ones.

F1. Payment Provider. Third-party payment providers such as PayPal or Visa 3D-secure offer payment services to a variety of sites on the web.

Figure 2(a) shows the scenario for PayPal's *Buy Now* button. When a user clicks on this button, the browser sends a request to PayPal (step 2), that redirects the user to the payment page (step 4). When the user accepts the payment (step 7), the processing page redirects the browser to the dispatch page (step 10), that loads the landing page of the site that requested the payment (step 13).



(a) Payment scenario

(b) Central authentication scenario

Fig. 2. Non-malicious cross-origin scenarios

Note that step 13 is a cross-origin request from PayPal to A in an authenticated session, for instance a shopping session in web shop A.

F2. Central Authentication. The majority of interactive websites require some form of authentication. As an alternative to each site using its own authentication mechanism, a single sign-on service (such as OpenID or Windows Live ID) provides a central point of authentication.

An example scenario for OpenID authentication using MyOpenID is shown in Figure 2(b). The user chooses the authentication method (step 1), followed by a redirect from the site to the authentication provider (step 4). The authentication provider redirects the user to the login form (step 6). The user enters the required credentials, which are processed by the provider (step 10). After verification, the provider redirects the browser to the dispatching page (step 12), that redirects to the processing page on the original site (step 14). After processing the authentication result, a redirect loads the requested page on the original site (step 16).

Again, note that step 16 is a cross-origin request in an authenticated session.

These two scenarios illustrate that mitigating CSRF attacks by preventing cross-origin requests in authenticated sessions breaks important and useful web scenarios. Existing client-side countermeasures against CSRF attacks [5,13,16] either are incompatible with such scenarios or require user interaction for these cases.

3 Automatic and Precise Request Stripping

The core idea of our new countermeasure is the following: client-side state (i.e. session cookie headers and authentication headers) is stripped from all cross-origin requests, except for *expected* requests. A cross-origin request from origin A to B is *expected* if B previously (earlier in the browsing session) *delegated* to A. We say that B *delegates* to A if B either issues a POST request to A, or if B redirects to A using a URI that contains parameters.

The rationale behind this core idea is that (1) non-malicious collaboration scenarios follow this pattern, and (2) it is hard for an attacker to trick A into delegating to a site of the attacker: forcing A to do a POST or parametrized redirect to an evil site E requires the attacker to either identify a cross-site scripting (XSS) vulnerability in A, or to break into A's webserver. In both these cases, A has more serious problems than CSRF.

Obviously, a GET request from A to B is not considered a delegation, as it is very common for sites to issue GET requests to other sites, and as it is easy for an attacker to trick A into issuing such a GET request (see for instance attack scenario A2 in Section 2).

Unfortunately, the elaboration of this simple core idea is complicated somewhat by the existence of HTTP redirects. A web server can respond to a request with a *redirect* response, indicating to the browser that it should resend the request elsewhere, for instance because the requested resource was moved. The browser will follow the redirect automatically, without user intervention. Redirects are used widely and for a variety of purposes, so we cannot ignore them. For instance, both non-malicious scenarios in Section 2 heavily depend on the use of redirects. In addition, attacker-controlled websites can also use redirects in an attempt to bypass client-side CSRF protection. Akhawe et al. [1] discuss several examples of how attackers can use redirects to attack web applications, including an attack against a CSRF countermeasure. Hence, correctly dealing with redirects is a key requirement for security.

The flowgraph in Figure 3 summarizes our filtering algorithm. For a given request, it determines what session state (cookies and authentication headers) the browser should attach to the request. The algorithm differentiates between simple requests and requests that are the result of a redirect.

Simple Requests. Simple requests that are not cross-origin, as well as expected cross-origin requests are handled as unprotected browsers handle them today. The browser automatically attaches the last known client-side state associated with the destination origin (point 1). The browser does not attach any state to non-expected cross-origin requests (point 3).

Redirect Requests. If a request is the consequence of a redirect response, then the algorithm determines if the redirect points to the origin where the response came from. If this is the case, the client-side state for the new request is limited to the client-side state known to the previous request (i.e. the request that triggered this redirect) (point 2). If the redirect points to another origin, then, depending

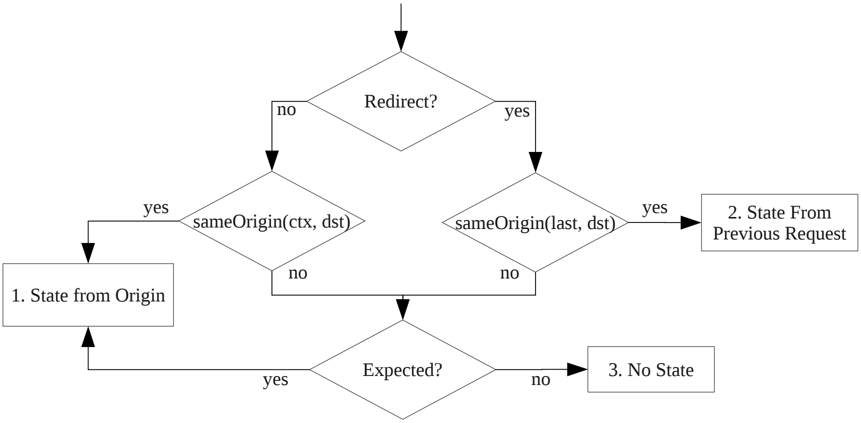


Fig. 3. The request filtering algorithm

on whether this cross-origin request is expected or not, it either gets session-state automatically attached (point 1) or not (point 3).

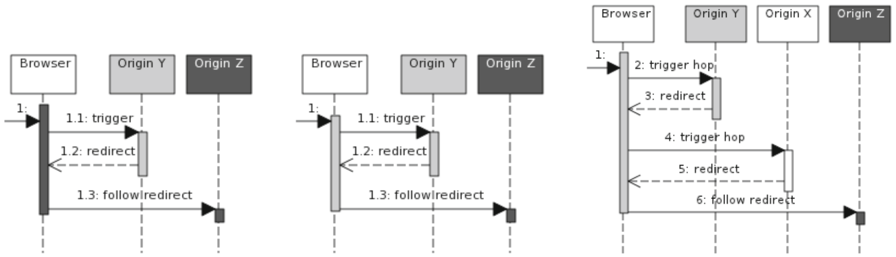
When Is a Request Expected?. A key element of the algorithm is determining whether a request is *expected* or not. As discussed above, the intuition is: a cross-origin request from B to A is expected if and only if A first delegated to B by issuing a POST request to B, or by a parametrized redirect to B. Our algorithm stores such trusted delegations, and an assumption that we rely on (and that we refer to as the *trusted-delegation assumption*) is that sites will only perform such delegations to sites that they trust. In other words, a site A remains vulnerable to CSRF attacks from origins to which it delegates. Section 6 provides experimental evidence for the validity of this assumption.

The algorithm to decide whether a request is expected goes as follows.

For a simple cross-origin request from site B to site A, a trusted delegation from site A to B needs to be present in the delegation store.

For a redirect request that redirects a request to origin Y (light gray) to another origin Z (dark gray) in a browsing context associated with some origin α, the following rules apply.

1. First, if the destination (Z) equals the source (i.e. $\alpha = Z$) (Figure 4(a)), then the request is expected if there is a trusted delegation from Z to Y in the delegation store. Indeed, Y is effectively doing a cross-origin request to Z by redirecting to Z. Since the browsing context has the same origin as the destination, it can be expected not to manipulate redirect requests to misrepresent source origins of redirects (cfr next case).
2. Alternatively, if the destination (Z) is not equal to the source (i.e. $\alpha \neq Z$) (Figure 4(b)), then the request is expected if there is a trusted delegation from Z to Y in the delegation store, since Y is effectively doing a cross-origin request to Z. Now, the browsing context might misrepresent source



(a) (b) (c)

Fig. 4. Complex cross-origin redirect scenarios

origins of redirects by including additional redirect hops (origin X (white) in Figure 4(c)). Hence, our decision to classify the request does not involve X.

Finally, our algorithm imposes that expected cross-origin requests can only use the GET method and that only two origins can be involved in the request chain. These restrictions limit the potential power an attacker might have, even if the attacker successfully deceives the trusted-delegation mechanism.

Mapping to Scenarios. The reader can easily check that the algorithm blocks the attack scenarios from Section 2, and supports the non-malicious scenarios from that section. We discuss two of them in more detail.

In the PayPal scenario (Figure 2(a)), step 13 needs to re-use the state already established in step 2, which means that according to the algorithm, the request from PayPal to A should be expected. A trusted delegation happens in step 2, where a cross-origin POST is sent from origin A to PayPal. Hence the GET request in step 13 is considered expected and can use the state associated with origin A. Also note how the algorithm maintains the established session with PayPal throughout the scenario. The session is first established in step 3. Step 4 can use this session, because the redirect is an internal redirect on the PayPal origin. Step 8 can use the last known state for the PayPal origin and step 10 is yet another internal redirect.

In the link injection attack (Figure 1(b)), the attack happens in step 13 and is launched from origin E to site A. In this scenario, an explicit link between A and E exists because of the link injected by the attacker. This link is however not a POST or parametrized redirect, so it is not a trusted delegation. This means that the request in step 10 is not considered to be expected, so it can not access the previously established client-side state, and the attack is mitigated.

4 Formal Modeling and Checking

The design of web security mechanisms is complex: the behaviour of (same-origin and cross-origin) browser requests, server responses and redirects, cookie and session management, as well as the often implicit threat models of web security can lead to subtle security bugs in new features or countermeasures. In order to evaluate proposals for new web mechanisms more rigorously, Akhawe et al. [1] have proposed a model of the Web infrastructure, formalized in Alloy.

The base model is some 2000 lines of Alloy source code, describing (1) the essential characteristics of browsers, web servers, cookie management and the HTTP protocol, and (2) a collection of relevant threat models for the web. The Alloy Analyzer – a bounded-scope model checker – can then produce counterexamples that violate intended security properties if they exist in a specified finite scope.

In this section, we briefly introduce Akhawe's model and present our extensions to the model. We also discuss how the model was used to verify the absence of attack scenarios and the presence of functional scenarios.

4.1 Modeling Our Countermeasure

The model of Akhawe et al. defines different principals, of which `GOOD` and `WEBATTACKER` are most relevant. `GOOD` represents an honest principal, who follows the rules imposed by the technical specifications. A `WEBATTACKER` is a malicious user who can control malicious web servers, but has no extended networking capabilities.

The concept of `Origin` is used to differentiate between origins, which correspond to domains in the real world. An origin is linked with a server on the web, that can be controlled by a principal. The browsing context, modeled as a `ScriptContext`, is also associated with an `origin`, that represents the origin of the currently loaded page, also known as the referrer.

A `ScriptContext` can be the source of a set of `HTTPTransaction` objects, which are a pair of an `HTTPRequest` and `HTTPResponse`. An HTTP request and response are also associated with their remote destination origin. Both an HTTP request and response can have headers, where respectively the `CookieHeader` and `SetCookieHeader` are the most relevant ones. An HTTP request also has a `method`, such as `GET` or `POST`, and a `queryString`, representing URI parameters. An HTTP response has a `statusCode`, such as `c200` for a content result or `c302` for a redirect. Finally, an HTTP transaction has a cause, which can be none, such as the user opening a new page, a `RequestAPI`, such as a scripting API, or another `HTTPTransaction`, in case of a redirect.

To model our approach, we need to extend the model of Akhawe et al. to include (a) the accessible client-side state at a certain point in time, (b) the trusted delegation assumption and (c) our filtering algorithm. We discuss (a) and (b) in detail, but due to space constraints we omit the code for the filtering algorithm (c), which is simply a literal implementation of the algorithm discussed in Section 3.

Client-Side State. We introduced a new signature `CSState` that represents a client-side state (Listing 1.1). Such a state is associated with an `Origin` and contains a set of `Cookie` objects. To associate a client-side state with a given request or response and a given point in time, we have opted to extend the `HTTPTransaction` from the original model into a `CSStateHTTPTransaction`. Such an extended transaction includes a `beforeState` and `afterState`, respectively representing the accessible client-side state at the time of sending the request and the updated client-side state after having received the response. The `afterState` is equal to the `beforeState`, with the potential addition of new cookies, set in the response.

```
sig CSState {
  dst: Origin,
  cookies: set Cookie
}

sig CSStateHTTPTransaction extends HTTPTransaction {
  beforeState : CSState,
  afterState : CSState
} {
 //The after state of a transaction is equal to the before state + any additional cookies set in the response
 beforeState·dst = afterState·dst
 afterState·cookies = beforeState·cookies + (resp·headers & SetCookieHeader)·thecookie

 // The destination origin of the state must correspond to the transaction destination origin
    beforeState·dst = req·host
}
```

Listing 1.1. Signatures representing our data in the model

Trusted-delegation Assumption. We model the trusted-delegation assumption as a fact, that honest servers do not send a POST or parametrized redirect to web attackers ((Listing 1.2).

```
fact TrustedDelegation {
  all r : HTTPRequest | {
   (r·method = POST || some (req·r)·cause & CSStateHTTPTransaction)
     &&
  ((some (req·r)·cause & CSStateHTTPTransaction && getPrincipalFromOrigin[(req·r)·cause·req·host] in GOOD) || getPrincipalFromOrigin[transactions·(req·r)·owner] in GOOD)
   implies
     getPrincipalFromOrigin[r·host] not in WEBATTACKER
  }
}
```

Listing 1.2. The fact modeling the trusted-delegation assumption

4.2 Using Model Checking for Security and Functionality

We formally define a CSRF attack as the possibility for a web attacker (defined in the base model) to inject a request with at least one existing cookie attached to it (this cookie models the session/authentication information attached to requests) in a session between a user and an honest server (Listing 1.3).

We provided the Alloy Analyzer with a universe of at most 9 HTTP events and where an attacker can control up to 3 origins and servers (a similar size as used in [1]). In such a universe, no examples of an attacker injecting a request through the user's browser were found. This gives strong assurance that the countermeasure does indeed protect against CSRF under the trusted delegation assumption.

```
pred CSRF[r : HTTPRequest] {
   //Ensure that the request goes to an honest server
   some getPrincipalFromOrigin[r·host]
   getPrincipalFromOrigin[r·host] in GOOD

    //Ensure that an attacker is involved in the request
    some (WEBATTACKER·servers & involvedServers[req·r]) || getPrincipalFromOrigin[(
          transactions·(req·r))·owner] in WEBATTACKER

    // Make sure that at least one cookie is present
    some c : (r·headers & CookieHeader)·thecookie | {
       //Ensure that the cookie value is fresh (i·e· that it is not a renewed value in a redirect
             chain)
       not c in ((req·r)·*cause·resp·headers & SetCookieHeader)·thecookie
    }
}
```

Listing 1.3. The predicate modeling a CSRF attack

We also modeled the non-malicious scenarios from Section 2, and the Alloy Analyzer reports that these scenarios are indeed permitted. From this, we can also conclude that our extension of the base model is consistent.

Space limitations do not permit us to discuss the detailed scenarios present in our model, but the interested reader can find the complete model available for download at [6].

Table 1. CSRF benchmark

	Test scenarios	Result
HTML	29 cross-origin test scenarios	✓
CSS	12 cross-origin test scenarios	✓
ECMAScript	9 cross-origin test scenarios	✓
Redirects	20 cross-origin redirect scenarios	✓

5 Implementation

We have implemented our request filtering algorithm in a proof-of-concept add-on for the Firefox browser, and used this implementation to conduct an extensive practical evaluation. First, we have created simulations for both the common attack scenarios as well as the two functional scenarios discussed in the paper (third party payment and centralized authentication), and verified that they behaved as expected.

Second, in addition to these simulated scenarios, we have verified that the prototype supports actual instances of these scenarios, such as for example the use of MyOpenID authentication on sourceforge.net.

Third, we have constructed and performed a CSRF benchmark[1], consisting of 70 CSRF attack scenarios to evaluate the effectiveness of our CSRF prevention

[1] The benchmark can be applied to other client-side solutions as well, and is downloadable at [6].

technique (see Table 1). These scenarios are the result of a CSRF-specific study of the HTTP protocol, the HTML specification and the CSS markup language to examine their cross-origin traffic capabilities, and include complex redirect scenarios as well. Our implementation has been evaluated against each of these scenarios, and our prototype passed all tests successfully.

The prototype, the scenario simulations and the CSRF benchmark suite are available for download [6].

6 Evaluating the Trusted-Delegation Assumption

Our countermeasure drastically reduces the attack surface for CSRF attacks. Without CSRF countermeasures in place, an origin can be attacked by any other origin on the web. With our countermeasure, an origin can only be attacked by another origin to which it has delegated control explicitly by means of a cross-origin POST or redirect. We have already argued in Section 3 that it is difficult for an attacker to cause unintended delegations. In this section, we measure the remaining attack surface experimentally.

We conducted an extensive traffic analysis using a real-life data set of 4.729.217 HTTP requests, collected from 50 unique users over a period of 10 weeks. The analysis revealed that 1.17% of the 4.7 million requests are treated as delegations in our approach. We manually analyzed all these 55.300 requests, and classified them in the interaction categories summarized in Table 2.

For each of the categories, we discuss the resulting attack surface:

Third Party Service Mashups. This category consists of various third party services that can be integrated in other websites. Except for the single sign-on services, this is typically done by script inclusion, after which the included script can launch a sequence of cross-origin GET and/or POST requests towards offered AJAX APIs. In addition, the service providers themselves often use cross-origin redirects for further delegation towards content delivery networks.

As a consequence, the origin A including the third-party service S becomes vulnerable to CSRF attacks from S. This attack surface is unimportant, as in these scenarios, S can already attack A through script inclusion, a more powerful attack than CSRF.

In addition, advertisement service providers P that further redirect to content delivery services D are vulnerable to CSRF attacks from D whenever a user clicks an advertisement. Again, this attack surface is unimportant: the delegation from P to D correctly reflects a level of trust that P has in D, and P and D will typically have a legal contract or SLA in place.

Multi-origin Websites. Quite a number of larger companies and organizations have websites spanning multiple origins (such as *live.com* - *microsoft.com* and *google.be* - *google.com*). Cross-origin POST requests and redirects between these origins make it possible for such origins to attack each other. For instance, *google.be* could attack *google.com.* Again, this attack

Table 2. Analysis of the trusted-delegation assumption in a real-life data set of 4.729.217 HTTP requests

	# requests	POST	redir.
Third party service mashups	29.282 (52,95%)	5.321	23.961
Advertisement services	*22.343 (40,40%)*	*1.987*	*20.356*
Gadget provider services (appspot, mochibot, gmodules, ...)	*2.879 (5,21%)*	*2.757*	*122*
Tracking services (metriweb, sitestat, uts.amazon, ...)	*2.864 (5,18%)*	*411*	*2.453*
Single Sign-On services (Shibboleth, Live ID, OpenId, ...)	*1.156 (2,09%)*	*137*	*1.019*
3rd party payment services (Paypal, Ogone)	*27 (0,05%)*	*19*	*8*
Content sharing services (addtoany, sharethis, ...)	*13 (0,02%)*	*10*	*3*
Multi-origin websites	13.973 (25,27%)	198	13.775
Content aggregators	8.276 (14,97%)	0	8.276
Feeds (RSS feeds, News aggregators, mozilla fxfeeds, ...)	*4.857 (8,78%)*	*0*	*4.857*
Redirecting search engines (Google, Comicranks, Ohnorobot)	*3.344 (6,05%)*	*0*	*3.344*
Document repositories (ACM digital library, dx.doi.org, ...)	*75 (0,14%)*	*0*	*75*
False positives (wireless network access gateways)	1.215 (2,20%)	12	1.203
URL shorteners (gravatar, bit.ly, tinyurl, ...)	759 (1,37%)	0	759
Others (unclassified)	1.795 (3,24%)	302	1.493
Total number of 3rd party delegation initiators	**55.300 (100%)**	**5.833**	**49.467**

surface is unimportant, as all origins of such a multi-origin website belong to a single organization.

Content Aggregators. Content aggregators collect searchable content and redirect end-users towards a specific content provider. For news feeds and document repositories (such as the ACM digital library), the set of content providers is typically stable and trusted by the content aggregator, and therefore again a negligible attack vector.

Redirecting search engines register the fact that a web user is following a link, before redirecting the web user to the landing page (e.g. as Google does for logged in users). Since the entries in the search repository come from all over the web, our CSRF countermeasure provides little protection for such search engines. Our analysis identified 4 such origins in the data set: *google.be*, *google.com*, *comicrank.com*, and *ohnorobot.com*.

False Positives. Some fraction of the cross-origin requests are caused by network access gateways (e.g. on public Wifi) that intercept and reroute requests towards a payment gateway. Since such devices have man-in-the-middle capabilities, and hence more attack power than CSRF attacks, the resulting attack surface is again negligible.

URL Shorteners. To ease URL sharing, URL shorteners transform a shortened URL into a preconfigured URL via a redirect. Since such URL shortening services are open, an attacker can easily control a new redirect target. The effect is similar to the redirecting search engines; URL shorteners are essentially left unprotected by our countermeasure. Our analysis identified 6 such services in the data set: *bit.ly*, *gravatar.com*, *post.ly*, *tiny.cc*, *tinyurl.com*, and *twitpic.com*.

Others(unclassified). For some of the requests in our data set, the origins involved in the request were no longer online, or the (partially anonymized) data did not contain sufficient information to reconstruct what was happening, and we were unable to classify or further investigate these requests.

In summary, our experimental analysis shows that the trusted delegation assumption is realistic. Only 10 out of 23.592 origins (i.e. 0.0042% of the examined origins) – the redirecting search engines and the URL shorteners – perform delegations to arbitrary other origins. They are left unprotected by our countermeasure. But the overwhelming majority of origins delegates (in our precise technical sense, i.e. using cross-origin POST or redirect) only to other origins with whom they have a trust relationship.

7 Related Work

The most straightforward protection technique against CSRF attacks is server-side mitigation via validation tokens [4,10]. In this approach, web forms are augmented with a server-generated, unique validation token (e.g. embedded as a hidden field in the form), and at form submission the server checks the validity of the token before executing the requested action. At the client-side, validation tokens are protected from cross-origin attackers by the same-origin-policy, distinguishing them from session cookies or authentication credentials that are automatically attached to any outgoing request. Such a token based approach can be offered as part of the web application framework [14,7], as a server-side library or filter [15], or as a server-side proxy [10].

Recently, the `Origin` header has been proposed as a new server-side countermeasure [3,2]. With the `Origin` header, clients unambiguously inform the server about the origin of the request (or the absence of it) in a more privacy-friendly way than the `Referer` header. Based on this origin information, the server can safely decide whether or not to accept the request. In follow-up work, the `Origin` header has been improved, after a formal evaluation revealed a previously unknown vulnerability [1]. The Alloy model used in this evaluation also formed the basis for the formal validation of our presented technique in Section 4.

Unfortunately, the adoption rate of these server-side protection mechanisms is slow, giving momentum to client-side mitigation techniques as important (but hopefully transitional) solutions. In the next paragraphs, we will discuss the client-side proxy RequestRodeo, as well as 4 mature and popular browser addons (CsFire, NoScript ABE, RequestPolicy, and CSD[2]). In addition, we will evaluate how well the browser addons enable the functional scenarios and protect against the attack scenarios discussed in this paper (see Table 3).

RequestRodeo [9] is a client-side proxy proposed by Johns and Winter. The proxy applies a client-side token-based approach to tie requests to the correct source origin. In case a valid token is lacking for an outgoing request, the request is considered suspicious and gets stripped of cookies and HTTP `authorization` headers. RequestRodeo lies at the basis of most of the client-side CSRF solutions [5,12,13,16,19], but because of the choice of a proxy, RequestRodeo often lacks context information, and the rewriting technique on raw responses does not scale well in a web 2.0 world.

[2] Since the client-side detection technique described in [17] is not available for download, the evaluation is done based on the description in the paper.

Table 3. Evaluation of browser-addons

	Functional scenarios		Attack scenarios	
	F1. Payment Provider	F2. Central Authentication	A1. Classic CSRF	A2. Link Injection
CsFire [5]	×	×	✓	✓
NoScript ABE [13] [a]	×	×	✓	✓
RequestPolicy [16]	⋈ [b]	⋈ [b]	✓[c]	✓[c]
Client-Side Detection [17]	×	×	✓	✓
Our Approach	✓	✓	✓	✓

[a] ABE configured as described in [8]

[b] Requires interactive feedback from end-user to make the decision

[c] Requests are blocked instead of stripped, impacting the end-user experience

CsFire [5] extends the work of Maes *et al.* [11], and strips cookies and HTTP `authorization` headers from a cross-origin request. The advantage of stripping is that there are no side-effects for cross-origin requests that do not require credentials in the first place. CsFire operates autonomously by using a default client policy which is extended by centrally distributed policy rules. Additionally, CsFire supports users creating custom policy rules, which can be used to blacklist or whitelist certain traffic patterns. Without a central or user-supplied whitelist, CsFire does not support the payment and central authentication scenario.

To this extent, we plan to integrate the approach presented in this paper to the CsFire Mozilla Add-On distribution in the near future.

NoScript ABE [13], or Application Boundary Enforcer, restricts an application within its origin, which effectively strips credentials from cross-origin requests, unless specified otherwise. The default ABE policy only prevents CSRF attacks from the internet to an intranet page. The user can add specific policies, such as a CsFire-alike stripping policy [8], or a site-specific blacklist or whitelist. If configured with [8], ABE successfully blocks the three attack scenarios, but disables the payment and central authentication scenario.

RequestPolicy [16] protects against CSRF by blocking all cross-origin requests. In contrast to stripping credentials, blocking a request can have a very noticeable effect on the user experience. When detecting a cross-origin redirect, RequestPolicy injects an intermediate page where the user can explicitly allow the redirect. RequestPolicy also includes a predefined whitelist of hosts that are allowed to send cross-origin requests to each other. Users can add exceptions to the policy using a whitelist. RequestPolicy successfully blocks the three attack scenarios (by blocking instead of stripping all cross-origin requests) and requires interactive end-user feedback to enable the payment and central authentication scenario.

Finally, in contrast to the CSRF *prevention* techniques discussed in this paper, Shahriar and Zulkernine proposes a client-side *detection* technique for CSRF [17]. In their approach, malicious and benign cross-origin requests are distinguished from each other based on the existence and visibility of the submitted form or link in the originating page, as well as the visibility of the target. In addition, the expected content type of the response is taken into account to detect false negatives during execution. Although the visibility check closely approximates the end-user intent, their technique fails to support the script inclusions of third party service mashups as discussed in Section 6. Moreover, without taking into account the delegation requests, expected redirect requests (as defined in Section 3) will be falsely detected as CSRF attacks, although these requests are crucial enablers for the payment and central authentication scenario.

8 Conclusion

We have proposed a novel technique for protecting at client-side against CSRF attacks. The main novelty with respect to existing client-side countermeasures is the good trade-off between security and compatibility: existing countermeasures break important web scenarios such as third-party payment and single-sign-on, whereas our countermeasure can permit them.

Acknowledgements. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT, IWT, the Research Fund K.U. Leuven and the EU-funded FP7-projects WebSand and NESSoS.

References

1. Akhawe, D., Barth, A., Lam, P.E., Mitchell, J., Song, D.: Towards a formal foundation of web security. In: IEEE Computer Security Foundations Symposium, pp. 290–304 (2010)
2. Barth, A., Jackson, C., Hickson, I.: The web origin concept (November 2010), `http://tools.ietf.org/html/draft-abarth-origin-09`
3. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: 15th ACM Conference on Computer and Communications Security, CCS 2008 (2008)
4. Burns, J.: Cross site reference forgery: An introduction to a common web application weakness. In: Security Partners, LLC (2005)
5. De Ryck, P., Desmet, L., Heyman, T., Piessens, F., Joosen, W.: CsFire: Transparent client-side mitigation of malicious cross-domain requests. In: Massacci, F., Wallach, D., Zannone, N. (eds.) ESSoS 2010. LNCS, vol. 5965, pp. 18–34. Springer, Heidelberg (2010)
6. De Ryck, P., Desmet, L., Piessens, F., Joosen, W.: Automatic and precise client-side protection against csrf attacks - downloads (2011), `https://distrinet.cs.kuleuven.be/software/CsFire/esorics2011/`
7. Django. Cross site request forgery protection (2011), `http://docs.djangoproject.com/en/dev/ref/contrib/csrf/`
8. Informaction Forums. Which is the best way to configure ABE? (July 2010), `http://forums.informaction.com/viewtopic.php?f=23\&t=4752`
9. Johns, M., Winter, J.: RequestRodeo: client side protection against session riding. In: Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448, pp. 5–17 (2006)
10. Jovanovic, N., Kirda, E., Kruegel, C.: Preventing cross site request forgery attacks. In: IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), pp. 1–10 (2006)

11. Maes, W., Heyman, T., Desmet, L., Joosen, W.: Browser protection against cross-site request forgery. In: Proceedings of the First ACM Workshop on Secure Execution of Untrusted Code, pp. 3–10. ACM, New York (2009)
12. Mao, Z., Li, N., Molloy, I.: Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 238–255. Springer, Heidelberg (2009)
13. Giorgio Maone. Noscript 2.0.9.9 (2011), `http://noscript.net/`
14. Ruby on Rails. Actioncontroller::requestforgeryprotection (2011), `http://api.rubyonrails.org/classes/ActionController/RequestForgeryProtection.html`
15. Owasp. Csrf guard (October 2008), `http://www.owasp.org/index.php/CSRF_Guard`
16. Samuel, J.: Requestpolicy 0.5.20 (2011), `http://www.requestpolicy.com`
17. Shahriar, H., Zulkernine, M.: Client-side detection of cross-site request forgery attacks. In: 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE), pp. 358–367 (November 2010)
18. Zalewski, M.: Browser security handbook (2010), `http://code.google.com/p/browsersec/wiki/Main`
19. Zeller, W., Felten, E.W.: Cross-site request forgeries: Exploitation and prevention. Technical report, Princeton University (2008)

Timing Is Everything: The Importance of History Detection

Gunnar Kreitz

KTH – Royal Institute of Technology
gkreitz@kth.se

Abstract. In this work, we present a *Flow Stealing* attack, where a victim's browser is redirected during a legitimate flow. One scenario is redirecting the victim's browser as it moves from a store to a payment provider. We discuss two attack vectors.

Firstly, browsers have long admitted an attack allowing a malicious web page to detect whether the browser has visited a target web site by using CSS to style visited links and read out the style applied to a link. For a long time, this CSS history detection attack was perceived as having small impact. Lately, highly efficient implementations of the attack have enabled malicious web sites to extract large amounts of information. Following this, browser developers have deployed measures to protect against the attack. Flow stealing demonstrates that the impact of history detection is greater than previously known.

Secondly, an attacker who can mount a man-in-the-middle attack against the victim's network traffic can also perform a flow stealing attack.

Noting that different browsers place different restrictions on cross-frame navigation through JavaScript window handles, we suggest a stricter policy based on pop-up blockers to prevent Flow Stealing attacks.

Keywords: Web Security, Flow Stealing, CSS History Detection.

1 Introduction

In this paper we discuss an attack related to when a user's session is transfered between two different sites. One scenario in which such transfers occur is when a user moves from a store, `store.com`, to a payment provider, `pay.com`. We use this as a running example throughout the paper.

A typical integration mechanism is that the store sends information about the purchase to the payment provider (at least the total amount to be paid) and gets a transaction ID. The store then redirects the user to the payment provider with the transaction ID, either by a GET or POST request[1]. In this paper, we outline

[1] Several payment providers also provide lightweight integration where the store directly redirects the customer with information about the purchase instead of a transaction ID. This does not materially affect the attack, so we consider this equivalent to sending a transaction ID.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 117–132, 2011.

an attack where an attacker at this point redirects the user's browser to the same payment provider, but with a different transaction ID. The attacker could also choose to redirect the user to a malicious site stealing the user's credit card details, at the risk of such a redirect being more easily detectable by the victim. We refer to this class of hijacking the user's session as it transfers cross-domain as *Flow Stealing*.

The steps in a typical version of the attack are as follows:

1. Victim visits `evil.com`, and follows link to `store.com`
2. Victim interacts with `store.com`, eventually reaching checkout
3. `store.com` creates transaction on `pay.com`, which assigns transaction ID i_u
4. `store.com` redirects victim to `pay.com` with transaction ID i_u.
5. `evil.com` detects that victim hits `pay.com`
6. `evil.com` creates transaction on `pay.com`, which assigns transaction ID i_a
7. `evil.com` redirects the victim's tab to `pay.com` with transaction ID i_a

To the victim, the flow appears normal. She follows a link to `store.com` which opens in a new tab. The site is legitimate, so all interaction and security indicators such as certificates function as they would normally. When she goes to pay, she is transfered to the legitimate `pay.com` site, also with intact security indicators. The only indicator of the attack is in the payment information displayed by `pay.com`. What the difference is, how prominent it is, or if there even is one, depends on the information associated with the transaction that `pay.com` displays. This in turn often depends on the amount of detail about the purchase communicated from `store.com` to `pay.com` when it initializes the transaction.

Two questions arise: firstly, how does the attacker redirect the browser, and secondly, how does the attacker know when to redirect? We address these in Section 2 and Section 3. In one version, our attacker makes use of an old and well-known security hole, CSS history detection [1], in order to time her attack. To be able to redirect the victim's browser, the attacker needs JavaScript running in the browser and a window handle to the window which is to be redirected.

1.1 Attacker and Victim Model

We consider two forms of attackers: an attacker running a web page, and an attacker who can mount man-in-the-middle (MITM) attacks against the victim's network traffic. Our primary focus is on an attacker operating a web site, `evil.com`, visited by the victim. We assume that the attacker can convince victims to click on a link from `evil.com` to `store.com` and buy something. This means that our attacker could make some money (legally) by hosting advertisements or participating in an affiliate program. We remark that our attacker is *weaker* compared to the traditional attacker model in many CSRF and XSS attacks, as the attacker only needs the victim to follow a *legitimate* link to a well-known site.

We also consider a *network attacker* who can intercept and modify the victim's network traffic. There are several ways in which an attacker could get this ability. For an attacker on the same local network as the victim, the attacker can utilize standard tools such as ARP or DHCP spoofing to get access to the victim's traffic. Alternatively, an attacker could set up a Tor exit node and thereby mount MITM attacks against anonymous victims. Given MITM access to the victim's network communication, all information sent and received over http can trivially be attacked, but our focus is on pages protected by https, a protocol intended to protect against network attacks. We do not assume that the network attacker can trick the victim into visiting her web site, so the network attacker is not strictly stronger than our normal attacker.

We consider a potential victim of our attack who follows the guidelines taught by the security community. She will only provide sensitive information over https, but not before verifying that the certificate is authentic. In addition to a security-conscious victim, we assume that the attacked flow is on domains served only over https.

1.2 Our Contribution

In this paper, we describe a new type of attack which we call flow stealing. Our attack makes new use of a well-known security issue in the CSS specification to time the execution of a redirection attack. By timing the redirection precisely, the attacker can give the victim a false sense of security by having her browse well-known sites before the attack is executed. This new use of an old attack emphasizes the importance of closing also minor security holes where the impact is not fully understood. Most major browsers have now closed the CSS history detection hole in their latest stable versions. However, flow stealing attacks can also be performed as a man-in-the-middle attack. Our flow stealing attack highlights a part of typical web flows which is difficult to protect using current mechanisms, namely legitimate cross-domain redirects.

We identify several scenarios in which the attack can be mounted, and we suggest new protection mechanisms which can be used to prevent flow stealing, as well as similar attacks. In particular, we point out the dangers of allowing JavaScript to navigate and close windows to which it holds a window handle and propose a new policy based on pop-up blocking.

1.3 Related Works

Flow stealing shares some similarities with cross-site request forgery (CSRF) and session fixation attacks. A related form of CSRF is the login CSRF attack, described by Barth *et al.* [2]. In a login CSRF attack, the attacker logs the victim on to a legitimate site using an account controlled by the attacker. The purpose of this is for the attacker to extract or use information stored by the victim's activity on the site. Examples of such abuse includes stealing the search history of the victim, or using stored credit card details to transfer money or make purchases.

As discussed in [2], the login CSRF attack is an example of vulnerabilities in session initialization. Another type of vulnerability in the same class is that of session fixation, where the attacker tricks the victim into logging in on a legitimate site with a session ID known to the attacker. The attacker can then visit the legitimate site using the same session ID and then be logged in as the victim.

Also similar in spirit to flow stealing is the tabnabbing attack by Raskin [3]. In this attack, a malicious site detects when the victim is not looking at it and then replaces its content with a phishing site looking like a login or error message page at a legitimate site.

2 Redirecting the Victim's Tab

How can the attacker redirect the victim's browser? Firstly, this requires the attacker to get the victim's browser to run malicious JavaScript. This is easily accomplished for an attacker who convinces the victim to visit `evil.com`, as the page can contain the JavaScript required for the attack. A network attacker using a man-in-the-middle attack can insert malicious JavaScript into any page or script content served over unprotected http. For more details, see Section 2.2

Furthermore, the script needs to have a window handle to the tab in which the victim is visiting `store.com`, and later `pay.com`. If the victim opened the tab by clicking a link on `evil.com`, the attacker's JavaScript can store a window handle to the tab. We defer discussion of the man-in-the-middle case to Section 2.2.

Many browsers permit JavaScript to freely navigate any top-level window handles it holds. One notable exception is Opera which does not allow a window w_1 to navigate a window w_2 to which it has a handle if w_2 is currently browsed to a https page at a domain different from w_1. There is a simple way for our attack to get around this restriction in Opera, but it does make the attack easier to detect for the victim. We discuss the circumvention in Section 2.1.

We remark that once an attacker's JavaScript has a window handle to a window, it retains its rights over that window regardless of what happens. In particular, a user manually typing in a different address in the navigation bar does not revoke any of the opener's privileges.

2.1 Working around Opera's Navigation Restrictions

Opera prevents a window from navigating another window via a window handle in some scenarios. In our flow stealing attack, we need to change the address of the victim's window when it goes to `pay.com`, which we assume is served over https. Thus, we propose a slightly different variation when attacking the Opera browser.

If a window w_1 wants to navigate the window w_2 to some address, it can accomplish a similar effect which may not be noticed if it closes window w_2 and navigates itself to the address it wanted window w_2 to go to. We are not aware of any browser placing restrictions on closing windows via a JavaScript handle. Depending on the victim's configuration and how many tabs she has open, this "navigation" may be more or less noticeable.

If the attacker can close window w_2, why not simply open another window with the right address in its place? The answer is that such an attempt will likely be prevented by a pop-up blocker. All mainstream browsers today prevent sites from arbitrarily opening new tabs, unless the action is initiated by a user action such as a mouse click.

2.2 Page Modification by a Network Attacker

In our attacker model, we consider a network attacker who is not assumed to be able to entice victims to visit her web site. Thus, the network attacker needs some other way to get JavaScript running in the victim's browser, as well as a window handle to a window where the victim then makes a purchase.

Most web browsing is still done over http, instead of https. However, we assume that both `store.com` and `pay.com` have invested in security and are served only over https. Thus, the network attacker cannot perform man-in-the-middle attacks against these domains directly.

Our network attacker can, however, easily modify any other page the victim visits over http. Thus, an attacker can write a proxy inserting malicious JavaScript into all pages the victim visits over http. To make this attack efficient, we assume that the attacker wants to adapt the JavaScript as little as possible to the page the attack is inserted into.

We begin with a discussion on what the JavaScript should do. We assume that the network attacker wants to avoid detection, and thus not modify any user-visible behavior of web sites. This means that she will want to insert JavaScript on the page such that it captures a window reference to any window opened by the page. A page can be opened for one of two reasons, either by the user clicking on a link with the `target` attribute set to "`_blank`", or by JavaScript on the page calling `window.open`.

Thus, the attack flow for our network attacker is as follows:

1. Victim visits `http://example.com`
2. Attacker's proxy inserts JavaScript into returned `example.com` page
3. Victim clicks on link to `example2.com`, opening in new window
4. Attacker's JavaScript captures a reference to the opened window

in which situation the network attacker is almost in the same position as when the victim visits `evil.com` and follows a link from there.

We start with links using the `target` attribute to open a new window. The attacker can insert JavaScript which executes when the page is loaded, and which loops through all anchor tags on the web page. When it reaches an anchor tag with `target` set to `_blank`, it modifies the tag to call a JavaScript function opening the window and storing the window handle when clicked. We remark that as these tags are easily detectable if the attacker parses the page, it would be easy to make this modification statically as part of a man-in-the middle attack as well. We present a simplified JavaScript example in Figure 1.

Handling windows opened by JavaScript on the original web site at first appears more difficult. To detect when windows may be opened could involve dynamic analysis of JavaScript code. However, there is an easy way to capture references opened by JavaScript on the original page.

```
window.real_open = window.open;
window.open = function(URL, name, specs, replace) {
   var openedWindow = real_open.apply(this, arguments);
   storeReferenceAndStartTiming(openedWindow);
   return openedWindow;
}

function modifyLinks() {
   var links = document.getElementsByTagName("a");
   for (i=0; i<links.length; i++) {
      if(links[i].getAttribute("target") == "_blank") {
         links[i].setAttribute("onClick", "window.open(\"" +
links[i].getAttribute("href") + "\"); return false;");
      }
   }
}
window.onload = modifyLinks;
```

Fig. 1. Simplified JavaScript code to capture window references from non-malicious pages

To do this, we use a technique which has been used by Phung *et al.* [4] to construct a security mechanism for policy enforcement in JavaScript. The technique is based on the observation that even built-in functions can be aliased by user-defined functions in JavaScript. Thus, the malicious JavaScript can replace the `window.open` method with a JavaScript function which calls the original `window.open` method and stores a copy of the returned window handle before returning it to the caller. Slightly simplified JavaScript code illustrating the principle is shown in Figure 1.

3 Timing the Attack

We now turn to the question of how the attacker can learn when the victim is redirected to `pay.com`. We present two mechanisms for accomplishing this. The first, and easiest method builds on the well-known CSS history detection attack to periodically poll whether the `pay.com` URL has become visited. The second method is based on traffic analysis by a network attacker.

3.1 CSS History Detection

An early feature in web browsers is the distinction between a visited and an unvisited link. With the advent of Cascading Style Sheets (CSS), the creator of a web site gained the ability to decide how the two types of links would be rendered. It was soon realized [5] that this feature could be abused by a web site to determine of its visitor had also visited some other site. The CSS 2.1 specification [6, Section 5.11.2] notes the vulnerability and states that browsers may treat all links as unvisited or implement other counter-measures.

We remark that while an attacker can test if a visitor has visited a specific URL, she cannot extract the full browsing history of the visitor. In particular, she does not learn anything about URLs she cannot guess. The rate at which the attacker can test URLs is also an issue as it limits the privacy exposure of the attack. Here, the increasing prevalence of Web 2.0 applications and the accompanying optimization in general JavaScript performance has benefited an attacker. Speeds of 30000 tested URLs/second have been reported by Janc and Olejnik [1] with their optimized version of the attack.

Recall that the integration with a payment provider is typically done by setting up a transaction and then redirecting the user to the payment provider with a unique transaction ID assigned by the payment provider. The attacker is not able to predict the transaction ID, so if it had been a part of the URL, the attacker would not be able to use the CSS history detection attack to learn when the user visited the payment provider. However, common practice is to send the transaction ID to the payment provider as a POST parameter to a static URL, which allows our attack to work.

History detection attacks have been studied in the academic literature, and several demonstration web sites [7,8] have been created to raise awareness of the issue. Wondracek *et al.* [9] showed that stolen history data can also be used for a de-anonymization attack against users of social network sites. Jakobsson and Stamm [10] discussed the potential of using history detection in phishing attacks. Benevolent uses of the history detection attack have also been discussed. One example is to guess at which OpenID provider a user has to ease OpenID-logins [11], and another is to detect if a user has visited malicious sites and may have had malware installed [12].

The threat to user privacy is the most well-known implication of history detection. When coupled with fast testing, a non-trivial part of the user's visiting patterns can be extracted. This allows for testing of URLs containing location information such as zip codes entered on e.g., weather sites. In their real-world experiment Janc and Olejnik [1] noted that they could detect the US zip code for 9.2% of tested users.

3.2 Using History Detection to Learn When the Victim Reaches a Page

In our application of the history detection attack, we are not interested in the victim's browsing history but rather in what the victim is currently doing. In particular, we want to know when the victim's current browsing session reaches a target page (e.g., the landing page of a payment provider). To accomplish this, we can use the history detection attack to frequently poll the status of the target page to detect when it changes from unvisited to visited.

This use of history detection requires that the target page is marked as unvisited in the browser when the attack begins. Thus, the attack is easier to perform the quicker the browser forgets about visited links, in total contrast to privacy attacks which benefit from longer history retention. The CSS specification leaves it up to the implementor to select for how long a link will be treated as visited, and the major browsers have selected different periods. Internet Explorer and Safari stores history for 20 days, and Firefox for 90 days. Opera does not limit the time, but rather limits the number of stored entries to 1000. Chrome does not remove visited status, except when explicitly requested by the user.

Thus, our flow stealing attack is best suited to attacking pages which users trust, but which they visit rarely. We believe that payment providers fall in this category for many users.

3.3 Limitations of CSS History Detection

There are several ways in which the victim can be protected from the way we use CSS history detection in this attack. Firstly, Baron [13] has proposed a mechanism to close the CSS history detection security hole. The most basic mechanism involved is that the data returned by the JavaScript `getComputedStyle` method always return data as if the link had been unvisited. Furthermore, it prevents visited status of link from affecting which pictures are loaded, the layout of the page, and the time it takes to render a page to prevent a number of side-channel attacks. This proposal (or similar defenses) has been implemented in Firefox 4, Internet Explorer 9, as well as in browsers based on the WebKit rendering engine, such as Chrome and Safari. This means that in the latest versions mainstream browsers, with the exception of Opera, have closed the CSS history detection hole. Users may not always be able to upgrade to the latest version, for various reasons. For instance, Internet Explorer 9 is not supported on Windows XP, which will prevent many users from upgrading. Also, even if they could, some users simply refuse to upgrade their browsers. There is also a risk of regressions, or other history detection techniques being discovered. For instance, Weinberg *et al.* [14] reports that beta versions of Firefox 4 were vulnerable to CSS history detection through a debugging feature.

There are some techniques a user can deploy to protect herself, apart from switching or upgrading their browser. A user may choose to configure their browser not to store any browsing history. However, this comes at a usability price. Firefox users may also choose to install the SafeHistory extension [15] which essentially applies the same-origin policy to visited status on links, only treating a link as visited if it has been visited by a link from the current domain.

CSS history detection is not the only history detection attack that has been proposed against web browsers. In [16], Felten and Schneider discuss timing attacks to determine if cacheable elements of pages are present in the victim's cache. However, such attacks are not suitable to our history detection usage where we are not interested if the victim has historically visited a site, but rather in detecting the moment in time when a specific page is visited. Cache timing attacks cause the tested object to be cached, and thus the same object cannot be tested twice, making the attack unsuitable for repeated polling. Similarly, the history detection attacks building on user interaction of [14] cannot be used in our scenario. We remark that there is a companion extension to SafeHistory called SafeCache [15] to protect against cache timing attacks.

3.4 Network Based Timing

In the case of a network attacker who has access to the victim's network traffic, there are alternative timing mechanism for the cases when the CSS history detection timing mechanism does not work. As we assume that all the victim's browsing of `store.com` and `pay.com` is via https, the attacker is unable to directly observe how the victim interacts with the target domains. However, https does not protect against an attacker learning *that* the victim is visiting a certain domain, or the sizes of requests and responses.

There are several ways for the network attacker to learn when the victim visits `pay.com`. The first is by simply observing the victim's DNS traffic. When the attacker sees the victim's computer performing a DNS lookup for the IP address of `pay.com`, she can assume that the victim's browser is going to request something from that domain. However, if the victim frequently visits `pay.com`, she may already have the IP address cached in her browser, and thus not issue a DNS lookup when visiting the domain again. Another mechanism for the attacker is to look up the IP addresses of servers for `pay.com` and then trigger the attack when she sees the victim's computer connecting to one of those IP addresses on the https port.

Both these mechanisms may trigger the attack too early if other pages include elements from the `pay.com` domain, for instance if `store.com` includes a `pay.com` logo on their payment page. While this type of logo inclusion does occur, we remark that it is common practice for stores to host payment logos on their own servers, or for static content such as logos to be hosted on separate domains.

The attacker can learn if the store features `pay.com` logos served directly by `pay.com` servers by simply visiting the store herself before beginning the attack. If this is the case, she can perform a more thorough flow inspection and instead of just looking for a connection establishment to the right IP and port, analyze the number of bytes sent in each direction and the number of connections made to distinguish between the victim fetching a logo and visiting the landing page at the payment provider.

Communicating Back to Victim's Browser. When discussing the alternate timing mechanism available to the network attacker, we stated that the attacker "triggers the attack". However, the attacker is located as a man-in-the-middle to the victim's network traffic, and to trigger the attack, she must activate code running as JavaScript in a tab in the victim's browser. How is the trigger information communicated back to the victim's browser?

We remark that in our network attacker scenario, the malicious JavaScript has been inserted by the attacker on a web page not controlled by the attacker. Thus, the malicious JavaScript is prevented by the same-origin policy from directly communicating with the attacker-controlled server at `evil.com` via convenient mechanisms such as XMLHttpRequest.

However, as the attacker is mounting a man-in-the-middle attack on the victim's network traffic, this problem can be circumvented by the attacker intercepting and responding to requests to some specific path, regardless of what host the path is supposed to be located at. This allows the JavaScript inserted by the attacker to use XMLHttpRequest to periodically send a request to a path which the attacker will intercept. The attacker will not forward such requests, but instead respond with a boolean value indicating if the flow stealing redirect should be activated. There are several other options available, such as periodically loading images from `evil.com` and using the size of the returned images as a one-way communication channel to the JavaScript running in the victim's browser.

4 Impact and Feasibility of Flow Stealing

We have now described our proposed flow stealing attack, showing how it can be performed both by an attacker operating a web site as well as by a network attacker who can intercept the victim's network traffic. Apart from the conditions imposed by the type of attacker, the feasibility of the attack also depends on the victim's browser.

4.1 Browser Features

Our flow stealing attack combines two different vulnerabilities. Firstly, the attacker must be able to monitor when the victim is directed to `pay.com`. The primary mechanism for accomplishing this is by using a well-known history detection hole. Secondly, the attacker must at that point in time redirect the victim to `pay.com` with a new transaction ID.

While the redirection part is crucial for the flow stealing attack, the CSS history detection vulnerability is not needed for network attackers, as discussed in Section 3.4.

All mainstream browsers allow the redirection part of our attack. However, on the Opera browser, the attacker cannot simply redirect the victim's tab, but must instead close the tab and redirect another tab as discussed Section 2.1. This makes the attack more noticeable, as an alert victim may notice that a tab closed and become suspicious and abort the transaction.

To explore the feasibility of our attack, we have tested recent versions of browsers to see if the classic CSS-based history detection attack works, and what restriction they place on cross-domain window navigation through window handles. We present our results in Table 1. In the table, "CSS History Detection" indicates if the CSS history detection attack works. Redirection indicates if a window handle can always be redirected via JavaScript ("Permissive") or not ("Restricted"). The browsers were tested on Windows 7. We do not believe any of the results depend on the operating system the browser is run on.

4.2 Experiences with a Proof-of-Concept

In addition to testing the individual pieces of our flow stealing attack, we have also developed a proof-of-concept implementation of the attack as performed by a web-site hosting attacker. We consider the simplest version of attack which can

Table 1. Summary of browser's susceptibility to flow stealing

Browser	CSS History Detection	Window Navigation
Firefox 3.6.15	Yes	Permissive
Firefox 4.0.1	No	Permissive
IE 8.0.7600.16385	Yes	Permissive
IE 9.0.8112.16421	No	Permissive
Chrome 10.0.648.151	No	Permissive
Safari 5.0.4	No	Permissive
Opera 11.11	Yes	Restricted

be performed with a static html containing JavaScript for the attack using the CSS history detection timing mechanism. In our proof-of-concept, we replaced `store.com` with the donation page of a charity, to simplify testing (the donation page of the charity contains a link directly to the payment provider).

In our proof-of-concept, the transaction set up by the attacker has the attacker as the recipient instead of the charity. The recipient information is displayed by the payment provider, so an alert victim could notice that their flow had been hijacked by an attacker. To reduce the risk of this, an attacker could register names with the payment provider which are similar, or look identical [17], to the stores or charities that she will attack.

Another option for stealthy attacks is for the attacker to herself set up a purchase on `store.com`. She then records the transaction ID used by `store.com` when referring her to `pay.com`. By using this transaction ID in the attack, the attacker tricks the victim into paying for the her goods. In this scenario, the only indication to the victim that an attack is ongoing is if the information displayed on `pay.com` on what the purchase concerns differs from what she expected.

Guessing the Price. To make the attack convincing to the victim, the attacker needs to set up a transaction with the exact same cost that the victim expected. It seems likely that a large fraction of users would notice if the payment provider listed a different price compared to the store. We have not implemented any techniques for creating a transaction with the correct price in our proof-of-concept.

There are several ways for an attacker to guess the price. The easiest way is to attack subscription services or stores which sell a specific item or service for a fixed price, or a small number of different options so that the attacker can simply guess at the most common price. One such example is online streaming services such as Hulu, Napster, Netflix, and Spotify.

For stores with larger inventories, the attacker can use the CSS history detection attack to determine what items the victim has browsed and/or put in her shopping basket, depending on the URL scheme employed by the store. In the network attack scenario, traffic analysis on the number of requests and size of responses as the victim browses `store.com` may be used instead.

5 Proposed Counter-Measures

In this section, we discuss a simple server-side defense against CSS history detection that can be applied by payment providers for their landing page. We also discuss the information displayed to users of payment sites. We proceed to discuss the problem of frame navigation as it applies to top-level frames and propose a new policy based on pop-up blocking. Finally, we discuss why traditional CSRF defenses do not protect against flow stealing.

We note that our attack uses JavaScript to perform the redirection attack, so users can protect themselves against flow stealing by disabling JavaScript. However, this does remove functionality from a large number of web sites, so most users are unlikely to do so.

5.1 Closing the CSS History Detection Hole

We are happy that almost all of the mainstream browsers now have closed the CSS history detection hole. By closing this hole, attackers are denied the easiest route for performing flow stealing attacks. However, for various reasons, users are not always able to upgrade to the newest version of software in a timely manner. To protect users which are not able to upgrade, we propose that high-profile sites such as payment providers should consider implementing a server-side defense.

While landing pages of payment providers are external URLs in the nomenclature of [10], they could apply a protection technique by recommending sites linking to them to insert a random number in the link, which is simply ignored by the payment provider. As most payment providers want to help stores to very easily integrate payments, standard practice seems to be to provide some static HTML code to be included on the store's web site. Such code could include JavaScript code to generate a random number in the browser which is inserted into the URL of the landing page in a way that is ignored by the payment provider. This would prevent the link from being guessable, and thus detectable via CSS history detection.

5.2 Payment Provider Pages

The key place where the victim could detect that a flow stealing attack was ongoing is in the information shown by legitimate payment providers. It is important to provide as clear feedback as possible to end users of payment sites on who the recipient of the payment is, and what the payment concerns. For instance, the payment provider could indicate if the recipient is a company, a charity, or an individual.

In a typical payment provider integration, the information on the purchase depends on what information is sent from the store to the payment provider when setting up the transaction. Thus, stores can assist in making flow stealing attacks easier to detect by including more information. For instance, this may include the purchaser's username on the store, or the shipping address.

5.3 Limiting Window Manipulation via Window Handles

There is a difference in policy between browsers on what limits are applied to how a page can change the URL of another window to which it has a JavaScript window handle. Opera restricts such navigation based on the current location of the frame, and protects frames navigated to https sites from being navigated from another window. In Chrome, Firefox, Internet Explorer, and Safari, the opener is allowed to freely navigate an opened window, and in some of them, also other windows apart from the opener.

Frame navigation has previously been showed as being dangerously permissive in the context of embedded frames and iframes by Barth *et al.* [18], which influenced browser developers to implement a more restrictive policy. They note that top-level frames are often exempt from the browser's frame navigation policy, and that top-level frames are less vulnerable as their URL is shown in the location bar.

While it is true that top-level frames are less vulnerable than embedded frames, there is still a danger in permissive policies for navigation of top-level frames. We cannot trust a user to, at every point in time in their browsing session, validate that the location in the location bar is correct. For instance, we cannot expect users to note if their location is changed to a similarly looking URL, or identical looking URL via a homograph attack [17]. Neither can we expect users to notice if opaque identifiers in sessions are replaced.

The fact that different policies have been implemented in different browsers indicates that it is unlikely that a large number of pages rely on the most permissive policies for their functionality. The only policy restricting our flow stealing attack is Opera's. However, as we discussed in Section 2.1, Opera's policy is still sufficiently permissive that it allows flow stealing attacks by closing the window and redirecting the window running the attacking JavaScript. Thus, we argue that a replacement policy should not only restrict navigation, but rather all actions affecting the window, including closing it and resizing it (an attacker could emulate closing by resizing to a very small size).

We are not aware of any important applications where a window w_1 needs to modify another window w_2 where the modification is not prompted by user interaction with window w_1. For what types of user interaction would a user expect w_1 to modify the state of another window w_2? We argue that in any user interaction that would not allow w_1 to open a new window, w_1 should not be allowed to modify the state of another window either. In mainstream browsers today, the situations in which w_1 is allowed to open a window is restricted by a *pop-up blocker*. We believe a user would not expect w_1 to modify any windows unrelated to it, a policy already implemented in the Firefox browser which limits navigation to the *opener* window.

As far as we know, each mainstream browser implements its own algorithm for pop-up blocking, a feature enabled by default. Thus, most web sites have been adapted to page manipulations allowed by the pop-up blocking policies of browsers. We are not aware of any detailed descriptions of pop-up blocking algorithms, but they appear to work satisfactorily in major browsers. According to Chen [19], browser developers are hesitant to specify the exact policies used as that may prevent them from modifying the policy later, if a loophole is discovered.

Thus, we propose the following policy for controlling a window via a window handle:

Policy 1 (Window navigation, Proposed). *A window w_1 can modify (e.g., navigate, close, or resize) another window w_2 only if it is the opener of w_2, and the pop-up blocker policy currently allows w_1 to open a window.*

Furthermore, we believe that rights to a window should be relinquished entirely if the user manually navigates (e.g., by entering a new URL in the address bar) the tab. Currently, this does not appear to affect the rights granted to the JavaScript holding the handle to the window, but we believe it would match the user's expectation more closely that the opener retains no special privileges if the user navigates the window.

5.4 Traditional CSRF Defenses Do Not Prevent Flow Stealing

Our flow stealing attack has some similarities to traditional CSRF attacks, so one may wonder if traditional CSRF defenses protect against flow stealing as well. Sadly, the answer is no, and new techniques are needed to protect against flow stealing. We briefly mention the two major CSRF defenses from the literature and discuss why they do not protect against flow stealing. At the core of the problem is that in flow stealing the victim's browser is redirected at a point in time where the control is passed between sites operated by two different entities. Thus, the hijacked request is legitimately a cross-site request.

The most common class of CSRF defense consists of a secret validation token that must be sent along with all state-modifying requests, and that is matched to the user's session. There are several different implementations of this technique, and there are some subtleties in implementing the protection correctly, cf. [2]. Such tokens are designed to protect flows internally on web-sites, and are not immediately applicable to cross-site flows.

A second technique is based on inspecting either the `Referer` or the `Origin` HTTP header. Typically, this is described as only allowing requests if the host in the header matches the current host, but the policy could easily be extended to allowing external requests from some specific set of domains. As an example, a payment provider may require that users making payments to `store.com` come to `pay.com` with an Origin header set to `store.com`. However, this does not prevent flow stealing, as the attacker can register as a merchant with the payment provider and redirect via the correct domain for that merchant. The attacker can also redirect the victim to a fake payment site instead of the legitimate site, thus bypassing any controls that could be implemented by a payment provider.

6 Conclusion and Future Work

In conclusion, we have demonstrated an attack on current web browser implementations. The attack uses the CSS history detection attack, which has been publicly documented for about a decade, to time a redirection attack. By redirecting the tab the victim is using at a point where the victim legitimately expects to perform some security critical action, the victim can be tricked into doing something more sensitive than what can be achieved by e.g. phishing. We hope that our attack further aids in demonstrating the importance of closing the CSS history detection hole, and future holes with similar impact.

As future work, we propose developing a proof-of-concept version of the network attack as well. The purpose of such a proof-of-concept prototype would be to show that while closing the CSS history detection hole is an important step, it is also important to further limit JavaScript cross-site frame navigation, as well as deploying https as a default for a larger fraction of Internet sites. We note that other proof-of-concept attacks such as Firesheep [20] have been able to quickly raise public awareness of security issues and caused deployment improvements at large sites.

Acknowledgments. I would like to thank Emil Hesslow for great discussions, development of a proof-of-concept, and his JavaScript expertise.

References

1. Janc, A., Olejnik, L.: Web Browser History Detection as a Real-World Privacy Threat. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 215–231. Springer, Heidelberg (2010)
2. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security, pp. 75–88. ACM, New York (2008)
3. Raskin, A.: Tabnabbing: A new type of phishing attack, http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/
4. Phung, P.H., Sands, D., Chudnov, A.: Lightweight self-protecting javascript. In: Li, W., Susilo, W., Tupakula, U.K., Safavi-Naini, R., Varadharajan, V. (eds.) ASIACCS, pp. 47–60. ACM, New York (2009)
5. Ruderman, J.: Bug 57351 - css on a:visited can load an image and/or reveal if visitor been to a site, https://bugzilla.mozilla.org/show_bug.cgi?id=57351
6. W3C: Cascading style sheets level 2 revision 1 (CSS 2.1) specification, http://www.w3.org/TR/CSS2/
7. Anonymous: Did you watch porn (2010), http://www.didyouwatchporn.com/
8. Janc, A., Olejnik, L.: What the internet knows about you (2010), http://www.wtikay.com/
9. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users. In: IEEE Symposium on Security and Privacy, pp. 223–238. IEEE Computer Society, Los Alamitos (2010)
10. Jakobsson, M., Stamm, S.: Invasive browser sniffing and countermeasures. In: Carr, L., Roure, D.D., Iyengar, A., Goble, C.A., Dahlin, M. (eds.) WWW, pp. 523–532. ACM, New York (2006)

11. Kennedy, N.: Sniff browser history for improved user experience (2008), http://www.niallkennedy.com/blog/2008/02/browser-history-sniff.html
12. Jakobsson, M., Juels, A., Ratkiewicz, J.: Remote harm-diagnostics, http://www.ravenwhite.com/files/rhd.pdf
13. Baron, L.D.: Preventing attacks on a user's history through CSS :visited selectors, http://dbaron.org/mozilla/visited-privacy
14. Weinberg, Z., Chen, E.Y., Jayaraman, P.R., Jackson, C.: I still know what you visited last summer. In: IEEE Symposium on Security and Privacy (2011)
15. Jackson, C., Barth, A., Bortz, A., Shao, W., Boneh, D.: Protecting browsers from DNS rebinding attacks. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 421–431. ACM, New York (2007)
16. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: ACM Conference on Computer and Communications Security, pp. 25–32 (2000)
17. Holgers, T., Watson, D.E., Gribble, S.D.: Cutting through the confusion: A measurement study of homograph attacks. In: USENIX Annual Technical Conference, General Track, pp. 261–266. USENIX (2006)
18. Barth, A., Jackson, C., Mitchell, J.C.: Securing frame communication in browsers. Commun. ACM 52, 83–91 (2009)
19. Chen, R.: The internet explorer pop-up blocker follows guidelines, not rules, http://blogs.msdn.com/b/oldnewthing/archive/2007/08/31/4656351.aspx
20. Butler, E.: Firesheep, http://codebutler.com/firesheep

Reclaiming the Blogosphere, TalkBack: A Secure LinkBack Protocol for Weblogs

Elie Bursztein[1], Baptiste Gourdin[1,2], and John C. Mitchell[1]

[1] Stanford University
[2] LSV,ENS-Cachan
{elie,bgourdin,jcm}@cs.stanford.edu

Abstract. A LinkBack is a mechanism for bloggers to obtain automatic notifications when other bloggers link to their posts. LinkBacks are an important pillar of the blogosphere because they allows blog posts to cross-reference each other. Over the last few years, spammers have consistently tried to abuse LinkBack mechanisms as they provide an automated way to inject spam into blogs. A recent study shows that a single blog may receive tens of thousands of spam LinkBack notifications per day. Therefore, there is a great need to develop defenses to protect the blogosphere from spammer abuses. To address this issue, we introduce TalkBack, a secure LinkBack mechanism. While previous methods attempt to detecting LinkBack spam using content analysis, TalkBack uses distributed authentication and rate limiting to prevents spammers from posting LinkBack notifications.

1 Introduction

Since their emergence over a decade ago [15], blogs have become a major form of communication, with more than 184 million blogs read by more than 346 million readers in 2008 [14]. Along with widespread legitimate use for communicating information and opinions, blogs have naturally attracted two forms of spam (unwanted postings): *comment spam* and *LinkBack spam*. Comment spam is an extension of traditional email spam and can be mitigated by requiring users to authenticate before commenting, or to solve CAPTCHAs [4]. On the other hand, *LinkBack spam* is specific to blogs. LinkBack mechanisms [23] are used to automatically insert cross-references between blogs. A new blog post citing an older one on a different blog can send a **LinkBack notification** to insert a link in the older post **automatically**. LinkBack notifications are an intrinsic part of the blogosphere, and a key ingredient used in blog ranking [21]. Because LinkBack notifications are automated, CAPTCHAs [22] and registration requirements cannot be used to defend against spam. So far, very little research has been conducted on LinkBack spam specifically, and only general anti-spam techniques based on content analysis are currently used.

A recent study [5] reveals that LinkBack spam is a huge issue as a single blog may receive tens of thousands of spam LinkBack notifications per day. Further, it found that LinkBack spammers skillfully use anti-spam-analysis techniques

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 133–149, 2011.

that foil content analysis by inserting random words to escape Bayesian filters. Because LinkBacks are concise, with very little content to analyze, it is difficult to filter LinkBack content by applying content-based filtering techniques.

To combat notification spam, in this paper, we introduce *TalkBack* a secure LinkBack mechanism based on public-key cryptography. TalkbBack departs from the previous approaches as it tackles the LinkBack spam problem at its root: instead of detecting spam via content analysis, TalkBack is designed to prevent spammers from posting LinkBack notifications. TalkBack creates two lines of defense. The first one is a lightweight PKI [1] (Public Key Infrastructure) that ensures the identity of blogs by using public-key cryptography, and makes it hard for spammers to register a fake blog. As a second line of defense, TalkBack enforces a global rate limiting system that ensures that with a single blog identity, a spammer cannot massively spam any collection of participating blogs. An additional benefit of adopting TalkBack is that bloggers can leverage the TalkBack PKI infrastructure to build secure and reliable whitelists and blacklists of blogs. We believe that by combining TalkBack with the other anti-spam mechanisms based on content analysis already in place, the blogosphere will have efficient defense in depth against the assault of spammers.

We have developed, tested and evaluated all the components needed for blog sites and bloggers to adopt TalkBack quickly, with additional attention directed toward Wordpress, the leading blog platform. To make TalkBack readily available to bloggers, we have developed a Wordpress plugin that is available from the standard Wordpress plugin directory `http://ly.tl/tb`. A full implementation of the TalkBack protocol is also available as an open-souce PHP library, maintained and freely available from google code (`http://ly.tl/tbs`), and an operating talkback authority is implemented and maintained at `https://talkback.stanford.edu`.

2 Background

In this section we summarize the way LinkBack mechanisms work, how they are abused by spammers, and some reasons why spammers do so.

The term **Blog** is a contraction of the term **web-log**. Blogs can be used for any topic but are often used by **bloggers** to share and exchange information and personal opinions on subjects that range from personal life to video games, politics, and wine. The **Blogosphere** is a collective term referring to all blogs and their interconnections, coined in 1999 by Brad L. Graham as a joke [6]. The blogosphere can be viewed as a graph, with blogs as nodes and edges corresponding to **LinkBacks** between blogs.

LinkBack mechanisms are used to produce a link from one blog post to another that references it. For instance if *Blog B* discussing French wine cites a post on *Blog A* about Bordeaux wine, a LinkBack mechanism allows *Blog B* to notify *Blog A* about this citation. As a result of this LinkBack notification, *Blog A* may then display a link back to *Blog B* (hence the name LinkBack). There are three main LinkBack mechanisms, which differ in their implementation details, that are currently used. These three mechanism are *TrackBack* [3], *PingBack* [12] and *RefBack* [23]. Note that for the largest blog platform *Blogger*, Google provides a specific LinkBack implementation based on Google Infrastructure. These LinkBack mechanisms were designed to help blog readers navigate from one post to other relevant posts. Every LinkBack mechanism is implemented into two parts: the **auto-discovery mechanism** and the **notification page**:

The **auto-discovery** mechanism embeds a `<link>` tag or a small Resource Description Framework (RDF) fragment in each blog post to tell other blogs to which page they should submit their LinkBack notification. RDF is a family of World Wide Web Consortium (W3C) specifications designed as a metadata data model, that are used as a general method for conceptual description or modeling of information in web resources.

The **notification page** is the web page dedicated to collecting LinkBacks notification and processing them. For example, the TrackBack notification is an HTTP POST request sent to the notification page which contains four post values: the *post title*, its *URL*, an *excerpt*, and the *blog name*. An example of a TrackBack [3] post request is:

```
POST http://www.example.com/TrackBack/5
Content-Type: application/x-www-form-urlencoded
title=Foo&url=http://www.bar.com/&
excerpt=My+Excerpt&blog_name=Foo
```

Blog Spam. Because LinkBack notifications provide an automated way to insert links into other bloggers' blogs, it is not surprising that malicious users began using it soon after it appeared. There are two main motivations for abusing LinkBack mechanisms: search engine optimization, and spam to lure users to malicious sites. One of the major spam-blocker providers, Akismet [18], reported blocking around 15 million LinkBack spams a day in April 2009, in comparison with 1.8 million legitimate LinkBacks. Hence it seems that the percentage of blog spam is slightly lower (90%) than the 98% spam reported for email [8]. However blog spam is more pernicious because it is asymmetric: one spam LinkBack notification might lure thousands of blog readers to a malicious site, whereas delivered email spam usually reaches at most one user.

3 Threat Analysis

In this section we introduce the attacker and threat model that TalkBack needs to address. These models are based on the conclusions of our blog spam study [5]. For this longitudinal study of TrackBack spam, 10 million samples from a massive spam campaign over a one-year period where collected and analyzed. Based on the analysis of blog spammers' behavior and resources, we believe that TalkBack needs to block the efforts of a sophisticated adversary that will not be fooled by simple defenses. In particular, we believe that in order to be effective, TalkBack must be able to thwart an attacker that is ressourceful, knowledgable and adaptative and accordingly should assume an adversary with perfect knowledge of the protocol and a lot of IPs, Domains and CPU power at his disposal.

While important to the operation of blogs, our threat model does not address system, services or web attacks because they cannot be addressed directly by a LinkBack mechanism. Therefore, TalkBack focuses on LinkBack threats, which are:

- **Blog Spoofing:** The attacker should not be able to spoof a blog identity. In particular, the attacker should not be able to impersonate a real blog because otherwise it is not possible to use whitelisting or blacklisting mechanisms. TalkBack must ensure the identity of a notification sender.
- **Cried Wolf attack:** The attacker should not be able to report legitimate notifications as a spam otherwise he will be able to prevent legitimate users to use TalkBack by reporting them as spammer. To address this attack when a notification is reported as spam, TalkBack must verify the sender and receiver identities.
- **LinkBack modification:** The attacker should not be able to mount a person-in-the-middle attack that alters the content of a LinkBack in order to spam a blog or abusively report a legitimate user. Accordingly, TalkBack needs to ensure LinkBack integrity.
- **LinkBack replay:** It should not be possible to resend or replay a notification. This is central to enforcing a rate limiting system.
- **Accumulation attack:** It should not be possible for an attacker to accumulate posting authorizations, over an extended period of time, in order to use them all at once to perform a massive “Blitzkrieg” spam.
- **Spamming in breadth:** A Spammer can send a few or even a single spam to many blogs, in a way that each blog will only see a negligible amount of the spam and cannot rate limit it. Therefore the rate limiting and spam detection systems need to be global.
- **Spamming in depth:** Spammer can use many IP addresses and URLs to send spam, which makes blacklisting very hard. Therefore, TalkBack must restrict a sender to a single blog identity so that we can leverage identity-based blacklisting and whitelisting.

4 Overview

In this section, we give an overview of how TalkBack works. In particular, we describe how TalkBack addresses the threat of spammer notifications, the key steps required to post a TalkBack notification, how multiple authorities are handled and how TalkBack compares to the other LinkBack mechanisms.

The key idea of TalkBack is to prevent spammers from posting LinkBacks, or make it prohibitively costly to do so. As mentioned previously, based on our blog spam study [5], we do not consider content analysis effective when used alone. In a nutshell, TalkBack can be viewed as a lightweight PKI (*Public Key Infrastructure*) [1] that authenticates bloggers, blogs and LinkBacks. The blogger and blog identities are verified by a registration mechanism (Sec. 5) that ties a

Table 1. LinkBack mechanisms comparaison

	RefBack	**PingBack**	**TrackBack**	**TalkBack**
Trigger Mechanism	Visit from the sender site	Code executed at posting time	Code executed at posting time	Code executed at posting time
Notification via	HTTP referer	XML-RPC call	HTTP POST	HTTP POST
Information sent	none	- S post URL - R post URL	- S post URL - S site name - S post title - S post excerpt	- S post URL - S site name - S post title - S post excerpt - Seed Token - S Public key - R Public key - Signature
Auto-discovery mechanism	none	LINK Tag	Tag in the body	LINK Tag
S Authenticity	-	-	-	✓
R Authenticity	-	-	-	✓
Integrity	-	-	-	✓
Confidentiality	-	-	-	✓

Blogger and his blog to a public-key. This public key is used in every notification to ensure that a spammer can't spoofs a blog identity, modifies its content, or replays it.

The blog registration process uses various security checks to ensure that the blogger is the blog's real owner. It also implements security mechanisms to make sure that it is not possible to register a blog automatically. As a defense in depth mechanism, TalkBack uses a rate limiting system that ensures that even if a spammer is able to successfully register a blog, the amount of spam he can send with it is limited. To ensure that the authority is not a contention point, TalkBack was designed to work with multiples authorities.

TalkBack is also designed to accommodate any blogger' additional privacy needs with a *confidentiality mode* that encrypts notification content and makes sure that no information is disclosed to anyone (including the authority) except the receiver.

Posting a LinkBack. Once blogs are successfully registered with an authority, posting a LinkBack is achieved in at most four steps (Diagram 1). In the TalkBack protocol there are three participants (three if the sender and the receiver refer to the same authority) :

1. A: The **sender authority**, which is used to authenticate the sender and enforce rate limiting.
2. S : The **sender**, which is the blog that wants to send the LinkBack notification
3. R: The **receiver**, which is the blog that receives and processes the LinkBack notification.

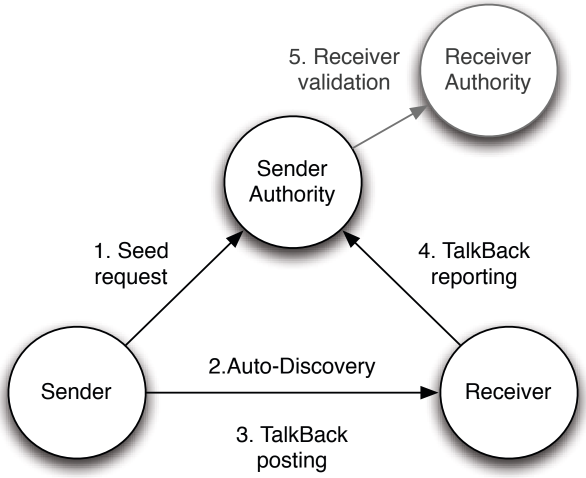


Fig. 1. TalkBack protocol overview

4. A': The **receiver authority**, which might be different from the sender authority and is used to authenticate the receiver.

The five steps used to send a LinkBack are depicted in diagram 1. Here is what happens during these steps:

1. The sender (S) requests a seed from the authority. This seed is used to prevent accumulation attacks, replay attacks, and to enforce rate limiting (Sec 7).
2. The sender (S) crawls the receiver (R) blog as usual, discovers the notification URL (Sec 6) and in addition fetches the receiver (R) public key used to authenticate the receiver in the LinkBack and to encrypt data (Sec 7).
3. The sender (S) uses the seed fetched at step 1 along with the notification URL and public key fetched at step 2 to build and send the secure LinkBack to the receiver (R).
4. The receiver (R) performs security verification on the received LinkBack and eventually forwards it to the authority A to ensure that the LinkBack and sender S are still valid.
5. If the receiver (R) authority (A') is different from the sender (S) authority (A), then the sender authority contacts the receiver authority to fetch and validate the receiver (R) identity. To improve performance, the receiver public key is cached. Note that there is a RESTful API in place that allows authorities to communicate. For clarity and because communication between authorities is straightforward, we assume for the rest of the paper that authorities (A) and (A') are the same.

As we will see in the sections 6 and 8, TalkBack provides numerous additional features, such as caching and whitelisting, designed to reduce the workload. Accordingly, in some cases posting a LinkBack only requires performing the Step 3.

Comparison with Other Mechanisms. As shown in the table 1, TalkBack is the only LinkBack mechanism that provides security features. None of the other mechanisms provide a way to ensure sender and receiver authenticity, or LinkBack integrity and confidentiality. To ensure that TalkBack will be a widely adopted standard, it has been design to be robust, lightweight, easy to implement and compatible with web standards. This is why we choose, like PingBack, to use the standard HTML tag `<link>` to embedded the discovery mechanism and the blog public key (Sec 6). This choice ensures that adding this information does not interfere with the page validity and makes the required information easily retrievable by a crawler. Similarly we choose to use, like TrackBack, the HTTP POST method to post notifications because the success of the RESTful API support the fact that using the HTTP POST method is the easiest way to send data from one web service to another. Finally, one can observe that TalkBack is backwards-compatible with the TrackBack mechanism which is currently the most popular LinkBack method. This is meant to ensure that the transition to the TalkBack method can be done smoothly without breaking existing systems.

5 Blog Registration

In this section, we describe the registration process that a blogger needs to complete before she is allowed to send or receive TalkBack notifications. The goal of this registration is two-fold, it aims at both linking the user identity to a blog URL and at linking the blog URL to a public key. We also describe the update process that the blogger can use to tell the authority that the blog's public key has changed.

The registration process is accomplished in four steps:

1. **Authority Selection:** First the user has to choose from a list which authority he wish to enroll with. Currently the only available option is to use our authority (`http://...`) but both our open-source library and Wordpress plugin are already able to handle multiples authorities. Adding an authority is as simple as adding its public key and URL in the plugin configuration file and push the update to the user via the Wordpress update system.
2. **Turing Test:** The second part of the registration aims at preventing automated registration.To do so, we ask the user to solve a CAPTCHA [22]. During this phase we also ask for a name, blog URL, a password, and an email address. Note that our Wordpress plugin reuses the information supplied by the user when setting up Wordpress. Accordingly the registration process is almost completely automated the user should only have to verify the information and supply a password. At the end of this stage the authority asks other authorities is they have already the blog enrolled. If it is the

case, then the registration process is aborted and the user is redirected to the authority she is already enrolled with.

3. **Identity Verification:** The third part of the process involves verifying the user by sending an email to the supplied email address. The user is then required to click on an URL that embeds a secret token in its parameters. Because this is a crude and only partly effective identity verification, we give an incentive to the user to provide a stronger form of identity verification, as explained below.
4. **Blog Ownership Verification:** The last part of the registration process ensures that the user registers a blog that he owns. This step is very close to the blog reclaiming process used by Technorati [21] and various analytic tools such as Google Analytics: we ask the user to add a random string to his blog index page headers. The random string is embedded into the header by adding the following meta tag:

   ```
   <meta name=TalkBack-Id"  content="random-string" />
   ```

   Once the users says he is ready, we crawl the blog URL, verify that the string is present and fetch the public key and link it to the user identity. Our Wordpress plugin takes care of generating the blog private/public key pair and add the necessary header. The public key fetched at this step is the one that will be used to identify the blog and verify TalkBack notification signature validity.

Note here that the authority never sees the user's private key. As a matter of fact this key is intentionally not part of the generation process for two reasons: First, this limits the incentive to compromise the authority database: even with its content, the attacker will not be able to forge notifications.

Secondly, it limits the trust that the user needs to have in the authority – the authority is not able to post notifications on the user's behalf because they must be signed by her private key.

TalkBack's rate limiting system is reputation-based: the better the user reputation, the more TalkBack notifications she is allowed to send per day. The user's reputation can increases in two ways: first, every time the user sends a talkback notification and this notification is not reported by the receiver as spam, the user's reputation increases toward a maximum. Secondly, the user can increase her reputation by providing stronger proof of her identity. For instance, we envision that the user will be able to link her Facebook or Twitter account to her blog account to have a higher limit[1]. In case of spam reports the reputation of the user decreases until the account is locked. The rate limiting enforcement and the lockout system are designed to mitigate the harm that a spammer can do by stealing the user's private key or registering a fake blog.

Additionally, recall here that taking over the user's account does not allow the attacker to post TalkBack notifications on user's behalf because the authority does not know the user's private key. Similarly attackers can't perform a "*cry*

[1] This feature is not yet implemented.

wolf attack" and abusively report legitimate notifications as spam, because a blogger can only report notifications actually received and proves her identity by signing the report with her private key.

6 Auto-Discovery

In this section we show how the auto-discovery and notification mechanism for TalkBack is implemented.

Similarly to the PingBack method, the URL for the notification mechanism is embedded in each blog post using a `<link>` tag. For example, if the blog's URL is *"myblog.com"* then in each post, there will be an auto-discovery link which look like this:

```
<link  rel="alternate" type="talkback-notification/plain|encrypted|both"
href="http://myblog.com/notify.htm?id=%postid" />
```

The *rel* parameter is used to tell the sender which kind of TalkBacks the blog accepts. The three acceptable policies are:

1. **Plain:** This policy indicates that the blog only accepts TalkBack notifications that are signed for authenticity and integrity but not encrypted (confidentiality). Confidentiality requires extra computation and is not useful if the blog is public because the content of the TalkBack will ultimately be displayed on a public blog post.
2. **Encrypted:** This policy indicates that the blog only accepts TalkBack notifications that are signed for authenticity and integrity and encrypted to ensure confidentiality. This is useful when one wants to preserve notification confidentiality. In this case even the authority (Sec 7) is unable to read it!
3. **Both:** This policy indicates that the blog is accepting both plain and encrypted TalkBack notifications.

This URL contains a variable: **%postid** which is an internal ID used by the blog to know to which post the notification is referring to. To be able to send a notification the sender also needs the receiver's public key, if he doesn't have it, he can fetch it by looking to the link tag:

```
<link rel="alternate" type="talkback-crypto/publicList-hashList"
href="http://myblog.com/talkback-key">
```

The *rel* parameter is used to indicate to the sender which cryptographic algorithms the blog supports. There are two lists of algorithms separated by a **-** (dash):

1. **PublicList:** This is the list used to tell which public key cryptographic algorithms the blog supports. If the blog supports multiple algorithms, they are separated by a **,** (comma). Currently TalkBack uses RSA, but we hope in the future to use elliptic curves (EC) when they become more widely available as this will decrease the size of the public keys and thus reduce the network load.

2. **HashList:** The hash list specifies which hash functions can be used. Currently, TalkBack uses SHA1. In the future TalkBack will support the upcoming SHA3 standard.

Note that, as explained in the threat model section, we assume that the attacker will be able to fetch any public information so we don't even try to prevent him from doing so. Instead we rely on the Kerckhoffs' principle and base the security of TalkBack on the security of the keys used.

7 Protocol

In this section, we describe how the TalkBack core protocol works step by step (Diag 1). To do so, we use a formal representation that abstracts away some implementation details for the purpose of clarity. Note that, like every other LinkBack protocol, TalkBack is fully automated and does not require any blogger intervention. The blog engine takes care of sending notifications automatically when a blogger writes a blog post. Our Wordpress plugin hook into the Wordpress notification system to do so.

Notation. We take the following conventions: Ar denotes the receiver TalkBack authority, As the sender TalkBack authority, S the sender of the TalkBack notification, and R the receiver of the TalkBack notification.

For message direction, we write $R \rightarrow A$ to say that the TalkBack receiver (R) sends a message to the TalkBack authority (A). For encryption, we use $\{n\}_A$ to denote that the nonce n is encrypted with the public key of A. For the sake of clarity, we take the convention that the signature is applied to all the nonces located on the left-hand of the signature symbols. For example $n1, n2, n3, Sig_A$ is equivalent to $n1, n2, n3, Sig(n1, n2, n3)_A$. Finally the letter on the left is used to indicate whether the message is used for the plain version $\mathcal{P}$, the encrypted version $\mathcal{E}$, or both versions $\mathcal{B}$ of TalkBack.

Step 1: Seed Request. In the first step the sender S requests a seed from the authority As that will be used to generate the tokens used in the next step. The seed request goes as follows:

$\mathcal{B}$: $S \rightarrow As\ \{ts,\ \mathsf{H}(TB)\}_{As},\ Pk_S,\ Sig_S$
$\mathcal{B}$: $As \leftarrow S\ \{R_s, R_n, R_t\}_S, Sig_{As}$

First, S sends a seed request that contains its public key Pk_S used to be identified and the *hash* ($\mathsf{H}(TB)$) of the four TalkBack content variables which are *title, excerpt, URL, blog_name*. The sender public key (Pk_S) is used to identify the sender blog and the timestamp to introduce randomness. In return, the sender (S) receives from our authority As a *random seed (R_s)*, the *number of TalkBack notifications* (R_n) he is allowed to use this seed for and the seed expiration time (R_t). Note here that the number of notifications allowed for a given seed depends on the user reputation and the version used. The rationale behind this decision is that if the user chooses to use the secure version to ensure confidentiality, it is unlikely that he will notify a lot of blogs.

The seed is used to enforce the rate limiting system: since the blogger can only use the seed to generate a limited number of tokens, he will not be able to spam the blogosphere massively. Using a seed decreases the network load as only one request/response to the authority is sufficient to acquire all the tokens needed to post the notification for a given post. In case the user exceeds his quota of notifications, which should not happen often for an honest user as it is relatively large, he has the ability to visit the authority to reset his quota by solving a CAPTCHA. Of course, we rate limit the number of times the user can reset his quota. The security rationale behind allowing users to reset their quotas is that if we make it as expensive for the spammer to reset the quota than creating a blog, he will have no incentive to use this functionality.

Step 2: Discovery and Token Generation. Once the sender has acquired the *random seed (R_s)* and the *number of TalkBack notifications* (R_n) he is allowed to send, he crawls all the links embedded in his blog post and discover those who point to blogs that use TalkBack thanks to the discovery mechanism. For each notification that needs to be sent, the following standard S/Key generation algorithm [7] is used to compute the required unique token from the seed.

Step 3–4: Posting Notifications. Now that the sender has the list of URL he needs to send notifications to and the associated tokens, he will start posting these notifications. Each TalkBack notification is posted using the following protocol:

$\mathcal{P}$: $S \rightarrow R$ $\mathsf{H}(TB)$, T_x, ts, Pk_{As}, Pk_S, Pk_R, Sig_S TB
$\mathcal{E}$: $S \rightarrow R$ $\{\mathsf{H}(TB)\}_{As}$, T_x, ts, Pk_{As}, Pk_S, Pk_R, Sig_S, $\{TB\}_R$

$\mathcal{P}$: $R \rightarrow Ar$ TB, T_x, ts, Pk_{As}, Pk_S, Pk_R, Sig_S, Sig_R
$\mathcal{E}$: $R \rightarrow Ar$ $\{\mathsf{H}(TB)\}_{As}$, T_x, ts, Pk_{As}, Pk_S, Pk_R, Sig_S, Sig_R

$\mathcal{B}$: $R \leftarrow Ar$ $D, ts2, Sig_R, Sig_{Ar}$

In the plain ($\mathcal{P}$) version, the sender (S) does a HTTP POST to the receiver (R) notification page discovered in the previous step. The sender sends the four content variables (TB) which are the *title, excerpt, URL and the blog_name*, the unique token T_x generated during the previous step, *ts* the notification creation time, the sender public key Pk_S, and the receiver public key Pk_R. Everything is signed with the sender private key (Sig_S). Sending the sender authority public key (Pk_A) is needed to support multiple authorities: the receiver uses it to know which authority to contact to validate the TalkBack. The encrypted ($\mathcal{E}$) version sends the four content variables encrypted $\{TB\}_R$ with the receiver's public key R and the encrypted hash of the four content variables $\{\mathsf{H}(TB)\}_A$ with the sender authority's public key. The receiver can verify the talkback content integrity by hashing and encrypting it with the authority public key and finally compare these two encrypted hashes. This hash is needed to ensure that the attacker does not tamper with the encrypted content and to prove to the authority that the encrypted content is really what the sender sent. Please note that we intentionally did not include the encrypted variables as part of the signature to ensure that the receiver does not have to forward them to the authority while validating the notification.This is not an integrity issue because the hash of encrypted variables is signed and acts therefore as the signature itself. Finally it is worthwhile to note that it is necessary to include the receiver's public key (Pk_R) in the notification and as part of the signature because otherwise the spammer will be able to perform a "cry wolf attack" and report an arbitrary sender even if this one never sent him a notification directly.

In the second step the receiver (R) verifies that the TalkBack is valid by verifying that that receiver public-key (Pk_R) is his own key and that the verifies the sender signature is valid. If the TalkBack is valid, he signs it and forwards it to the sender authority for validation.

Finally, in both ($\mathcal{B}$) cases the authority A answers whether the notification is valid or not (D) and the reason for rejection if needed. To decide if a notification is valid the authority verifies the following elements:

- **Signatures:** The authority verifies that the signatures match the public keys present in the notification and the notification integrity with them.
- **Token:** The authority verifies three things about the random token T_x: First, that it is indeed associated with the sender's public key, secondly that it wasn't used before, and finally, that it is not expired.
- **Sender Authenticity:** The authority is able to verify the sender's authenticity by correlating two things : first that the signature matches the public key present in the TalkBack, which means that the sender knows the private key, and secondly that the token matches the seed request made by this sender. No one is able to change the sender's public key embedded into the notification because it will invalidate the sender's signature.
- **Receiver Authenticity:** The receiver's authenticity is ensured by the fact the the signature matches the public key present in the notification. The receiver's public key was put in the notification by the sender and therefore can't be swapped with another one without invalidating the sender's signature. Therefore, if the sender's and the receiver's signatures are valid it is a valid TalkBack.

8 Optimizations

In this section, we present some optimizations that have be implemented to improve TalkBack's performance. These optimizations aim at reducing blogs and authority workload.

The first optimization that is used to improve TalkBack performance is to use notification batch processing. Upon receiving a notification, the receiving blog instead of forwarding it immediately to the authority for validation, puts it in a queue and waits until the queue is full or a maximum age is reached to send to the authority. Queuing decreases both the workload and the network load as it only uses a single pipelined connection between the blog and our authority. Our Wordpress plugin allows bloggers to customize the queuing behavior in terms of queue size and maximum time before processing to accommodate their needs.

As a second optimization, a blogger can also leverage the TalkBack PKI to build a whitelist of blogs that he trusts and for which he will accept TalkBack notifications without validating them with the authority. Using sender public-keys to build a whitelist is more simple and robust than using a password or a "secret URL" as it does not requires the use of a shared secret that needs to be exchanged and can be leaked. Then, on a regular basis the blog can check the authority revocation list to see which keys in the whitelist have been revoked and for what reasons. Whitelisting decreases the blog and authority workload. The main downside of whitelisting is that once a blog is whitelisted, the blogger will be able to flood this particular blog with notifications as the rate limiting is enforced by the authority, so the blogger should only whitelist trusted blogs. Our Wordpress plugin already support this form of whitelisting.

9 Performance

To evaluate the efficiency of the TalkBack approach and facilitate adoption, we have implemented the protocol in an open-source PHP library (`http://ly.tl/tbs`), are maintaining an authority `https://talkback.stanford.edu` on a dedicated 16 core server and are providing a Wordpress plugins to make TalkBack readily available to bloggers (`http://ly.tl/tb`). The TalkBack blog-side implementation in PHP is around 5000 lines of PHP and uses openSSL and mcrypt as crypto engines. Mcrypt is used to provide backward comparability to PHP version < 5.3.

In order to evaluate how TalkBack will scale, we have performed two evaluations: the first one to understand how well an authority can scale and the second evaluation to determine how well the receiving blog-side process will scale.

To understand how well authorities scale, we have evaluated how many TalkBacks a single authority (ours) is able to process per second. To make sure that

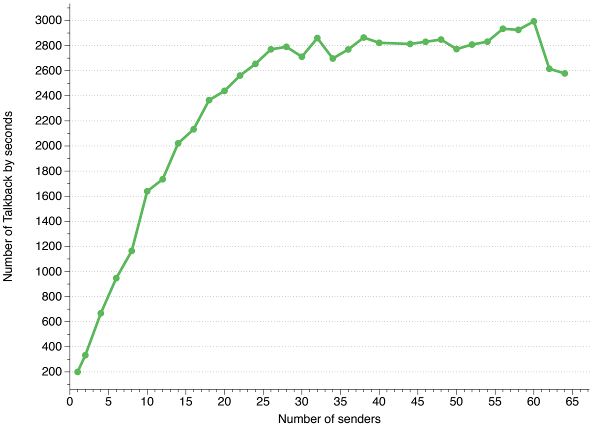


Fig. 2. Number of TalkBacks processed by second by Authority

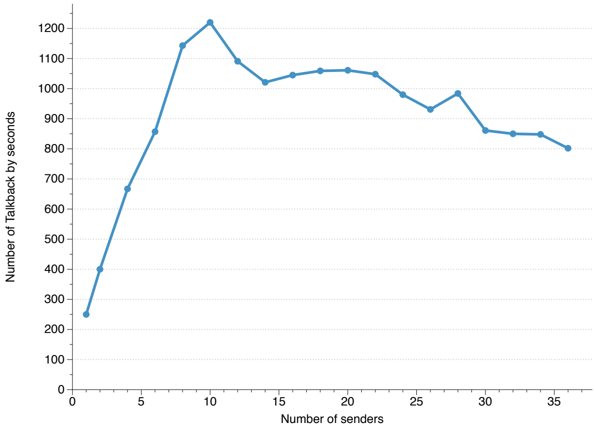


Fig. 3. Number of TalkBacks processed by second by the receiving blog

the bottleneck was on the authority, we generated ahead of time 100 000 TalkBacks and used several senders/machines to send them at once to the authority. The senders are not actual blogs but custom php scripts that use our library. As visible on figure 2, as the number of senders increases the number of TalkBacks processed increases until it reach a plateau around 2800 TalkBacks a second. This benchmark is properly evaluated on a 24-hour basis, because blogs notification is spread relatively evenly across a 24-hours period [20], due to timezone variations and blogger habits. Accordingly the fact that our authority using a single frontend is able to process around *242 Million* TalkBacks a day makes us confident that even though TalkBack is designed to allow multiple authorities our authority alone with additional frontends will be able to sustain the entire blogosphere that currently consists of around 180 Million blogs [14].

We conducted a similar experiment to see how fast a receiving blog is able to process TalkBack notifications. To make this test realistic, we used as a receiving blog Wordpress 3.0 (the latest version) equipped with our plugin. As in the previous test senders are custom scripts that send 1000 talkback notifications as fast as possible. We also used a more standard hardware platform as the blog was hosted on a 2.4GHz Intel quad core. As visible in figure 3 a single blog is able to process more than 1000 TalkBacks a second which is more than enough even for very high traffic blog. It is unlikely that a single blog will received more than *84 millions* notifications a day.

10 Additional Relevant work

In this section we present relevant work to our approach.

TrackBack Validator. The WordPress TrackBack Validator [19] looks at the sender URL to validate that the post contains the URL of the receiver. This approach increases the network load because each receiver will look at the sender's page leaving the blog vulnerable to a DDOS attack.

Reputation System. Using a reputation system alone for TrackBack spam is ineffective because an attacker may change the blog URL for every posts. Therefore any long-term classification based on TrackBack is bound to fail because there is no way to prevent spoofing (under the current TrackBack specification).

IP Blacklisting. While blacklisting based on IP might currently work as the spammers today seem to use only a small number of IPs, it is not a sustainable solution because in the long run, it is likely that spammers will use botnets and therefore have a huge pool of IPs.

Rate Limiting. Rate limiting at the blog level is not effective because a blog does not have a global view of the situation and therefore cannot stop spammers that target a huge number of blogs and post only once to each of them with the same IP.

More Relevant Work. Previous studies of spam email report that around 120 billions spam emails are sent every day [8]. In [9] and [11], the authors study a spam campaign by infiltrating the Storm botnet, while [2] analyzes the revenue generated by Storm spam. A DOS defense study [13] notes that ideas spread more quickly in the blogosphere than by email. In previous work on linkback spam, [16] examines ways that the language appearing in a blog can be used as a blocking defense. Similarly, [17] studies how the language of web pages, including blogs, can be used to detect spam. In [10] the authors use Support Vector Machines (SVM) to classify blog spam.

11 Conclusion

We propose a secure LinkBack protocol called TalkBack. This protocol is designed to prevent unauthorized LinkBack notifications by verifying blogs' authenticity and by imposing a rate limiting system. Although TalkBack adds cryptographic operations to the main LinkBack actions, we believe this level of defensive effort is appropriate, given the result reported by previous study of blog spam activity [5]. We have implemented and are maintaining the required TalkBack authority. We also provide an open-source library for integrating TalkBack into blog engines and a Wordpress plugin that makes TalkBack readily available to bloggers. Our performance evaluation shows that a single authority can sustains the entire blogosphere which makes TalkBack a viable option to defend against spammer.

References

1. Adams, C., Lloyd, S.: Understanding PKI: Concepts, Standards, and Deployment Considerations, 2nd edn. Addison-Wesley, Reading (2002)
2. Akass, C.: Storm worm 'making millions a day (February 2008), http://www.pcw.co.uk/personal-computer-world/news/2209293/strom-worm-making-millions-day
3. Apart, S.: Trackback technical specification (2004), http://www.sixapart.com/pronet/docs/trackback_spec
4. Apart, S.: Six apart guide to comment spam (2006), http://www.sixapart.com/pronet/comment_spam
5. Bursztein, E., Lam, P., Mitchell, J.C.: Trackback spam: Abuse and prevention. In: Cloud Computing Security Workshop (CCSW 2009). ACM, New York (2009)
6. Graham, B.L.: Bradland must see http comments. blog (September 1999), http://www.bradlands.com/weblog/comments/september_10_1999/
7. Haller, N.M.: The s/key one-time password system. In: Symposium on Network & Distributed Systems Security, Internet Society (1994)
8. Ironport. Internet security trends (2008), http://www.ironport.com/securitytrends
9. Kanich, C., Kreibich, C., Levchenko, K., Enright, B., Voelker, G.M., Paxson, V., Savage, S.: Spamalytics: an empirical analysis of spam marketing conversion. In: CCS 2008: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 3–14. ACM, New York (2008)
10. Kolari, P., Java, A., Finin, T., Oates, T., Joshi, A.: Detecting spam blogs: A machine learning approach. In: 2006 Proceedings of the 21st National Conference on Artificial Intelligence, AAAI (2006)
11. Kreibich, C., Kanich, C., Levchenko, K., Enright, B., Voelker, G., Paxson, V., Savage, S.: Spamcraft: An inside look at spam campaign orchestration. In: LEET. USENIX (2009)
12. Langridge, S., Hickson, I.: Pingback 1.0. Technical report, Hixie (2002)
13. Matrawy, A., Somayaji, A., Oorschot, P.C.: Mitigating network denial-of-service through diversity-based traffic management. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 104–121. Springer, Heidelberg (2005)
14. McCann, U.: Power to the people - social media tracker wave.3 (2008), http://www.universalmccann.com/Assets/wave_3_20080403093750.pdf
15. McCullagh, D., Broache, A.: Blogs turn 10 who is the father? (2010), http://news.cnet.com/2100-1025_3-6168681.html
16. Mishne, G., Carmel, D., Lempel, R.: Blocking blog spam with language model disagreement. In: Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web, AIRWeb (2005)
17. Ntoulas, A., Manasse, M.: Detecting spam web pages through content analysis. In: Proceedings of the World Wide Web Conference, pp. 83–92. ACM Press, New York (2006)
18. Automattic Production. Askimet trackback statistics (2010), http://akismet.com/stats/
19. Sandler, D., Thomas, A.: Trackback validator (2009), http://seclab.cs.rice.edu/proj/trackback/

20. Sia, K.C., Cho, J., Cho, H.K.: Efficient monitoring algorithm for fast news alerts. IEEE Transactions on Knowledge and Data Engineering, 950–961 (2007)
21. Technorati. Technorati top 100 blogs (2011), `http://technorati.com/pop/blogs/`
22. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: Captcha: Using hard ai problems for security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656. Springer, Heidelberg (2003)
23. Wikipedia. Linkback (2011), `http://en.wikipedia.org/wiki/Linkback`

A Systematic Analysis of XSS Sanitization in Web Application Frameworks

Joel Weinberger, Prateek Saxena, Devdatta Akhawe,
Matthew Finifter, Richard Shin, and Dawn Song

University of California, Berkeley

Abstract. While most research on XSS defense has focused on techniques for securing existing applications and re-architecting browser mechanisms, sanitization remains the industry-standard defense mechanism. By streamlining and automating XSS sanitization, web application frameworks stand in a good position to stop XSS but have received little research attention. In order to drive research on web frameworks, we systematically study the security of the XSS sanitization abstractions frameworks provide. We develop a novel model of the web browser and characterize the challenges of XSS sanitization. Based on the model, we systematically evaluate the XSS abstractions in 14 major commercially-used web frameworks. We find that frameworks often do not address critical parts of the XSS conundrum. We perform an empirical analysis of 8 large web applications to extract the requirements of sanitization primitives from the perspective of real-world applications. Our study shows that there is a wide gap between the abstractions provided by frameworks and the requirements of applications.

1 Introduction

Cross-site scripting (XSS) attacks are an unrelenting threat to existing and emerging web applications. Major web services such as Google Analytics, Facebook and Twitter have had XSS issues in recent years despite intense research on the subject [34, 52, 61]. Though XSS mitigation and analysis techniques have enjoyed intense focus [6, 7, 12, 13, 33, 36, 37, 39, 41, 43, 44, 47, 49, 50, 59, 64, 66, 68], research has paid little or no attention to a promising sets of tools for solving the XSS riddle—*web application frameworks*—which are gaining wide adoption [18, 21, 22, 28, 35, 42, 48, 55, 58, 69, 71]. Many of these frameworks claim that their sanitization abstractions can be used to make web applications secure against XSS [24, 69]. Though possible in principle, this paper investigates the extent to which it is presently true, clarifies the assumptions that frameworks make, and outlines the fundamental challenges that frameworks need to address to provide comprehensive XSS defense.

Researchers have proposed defenses ranging from purely server-side to browser-based or both [6, 13, 37, 43, 47, 64]. However, *sanitization* or *filtering*, the practice of encoding or eliminating dangerous constructs in untrusted data, remains the industry-standard defense strategy [45]. At present, each web application needs to implement XSS sanitization manually, which is prone to errors [7, 51]. Web frameworks offer a platform to automate sanitization in web applications, freeing developers from existing ad-hoc and error-prone manual analysis. As web applications increasingly rely on web

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 150–171, 2011.

frameworks, we must understand the assumptions web frameworks build on and the security of their underlying sanitization mechanisms.

XSS sanitization is deviously complex; it involves understanding how the web browser parses and interprets web content in non-trivial detail. Though immensely important, this issue has not been fully explained in prior XSS research. For instance, prior research does not detail the security ramifications of the complex interactions between the sub-languages implemented in the browser or the subtle variations in different interfaces for accessing or evaluating data via JavaScript's DOM API. This has important implications on the security of XSS sanitization, as we show through multiple examples in this paper. For instance, we show examples of how sanitization performed on the server-side can be effectively "undone" by the browser's parsing of content into the DOM, which may introduce XSS vulnerabilities in client-side JavaScript code.

A web framework can address XSS using sanitization if it correctly addresses all the subtleties. Whether existing frameworks achieve this goal is an important question and a subject of this paper. A systematic study of today's web frameworks should evaluate their security and assumptions along the following dimensions to quantify their benefits:

- **Context Expressiveness.** Untrusted data needs to be sanitized differently based on its *context* in the HTML document. For example, the sanitization requirements of a URI attribute are different from those of an HTML tag. *Do web frameworks provide sanitizers for different contexts that applications commonly use in practice?*
- **Auto-sanitization and Context-Sensitivity.** Applying sanitizers in code automatically, which we term *auto-sanitization*, shifts the burden of ensuring safety against XSS from developers to frameworks. However, a sanitizer that may be safe for use in one context may be unsafe for use in another. Therefore, to achieve security, auto-sanitization must be *context-sensitive*; otherwise, as we explain in Section 3.1, it may provide a false sense of security. *To what extent do modern web frameworks offer context-sensitive auto-sanitization?*
- **Security of Dynamic Client-Side Evaluation.** AJAX applications have significant client-side code components, such as in JavaScript. There are numerous subtleties in XSS sanitization because client-side code may read values from the DOM. *Do frameworks support complete mediation on DOM accesses in client-side code?*

Contributions and Approach. We explain the challenges inherent in XSS sanitization. We present a novel model of the web browser's parsing internals in sufficient detail to explain the subtleties of XSS sanitization. Our model is the first to comprehensively conceptualize the difficulties of sanitization. Our browser model includes details of the sub-languages supported by HTML5, their internal interactions, and the transductions browsers introduce on content. We provide examples of XSS scenarios that result.

This paper is a first step towards initiating research on secure web frameworks. It systematically identifies the features and pitfalls in XSS sanitization abstractions of today's web frameworks and the challenges a secure framework must address. We compare existing abstractions in frameworks to the requirements of web applications, which we derive by an empirical analysis. We study 14 mature, commercially used web frameworks and 8 popular open-source web applications. We establish whether the applications we study could be migrated to use the abstractions of today's web frameworks. We quantify the security of the abstractions in frameworks and clarify the liability developers will continue to take even if they were to migrate their applications to today's frameworks. We provide the first in-depth study of the gap between the sanitization abstractions provided by web frameworks and what web applications require for safety against XSS. We conclude that though web frameworks have the potential to secure applications against XSS, most existing frameworks fall short of achieving this goal.

2 A Systematic Browser Model for XSS

We formulate XSS with a comprehensive model of the browser's parsing behavior in Section 2.1. We discuss the challenges and subtleties XSS sanitization must address in Section 2.2, and how web frameworks could offer a potential solution in Section 2.3. We outline our evaluation objectives and formulate the dimensions along which we empirically measure the security of web frameworks in Section 2.4.

2.1 Problem Formulation: XSS Explained

Web applications mix control data (code) and content in their output, generated by server-side code, which is consumed as client-side code by the web browser. When data controlled by the attacker is interpreted by the web browser as if it was code written by the web developer, an XSS attack results. A canonical example of an XSS attack is as follows. Consider a blogging web application that emits untrusted content, such as anonymous comments, on the web page. If the developer is not careful, an attacker can input text such as `<script>...<script>`, which may be output verbatim in the server's output HTML page. When a user visits this blog page, her web browser will execute the attacker controlled text as script code.

XSS sanitization requires removal of such dangerous tags from the untrusted data. Unfortunately, not all cases are as simple as this `<script>` tag example. In the rest of this section, we identify browser features that make preventing XSS much more complicated. Previous research has indicated that this problem is complex, but we are not aware of an in-depth, systematic problem formulation.

The Browser Model. We present a comprehensive model of the web browser's parsing behavior. While the intricacies of browser parsing behavior have been discussed before [70], a formal model has not been built to fully explore its complexity. We show this model in Figure 1. Abstractly, the browser can be viewed as a collection of HTML-related sub-grammars and a collection of transducers. Sub-grammars correspond to parsers for languages such as URI schemes, CSS, HTML, and JavaScript (the rounded rectangles in Figure 1). Transducers transform or change the representation of the text, such as in HTML-entity encoding/decoding, URI-encoding, JavaScript Unicode encoding and so on (the unshaded rectangles in Figure 1). The web application's output, i.e., HTML page, is input into the browser via the network; it can be directly fed into the HTML parser after some pre-processing or it can be fed into JavaScript's HTML evaluation constructs. The browser parses these input fragments in stages—when a fragment is recognized as a term in another sub-grammar, it is shipped to the corresponding sub-grammar for reparsing and evaluation (e.g., edge 2). For example,

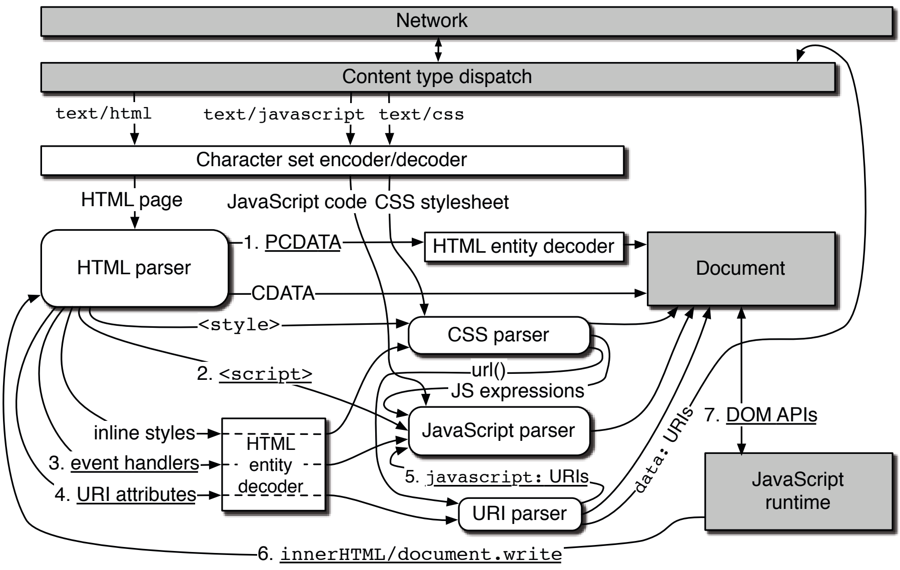


Fig. 1. Flow of Data in our Browser Model. Certain contexts such as PCDATA and CDATA directly refer to parser states in the HTML 5 specification. We refer to the numbered and underlined edges during our discussion in the text.

while the top-level HTML grammar identifies an anchor (<a>) tag in the HTML document, the contents of the href attribute are sent to the URI parser (edge 4). The URI parser handles a javascript: URI by sending its contents to the JavaScript parser (edge 3), while other URIs are sent to their respective parsers.

2.2 Subtleties and Challenges in XSS Sanitization

The model shows that the interaction between sub-components is complex; burdening developers with fully understanding their subtleties is impractical. We now describe a number of such challenges that correct sanitization-based defense needs to address.

Challenge 1: Context Sensitivity. Sanitization for XSS defense requires knowledge of where untrusted input appears structurally and semantically in the web application. For example, simple HTML-entity encoding is a sufficient sanitization procedure to neutralize XSS attacks when is placed inside the body of an HTML tag, or, in the PCDATA (edge 1) parsing context, as defined by HTML5 [30]. However, when data is placed in a resource URI, such as the src or href attribute of a tag, HTML-encoding is insufficient to block attacks such as via a javascript: URI (edge 4 and 5). We term the intuitive notion of *where* untrusted data appears as its *context*. Sanitization requirements vary by contexts. Frameworks providing sanitization primitives need to be mindful of such differences from context to context. The list of these differences is large [29].

Challenge 2: Sanitizing Nested Contexts. We can see in the model that a string in a web application's output can be parsed by multiple sub-parsers in the browser. We say that such a string is placed in *nested contexts*. That is, its interpretation in the browser will cause the browser to traverse more than one edge shown in Figure 1.

Sanitizing for nested contexts adds its own complexity. Consider an embedding of an untrusted string inside a script block, such as `<script> var x = ‘ UNTRUSTED DATA’...</script>`. In this example, when the underlined data is read by the browser, it is simultaneously placed in two contexts. It is placed in a JavaScript string literal context by the JavaScript parser (edge 2) due to the single quotes. But, before that, it is inside a `<script>` HTML tag (or `RCDATA` context according to the HTML 5 specification) that is parsed by the HTML parser. Two distinct attack vectors can be used here: the attacker could use a single quote to break out of the JavaScript string context, or inject `</script>` to break out of the script tag. In fact, sanitizers commonly fail to account for the latter because they do not recognize the presence of nested contexts.

Challenge 3: Browser Transductions. If dealing with multiple contexts is not arduous enough, our model highlights the *implicit transductions* that browsers perform when handing data from one sub-parser to another. These are represented by edges from rounded rectangles to unshaded rectangles in Figure 1. Such transductions and browser-side modifications can, surprisingly, *undo* sanitization applied on the server.

Consider a blog page in which comments are hidden by default and displayed only after a user clicks a button. The code uses an `onclick` JavaScript event handler:

```
<div class=‘comment-box’onclick=‘displayComment(" UNTRUSTED",this)’>
   ... hidden comment ... </div>
```

The underlined untrusted comment is in two nested contexts: the HTML attribute and single-quoted JavaScript string contexts. Apart from preventing the data from escaping out of the two contexts separately (Challenge 2), the sanitization must worry about an additional problem. The HTML 5 standard mandates that the browser HTML-entity decode an attribute value (edge 3) before sending it to a sub-grammar. As a result, the attacker can use additional attack characters even if the sanitization performs HTML-entity encoding to prevent attacks. The characters `&quot;` will get converted to `"` before being sent to the JavaScript parser. This will allow the untrusted comment to break out of the string context in the JavaScript parser. We call such implicit conversions *browser transductions*. Full details of the transductions are available in Appendix A.

Challenge 4: Dynamic Code Evaluation. In principle, the chain of edges traversed by the browser while parsing a text can be arbitrarily long because the browser can dynamically evaluate code. Untrusted content can keep cycling through HTML and JavaScript contexts. For example, consider the following JavaScript code fragment:

```
function foo(untrusted) {
    document.write("<input onclick=’foo(" + untrusted + ")’ >");
}
```

Since `untrusted` text is repeatedly pumped through the JavaScript string and HTML contexts (edges 3 and 6 of Figure 1), statically determining the context traversal chain on the server is infeasible. In principle, purely server-side sanitization is not sufficient for context determination because of dynamic code evaluation. Client-side sanitization is needed in these cases to fully mitigate potential attacks. Failure to properly sanitize such dynamic evaluation leads to the general class of attacks called DOM-based XSS or client-side code injection [60]. In contrast to Challenges 2 and 3, such vulnerabilities are caused not by a lack of understanding the browser, but of frameworks not understanding application behavior.

Another key observation is that browser transductions along the edges of Figure 1 vary from one edge to another, as detailed in Appendix A. This mismatch can cause XSS vulnerabilities. During our evaluation, we found one such bug (Section 3.2). We speculate that JavaScript-heavy web applications are likely to have such vulnerabilities.

Challenge 5: Character-Set Issues. Successfully sanitizing a string at the server side implicitly requires that the sanitizer and the browser are using the same character set while working with the string. A common source of XSS vulnerabilities is a mismatch in the charset assumed by the sanitizer and the charset used by the browser. For example, the ASCII string `+ADw-` does not have any suspicious characters. But when interpreted by the browser as UTF-7 character-set, it maps to the dangerous < character: this mismatch between the server-side sanitization and browser character set selection has led to multiple XSS vulnerabilities [62].

Challenge 6: MIME-Based XSS, Universal XSS, and Mashup Confinement. Browser quirks, especially in interpreting content or MIME types [10], contribute their own share of XSS vulnerabilities. Similarly, bugs in browser implementations, such as capability leaks [26] and parsing inconsistencies [9], or in browser extensions [11] are important components of the XSS landscape. However, these do not pertain to sanitization defenses in web frameworks. Therefore, we consider them to be out-of-scope for this study.

2.3 The Role of Web Frameworks

Web application development frameworks provide components to enable typical work flows in web application development. These frameworks can abstract away repetitive and complex tasks, freeing the developer to concentrate on his particular scenario. Consider session management, a common feature that is non-trivial to implement securely. Most web application frameworks automate session management, hiding this complexity from the developer. Similarly, web application frameworks can streamline and hide the complexity of XSS sanitization from the developer. In fact, increased security is often touted as a major benefit of switching to web application frameworks [24, 69].

Frameworks can either provide XSS sanitization routines in a library or they can automatically add appropriate sanitization code to a web application. We term the latter approach *auto-sanitization*. In the absence of auto-sanitization, the burden of calling the sanitizers is on the developer, which we have seen is an error-prone requirement. On the other hand, auto-sanitization, if incorrectly implemented, can give a false sense of security because a developer may defer all sanitization to this mechanism.

2.4 Analysis Objectives

In theory, use of a web application framework should free the developer from the complexities of XSS sanitization as discussed earlier and illustrated in Figure 1. If true, this requires the framework to grapple with all these complexities instead. We abstract the most important challenges into the following three dimensions:

- **Context Expressiveness and Sanitizer Correctness.** As we detailed in Challenge 1, sanitization requirements change based on the context of the untrusted data. We are interested in investigating the set of contexts in which untrusted data is used by web applications, and whether web frameworks support those contexts. In the absence of such support, a developer will have to revert to manually writing sanitization functions. The challenges outlined in Section 2.1 make manually developing *correct* sanitizers a non-starter. Instead, we ask, *do web frameworks provide correct sanitizers for different contexts that web applications commonly use in practice?*
- **Auto-sanitization and Context-Sensitivity.** Providing sanitizers is only a small part of the overall solution necessary to defend against XSS attacks. Applying sanitizers in code automatically, which we term *auto-sanitization*, shifts the burden of ensuring safety against XSS from developers to frameworks. The benefit of this is self-evident: performing correct sanitization in framework code spares each and every developer from having to implement correct sanitization himself, and from having to remember to perform that sanitization everywhere it should be performed. Furthermore, correct auto-sanitization needs to be context-sensitive—context-insensitive auto-sanitization can lead to a false sense of security. *Do web frameworks offer auto-sanitization, and if so, is it context-sensitive?*
- **Security of Client-Side Code Evaluation.** Much of the research on XSS has focused on the subtleties of parsing in HTML contexts across browsers. But AJAX web applications have significant client-side code components, such as in JavaScript. There are numerous subtleties in XSS sanitization because client-side code may read values from the DOM. Sanitization performed on the server-side may be "undone" during the browser's parsing of content into the DOM (Challenge 3 and Challenge 4). *Do frameworks support complete mediation on DOM accesses in client-side code?*

In this study, we focus solely on XSS sanitization features in web frameworks and ignore all other framework features. We also do not include purely client-side frameworks such as jQuery [1] because these do not provide XSS protection mechanisms. Additionally, untrusted data used in these libraries also needs server-side sanitization.

3 Analysis of Web Frameworks and Applications

In this section, we empirically analyze web frameworks and the sanitization abstractions they provide. We show that there is a mismatch in the abstractions provided by frameworks and the requirements of applications.

We begin by analyzing the "auto-sanitization" feature—a security primitive in which web frameworks sanitize untrusted data automatically—in Section 3.1. We identify the extent to which it is available, the pitfalls of its implementation, and whether developers can blindly trust this mechanism if they migrate to or develop applications on existing auto-sanitizing frameworks. We then evaluate the support for dynamic code evaluation via JavaScript in frameworks in Section 3.2. In the previous section, we identified subtleties in the browser's DOM interface. In Section 3.2, we discuss whether applications adequately understand it to prevent XSS bugs.

Frameworks may not provide auto-sanitization, but instead may provide sanitizers that developers can manually invoke. Arguably, the sanitizers implemented by frameworks would be more robust than the ones implemented by the application developer. We evaluate the breadth of contexts for which each framework provides sanitizers, or the *context expressiveness* of each framework, in Section 3.3. We also compare it to the requirements of the applications we study today to evaluate whether this expressiveness is enough for real-world applications.

Finally, we evaluate frameworks' assumptions regarding correctness of sanitization and compare these to the sanitization practices in security-conscious applications.

Methodology and Analysis Subjects. We examine 14 popular web application frameworks in commercial use for different programming languages and 8 popular PHP web applications ranging from 19 KLOC to 532 KLOC in size. We used a mixture of manual and automated exploration to identify sanitizers in the web application running on an instrumented PHP interpreter. We then executed the application again along paths that use these sanitization functions and parsed the outputs using an HTML 5-compliant browser to determine the contexts for which they sanitize. Due to space constraints, this paper focuses solely on the results of our empirical analysis. A technical report provides the full details of the techniques employed [65].

3.1 Auto-Sanitization: Features and Pitfalls

Auto-sanitization is a feature that shifts the burden of ensuring safety against XSS from the developer to the framework. In a framework that includes auto-sanitization, the application developer is responsible for indicating which variables will require sanitization. When the page is output, the web application framework can then apply the correct sanitizer to these variables. Our findings, summarized in Table 1, are as follows:

- Of the 14 frameworks evaluated, only 7 support some form of auto-sanitization.
- 4 out of the 7 auto-sanitization framework apply a "one-size-fits-all" strategy to sanitization. That is, they apply the same sanitizer to all flows of untrusted data irrespective of the context into which the data flows. We call this *context-insensitive* sanitization, which is fundamentally unsafe, as explained later.
- We measure the fraction of application output sinks actually protected by context-insensitive auto-sanitization mechanism in 10 applications built on Django, a popular web framework. Table 2 presents our findings. The mechanism fails to correctly protect between 14.8% and 33.6% of an application's output sinks.
- Only 3 frameworks perform context-sensitive sanitization.

No Auto-Sanitization. Only half of the studied frameworks provide any auto-sanitization support. In those that don't, developers must deal with the challenges of selecting where to apply built-in or custom sanitizers. Recent studies have shown that this manual process is prone to errors, even in security-audited applications [25, 51].

Table 1. Extent of automatic sanitization support in the frameworks we study and the pointcut (set of points in the control flow) where the automatic sanitization is applied

Language	Framework, Plugin, or Feature	Automatically Sanitizes in HTML Context	Performs Context-Aware Sanitization	Pointcut
PHP	CodeIgniter	●		Request Reception
VB, C#, C++, F#	ASP.NET Request Validation [5]	●		Request Reception
Ruby	xss_terminate Rails plugin [67]	●		Database Insertion
Python	Django	●		Template Processing
Java	GWT SafeHtml	●	●	Template Processing
C++	Ctemplate	●	●	Template Processing
Language-neutral	ClearSilver	●	●	Template Processing

We also observed instances of this phenomenon in our analysis. The following example is from a Django application called GRAMPS.

Example 1

```
{% if header.sortable %}
   <a href="{{header.url|escape}}">
{% endif %}
```

The developer sanitizes a data variable placed in the `href` attribute but uses the HTML-entity encoder (`escape`) to sanitize the data variable `header.url`. This is an instance of Challenge 2 outlined in Section 2. In particular, this sanitizer fails to prevent XSS attack vectors such as `javascript:` URIs.

Insecurity of Context-Insensitive Auto-Sanitization. Another interesting fact about the above example is that even if the developer relied on Django's default auto-sanitization, the code would be vulnerable to XSS attacks. Django employs context-insensitive auto-sanitization, i.e., it applies the same sanitizer (`escape`) irrespective of the output context. `escape`, which does an HTML entity encode, is safe for use in HTML tag context but unsafe for other contexts. In the above example, applying `escape`, automatically or otherwise, fails to protect against XSS attacks. Auto-sanitization support in Rails [67], .NET (request validation [5]) and CodeIgniter are all context-insensitive and have similar problems.

Context-insensitive auto-sanitization provides a false sense of security. On the other hand, relying on developers to pick a sanitizer consistent with the context is error-prone, and one XSS hole is sufficient to subvert the web application's integrity. Thus, because it covers some limited cases, context-insensitive auto-sanitization is better protection than no auto-sanitization.

We measure the percentage of output sinks protected by context-insensitive auto-sanitization in 10 Django-based applications that we randomly selected for further investigation [23]. We statically correlated the automatically applied sanitizer to the context of the data; the results are in Table 2. The mechanism protects between 66.4% and 85.2% of the output sinks, but conversely permits XSS vectors in 14.8% to 33.6% of the contexts, subject to whether attackers control the sanitized data or not. We did not determine the exploitability of these incorrectly auto-sanitized cases, but we observed that in most of these cases, developers resorted to custom manual sanitization.

Table 2. Usage of auto-sanitization in Django applications. The first 2 columns are the number of sinks in the templates and the percentage of these sinks for which auto-sanitization has not been disabled. Each remaining column shows the percentage of sinks that appear in the given context.

Web Application	No. Sinks	% Auto-sanitized Sinks	% Sinks not sanitized (marked safe)	% Sinks manually sanitized	% Sinks in HTML Context	% Sinks in URI Attr. (excl. scheme)	% Sinks in URI Attr. (incl. scheme)	% Sinks in JS Attr. Context	% Sinks in JS Number or String Context	% Sinks in Style Attr. Context
GRAMPS Genealogy Management	286	77.9	0.0	22.0	66.4	3.4	30.0	0.0	0.0	0.0
HicroKee's Blog	92	83.6	7.6	8.6	83.6	6.5	7.6	1.0	0.0	1.0
FabioSouto.eu	55	90.9	9.0	0.0	67.2	7.2	23.6	0.0	1.8	0.0
Phillip Jones' Eportfolio	94	92.5	7.4	0.0	73.4	11.7	12.7	0.0	2.1	0.0
EAG cms	19	94.7	5.2	0.0	84.2	0.0	5.2	0.0	0.0	10.5
Boycott Toolkit	347	96.2	3.4	0.2	71.7	1.1	25.3	0.0	1.7	0.0
Damned Lies	359	96.6	3.3	0.0	74.6	0.5	17.8	0.0	0.2	6.6
oebfare	149	97.3	2.6	0.0	85.2	6.0	8.0	0.0	0.0	0.6
Malaysia Crime	235	98.7	1.2	0.0	77.8	0.0	1.7	0.0	20.4	0.0
Philippe Marichal's web site	13	100.0	0.0	0.0	84.6	0.0	15.3	0.0	0.0	0.0

An auto-sanitization mechanism that requires developers to sanitize diligently is self-defeating. Developers should be aware of this responsibility when building on such a mechanism.

Context-Sensitive Sanitization. Context-sensitive auto-sanitization addresses the above issues. Three web frameworks, namely GWT, Google Clearsilver, and Google Ctemplate, provide this capability. In these frameworks, the auto-sanitization engine performs runtime parsing, keeping track of the context before emitting untrusted data. The correct sanitizer is then automatically applied to untrusted data based on the tracked context. These frameworks rely on developers to identify untrusted data. The typical strategy is to have developers write code in *templates*, which separate the HTML content from the (untrusted) data variables. For example, consider the following simple template supported by the Google Ctemplate framework:

Example 2

```
{{%AUTOESCAPE context="HTML"}}
<html><body><script> function showName() {
document.getElementById("sp1").textContent = "Name: {{NAME}}";} </script>
<span id="sp1" onclick="showName()">Click to display name.</span><br/>
Homepage: <a href="{{URI}}"> {{PAGENAME}} </a></body></html>
```

Variables that require sanitization are surrounded by `{{` and `}}`; the rest of the text is HTML content to be output. When the template executes, the engine parses the output and determines that `{{NAME}}` is in a JavaScript string context and automatically applies the sanitizer for the JavaScript string context, namely `:javascript_escape`. For other variables, the same mechanism applies the appropriate sanitizers. For instance, the variable `{{URI}}` is sanitized with the `:url_escape_with_arg=html` sanitizer.

3.2 Security of Client-Side Code Evaluation

In Section 2, we identified subtleties of dynamic evaluation of HTML via JavaScript's DOM API (Challenge 4). The browser applies different transductions depending on the DOM interface used (Challenge 3 and listed in Appendix A). Given the complexity of sanitizing dynamic evaluation, we believe web frameworks should provide support for this important class of XSS attack vectors too. Ideally, a web framework could incorporate knowledge of these subtleties, and provide automatic sanitization support during JavaScript code execution.

Support in Web Frameworks. The frameworks we studied do not support sanitization of dynamic flows. Four frameworks support sanitization of untrusted data used in a JavaScript string or number context (Table 3). This support is only *static*: it can ensure that untrusted data doesn't escape out during the parsing by the browser, but such sanitization can't offer any safety during dynamic code evaluation, given that dynamic code evaluation can *undo* previously applied transductions (Challenge 4).

Context-insensitivity issues with auto-sanitization also extend to JavaScript code. For example, Django uses the context-insensitive HTML-escape sanitizer even in JavaScript string contexts. Dangerous characters (e.g., `\n,\r,;`) can still break out of the JavaScript string literal context. For example, in the Malaysia Crime Application (authored in Django), the `crime.icon` variable is incorrectly auto-sanitized with HTML-entity encoding and is an argument to a JavaScript function call.

Example 3

```
map.addOverlay(new GMarker(point, {{ crime.icon }}))
```

Awareness in Web Applications. DOM-based XSS is a serious problem in web applications [49, 50]. Recent incidents in large applications, such as vulnerabilities in Google optimizer [46] scripts and Twitter [60], show that this is a continuing problem. This suggests that web applications are not fully aware of the subtleties of the DOM API and dynamic code evaluation constructs (Challenge 3 and 4 in Section 2).

To illustrate this, we present a real-world example from one of the applications we evaluated, `phpBB3`, showing how these subtleties may be misunderstood by developers.

Example 4

```
text = element.getAttribute('title');
// ... elided ...
desc = create_element('span', 'bottom');
desc.innerHTML = text;
tooltip.appendChild(desc);
```

In the server-side code, which is not shown here, the application sanitizes the `title` attribute of an HTML element by HTML-entity encoding it. If the attacker enters a string like `<script>`, the encoding converts it to `&lt;script&gt;`. The client-side code subsequently reads this attribute via the `getAttribute` DOM API in JavaScript code (shown above) and inserts it back into the DOM via the `innerHTML` method. The vulnerability is that the browser automatically decodes HTML entities (through edge 1 in Figure 1) while constructing the DOM. This effectively undoes the server's sanitization

Table 3. Sanitizers provided by languages and/or frameworks. For frameworks, we also include sanitizers provided by standard packages or modules for the language.

Language	Framework	HTML tag content or non-URI attribute	URI Attribute (excluding scheme)	URI Attribute (including scheme)	JS String	JS Number or Boolean	Style Attribute or Tag
Perl	Mason [2, 42]	●	●				
	Template Toolkit [58]	●	●				
	Jifty [35]	●	●				
PHP	CakePHP [15]	●	●				
	Smarty Template Engine [55]	●	●		●		
	Yii [32, 69]	●	●				
	Zend Framework [71]	●	●				
	CodeIgniter [19, 20]	●	●				
VB, C#, C++, F#	ASP.NET [4]	●	●				
Ruby	Rails [48]	●	●				
Python	Django [22]	●	●	●	●		
Java	GWT SafeHtml [28]	●	●	●			
C++	Ctemplate [21]	●	●	●	●	●	●
Language-neutral	ClearSilver [18]	●	●	●	●		●

in this example. The `getAttribute` DOM API reads the decoded string (e.g., `<script>`) from the DOM (edge 7). Writing `<script>` via `innerHTML` (edge 6) results in XSS.

This bug is subtle. Had the developer used `innerText` instead of `innerHTML` to write the data, or used `innerHTML` to read the data, the code would *not* be vulnerable. The reason is that the two DOM APIs discussed here read different serializations of the parsed page, as explained in Appendix A.

The prevalence of DOM-based XSS vulnerabilities and the lack of framework support suggest that this is a challenge for web applications and web frameworks alike. Libraries such as Caja and ADsafe model JavaScript and DOM manipulation but target isolation-based protection such as authority safety, not DOM-based XSS [3, 14]. Protection for this class of XSS requires further research.

3.3 Context Expressiveness

Having analyzed the auto-sanitization support in web frameworks for static HTML evaluation as well as dynamic evaluation via JavaScript, we turn to the support for manual sanitization. Frameworks may not provide auto-sanitization but instead may provide sanitizers which developers can call. This improves security by freeing the developer from (re)writing complex, error-prone sanitization code. In this section, we evaluate the breadth of contexts for which each framework provides sanitizers, or the *context expressiveness* of each framework. For example, a framework that provides built-in sanitizers for more than one context, say in URI attributes, CSS keywords, JavaScript string contexts, is more expressive than one that provides a sanitizer only for HTML tag context.

Expressiveness of Framework Sanitization Contexts. Table 3 presents the expressiveness of web frameworks we study and Table 4 presents the expressiveness required by our subject web applications. The key insights are:

- We observe that 9 out of the 14 frameworks do not support contexts other than the HTML context (e.g., as the content body of a tag or inside a non-URI attribute) and the URI attribute context. The most common sanitizers for these are HTML entity encoding and URI encoding, respectively.
- 4 web frameworks, ClearSilver, Ctemplate, Django, and Smarty, provide appropriate sanitization functions for emitting untrusted data into a JavaScript string. Only 1 framework, Ctemplate, provides a sanitizer for emitting data into JavaScript outside of the string literal context. However, the sanitizer is a restrictive whitelist, allowing only numeric or boolean literals. No framework we studied allows untrusted JavaScript code to be emitted into JavaScript contexts. Supporting this requires a client-side isolation mechanism such as ADsafe [3] or Google's Caja [14].
- 4 web frameworks, namely Django, GWT, Ctemplate, and Clearsilver, provide sanitizers for URI attributes in which a complete URI (i.e., including the URI protocol scheme) can be emitted. These sanitizers reject URIs that use the `javascript:` scheme and accept only a whitelist of schemes, such as `http:`.
- Of the frameworks we studied, we found only one that provides an interface for customizing the sanitizer for a given context. Yii uses HTML Purifier [32], which allows the developer to specify a custom list of allowed tags. For example, a developer may specify a policy that allows only `<b>` tags. The other frameworks (even the context-sensitive auto-sanitizing ones) have sanitizers that are not customizable. That is, untrusted content within a particular context is always sanitized the same way. Our evaluation of web applications strongly invalidates this assumption, showing that applications often sanitize data occurring in the same context differently based on other attributes of the data.

The set of contexts for which a framework provides sanitizers gives a sense of how the framework expects web applications to behave. Specifically, frameworks assume applications will not emit sanitized content into multiple contexts. More than half of the frameworks we examined do not expect web applications to insert content with arbitrary schemes into URI contexts, and only one of the frameworks supports use of untrusted content in JavaScript `Number` or `Boolean` contexts. Below, we challenge these assumptions by quantifying the set of contexts for which applications need sanitizers.

Expressiveness of Contexts and Sub-context Variance in Web Applications. We examined our 8 subject PHP applications, ranging from 19 to 532 KLOC, to understand what expressiveness they require and whether they could, theoretically, migrate to the existing frameworks. We systematically measure and enumerate the contexts into which these applications emit untrusted data. Table 4 shows the result of this evaluation. We observe that nearly all of the applications insert untrusted content into all of the outlined contexts. Contrast this with Table 3, where most frameworks support a much more limited set of contexts with built-in sanitizers.

More surprisingly, we find that applications often employ more than one sanitizer for each context. That is, an abstraction that ties a single sanitizer to a given context may be insufficient. We term this variation in sanitization across code paths *sub-context variance*. Sub-context variance evidence suggests that directly migrating web applications

Table 4. The web applications we study and the contexts for which they sanitize

Application	Description	LOC	HTML Context	URI Attr. (excl. scheme)	URI Attr. (incl. scheme)	JS Attr. Context	JS Number or String Context	No. Sanitizers	No. Sinks
RoundCube	IMAP Email Client	19,038	●	●	●	●	●	30	75
Drupal	Content Management System	20,995	●	●	●	●	●	32	2557
Joomla	Content Management System	75,785	●	●	●	●		22	538
WordPress	Blogging Application	89,504	●	●	●	●		95	2572
MediaWiki	Wiki Hosting Application	125,608	●	●	●	●	●	118	352
PHPBB3	Bulletin Board Software	146,991	●	●	●	●	●	19	265
OpenEMR	Medical Records Management	150,384	●			●	●	18	727
Moodle	E-Learning Software	532,359	●	●	●	●	●	43	6282

to web frameworks' (auto-) sanitization support may not be directly possible given that even context-sensitive web frameworks rigidly apply one sanitizer for a given context.

Sub-context variance is particularly common in the form of *role-based* sanitization, where the application applies different sanitizers based on the privilege of the user. We found that it is common to have a policy in which the site administrator's content is subject to no sanitization (by design). Examples include phpBB, WordPress, Drupal. For such simple policies, there are legitimate code paths that have no sanitization requirements. To illustrate this, we present a real-world example from the popular WordPress application which employs different sanitization along different code paths.

Example 5

WordPress, the popular blogging application, groups users into *roles*. A user in the author role can create a new post on the blog with most non-code tags permitted. An anonymous commenter, on the other hand, can only use a small number of text formatting tags. In particular, the latter cannot insert images in comments while an author can insert images in his post. Note that neither can insert `<script>` tags, or any other active content. In both cases, untrusted input flows into HTML tag context, but the sanitizer applied changes as a function of the user role.

Most auto-sanitizing frameworks do not support such rich abstractions to support auto-sanitization specifications at a sub-context granularity. Nearly all sanitization libraries (not part of web frameworks) are customizable. However, their connection to special role-based sanitization (or similar cases) are not supported presently. We believe that web frameworks can fill this gap. Only 1 framework, Yii, provides the flexibility to handle such customizations using the HTMLPurifier sanitization library. Unfortunately, Yii only provides this flexibility for the HTML tag context.

3.4 Enabling Reasoning of Sanitizer Correctness

Prior research on web applications has shown that developing sanitization functions, especially custom sanitizers, is tricky and prone to errors [7]. We investigate how the sanitizers in web frameworks handle this issue. We compare the structure of the sanitizers used in frameworks to the structure we observe in our subject applications and characterize the ground assumptions that developers should be aware of.

Blacklists vs. Whitelists. We find that most web frameworks structure their sanitizers as a *declarative-style whitelist* of code constructs explicitly allowed in untrusted content. For instance, one sanitization library employed in the Yii is HTML-Purifier [32], which permits a declarative list of HTML elements like event attributes of special tags in untrusted content. All of the web applications we studied also employ this whitelisting mechanism, such as the KSES library used in Wordpress [38]. Such sanitizers assume that the whitelist is only contains a well understood and safe subset of the language specification, and does not permit any unsafe structures.

In contrast, we find that only 1 subject web framework, viz. CodeIgniter, employs a blacklist-based sanitization approach. Even if one verifies that all the elements on a blacklist conform to an unsafe subset of the language specification, the sanitizer may still allow unsafe parts of the language. For example, CodeIgniter's `xss_clean` function removes a blacklist of potentially dangerous strings like `document.cookie` that may appear in any context. Even if it removes *all* references to `document.cookie`, there still may be other ways for attacker code to reference cookies, such as via `document['cookie']`.

Correctness of the sanitizers used is fundamental to the safety of the sanitization based strategy used in web frameworks. Based on the above examples, we claim that it is easier to verify that a whitelist policy is safe and recommend frameworks adopt such a strategy.

HTML Canonicalization. Essential to the safety of sanitization-based defense is that the user's browser parse the untrusted string in a manner consistent with the parsing applied by the sanitizer. For instance, if the context-determination in the frameworks differs from the actual parsing in the browser, the wrong sanitizer could be applied by the framework.

We observe that frameworks employ a *canonicalization* strategy to ensure this property; the web frameworks identify a 'canonical' subset of HTML-related languages into which all application output is generated. The assumption they rely on is that this canonical form parses the same way across major web browsers. We point out explicitly that these assumptions are not systematically verified today and, therefore, framework outputs may still be susceptible to XSS attacks. For example, a recent XSS vulnerability in the HTML Purifier library (used in Yii) was traced back to "quirks in Internet Explorer's parsing of string-like expressions in CSS [31]."

Finally, we point out that sanitization-based defense isn't the only alternative—proposals for *sanitization-free* defenses, such as DSI [43], BLUEPRINT [59] and the Content Security Policy [56] have been presented. Future frameworks could consider these. Verifying the safety of the whitelist-based canonicalization strategy and its assumptions also deserves research attention.

4 Related Work

XSS Analysis and Defense. Much of the research on cross-site scripting vulnerabilities has focused on finding XSS flaws in web applications, specifically on server-side code [7, 33, 36, 39–41, 44, 66, 68] but also more recently on JavaScript code [8, 27, 49, 50]. These works have underscored the two main causes of XSS vulnerabilities: *identifying untrusted data* at output and *errors in sanitization* by applications. There have been three kinds of defenses: purely server-side, purely browser-based, and those involving both client and server collaboration.

BLUEPRINT [59], SCRIPTGARD [51] and XSS-GUARD [13] are three server-side solutions that have provided insight into context-sensitive sanitization. In particular, BLUEPRINT provides a deeper model of the web browser and points out that browsers differ in how the various components communicate with one another. The browser model detailed in this work builds upon BLUEPRINT's model and more closely upon SCRIPTGARD's formalization [51]. We provide additional details in our model to demystify the browser's parsing behavior and explain subtleties in sanitization that the prior work did not address.

Purely browser-based solutions, such as XSSAuditor, are implemented in modern browsers. These mechanisms are useful in nullifying common attack scenarios by observing HTTP requests and intercepting HTTP responses during the browser's parsing. However, they do not address the problem of separating untrusted from trusted data, as pointed out by Barth et al. [12].

BEEP, DSI and NonceSpaces investigated client-server collaborative defenses. In these proposals, the server is responsible for identifying untrusted data, which it reports to the browser, and the browser ensures that XSS attacks can not result from parsing the untrusted data. While these proposals are encouraging, they require browser and server modifications. The closest practical implementation of such client-server defense architecture is the recent *content security policy* specification [56].

Correctness of Sanitization. While several systems have analyzed server-side code, the SANER [7] system empirically showed that custom sanitization routines in web applications can be error-prone. FLAX [50] and KUDZU [49] empirically showed that sanitization errors are not uncommon in client-side JavaScript code. While these works highlight examples, the complexity of the sanitization process remained unexplained. Our observation is that sanitization is pervasively used in emerging web frameworks as well as large, security-conscious applications. We discuss whether applications should use sanitization for defense in light of previous bugs.

Techniques for Separating Untrusted Content. Taint-tracking based techniques [16, 36, 44, 53, 63, 68] as well as security-typed languages [17, 47, 54, 57] aim to address the problem of identifying and separating untrusted data from HTML output to ensure that untrusted data gets sanitized before it is output. Web templating frameworks, some of which are studied in this work, offer a different model in which they coerce developers into explicitly specifying trusted content. This offers a fail-closed design and has seen adoption in practice because of its ease of use.

5 Conclusions and Future Work

We study the sanitization abstractions provided in 14 web application development frameworks. We find that frameworks often fail to comprehensively address the subtleties of XSS sanitization. We also analyze 8 web applications, comparing the sanitization requirements of the applications against the abstractions provided by the frameworks. Through real-world examples, we quantify the gap between what frameworks provide and what applications require.

Auto-sanitization Support and Context Sensitivity. Automatic sanitization is a step in the right direction. For correctness, auto-sanitization needs to be context-sensitive: context-insensitive sanitization can provide a false sense of security. Our application study finds that applications do, in fact, need to emit untrusted data in multiple contexts. However, the total number of contexts used by applications in our study is limited, suggesting that frameworks only need to support a useful subset of contexts.

Security of Client-side Code Evaluation. DOM-based XSS is a serious challenge in web applications, but no framework supports sanitization for dynamic evaluation on the client. Application developers must be particularly alert when using the DOM API. Of particular relevance to XSS sanitization is the possibility of the browser "undoing" server-side sanitization, making the application vulnerable to DOM-based XSS.

Context Expressiveness and Sanitizer Correctness. Some frameworks offer sanitization primitives as library functions the developer can invoke. We find that most frameworks do not provide sufficiently expressive sanitizers, i.e., the sanitizers provided do not support all the contexts that applications use. For instance, applications emit untrusted data into URI attribute and JavaScript literal contexts, but most of the frameworks we study do not provide sanitizers for these contexts. As a result, application developers must implement these security-critical sanitizers themselves, a tedious and error-prone exercise. We also find that sub-context variance, such as role-based sanitizer selection, is common. Only one of the frameworks we examined provides any support for this pattern, and its support is limited.

Finally, our study identifies the set of assumptions fundamental to frameworks. Namely, frameworks assume that their sanitizers can be verified for correctness, and that HTML can be canonicalized to a single, standard form. Developers need to be aware of these assumptions before adopting a framework.

Future Directions. As we outline in this work, the browser's parsing and transformation of the web content is complex. If we develop a formal abstract model of the web browser's behavior for HTML 5, sanitizers can be automatically checked for correctness. Our browser model is a first step in this direction. We identify that parts of the web browser are either transducers or language recognizers. There have been practical guides for dealing with these issues, but a formal model of the semantics of browsers could illuminate all of the intricacies of the browser [70]. Verification techniques and tools for checking correctness properties of web code is an active area of research.

If one can show the correctness of a framework's sanitizers, we can prove the security and correctness for code generated from it. Though existing auto-sanitization mechanisms are weak today, they can be improved. Google AutoEscape is one attempt at this type of complete sanitization but is currently limited to a fairly restrictive templating language [21]. If these abstractions can be extended to richer web languages, it would provide a basis to build web applications secure from XSS from the ground up—an important future direction for research.

Acknowledgments. This material is based on work partially supported by the National Science Foundation (NSF) under Grants No. 0311808, No. 0832943, No. 0448452, No. 0842694, and No. CCF-0424422, and a NSF Graduate Research Fellowship, as well as by the Air Force Office of Scientific Research under Grant No. A9550-09-1-0539. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. jQuery, `http://jquery.com/`
2. Aas, G.: CPAN: URI::Escape, `http://search.cpan.org/~gaas/URI-1.56/URI/Escape.pm`
3. Adsafe : Making javascript safe for advertising, `http://www.adsafe.org/`
4. How To: Prevent Cross-Site Scripting in ASP.NET, `http://msdn.microsoft.com/en-us/library/ff649310.aspx`
5. Microsoft ASP.NET: Request Validation – Preventing Script Attacks, `http://www.asp.net/LEARN/whitepapers/request-validation`
6. Athanasopoulos, E., Pappas, V., Krithinakis, A., Ligouras, S., Markatos, E., Karagiannis, T.: xJS: practical XSS prevention for web application development. In: Proceedings of the 2010 USENIX Conference on Web Application Development (2010)
7. Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA (2008)
8. Bandhakavi, S., King, S.T., Madhusudan, P., Winslett, M.: Vex: Vetting browser extensions for security vulnerabilities (2010)
9. Baron, D.: Mozilla's quirks mode, `https://developer.mozilla.org/en/mozilla's_quirks_mode`
10. Barth, A., Caballero, J., Song, D.: Secure content sniffing for web browsers *or* how to stop papers from reviewing themselves. In: Proceedings of the 30th IEEE Symposium on Security and Privacy, Oakland, CA (May 2009)
11. Barth, A., Felt, A.P., Saxena, P., Boodman, A.: Protecting browsers from extension vulnerabilities (2009)
12. Bates, D., Barth, A., Jackson, C.: Regular expressions considered harmful in client-side xss filters. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 91–100. ACM, New York (2010)
13. Bisht, P., Venkatakrishnan, V.: XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In: Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 23–43 (2008)
14. Google-caja: A source-to-source translator for securing javascript-based web content, `http://code.google.com/p/google-caja/`
15. CakePHP: Sanitize Class Info, `http://api.cakephp.org/class/sanitize`
16. Chin, E., Wagner, D.: Efficient character-level taint tracking for java. In: Proceedings of the 2009 ACM Workshop on Secure Web Services, SWS 2009, pp. 3–12. ACM, New York (2009)

17. Chong, S., Liu, J., Myers, A.C., Qi, X., Vikram, K., Zheng, L., Zheng, X.: Secure web applications via automatic partitioning. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, pp. 31–44. ACM, New York (2007)
18. ClearSilver: Template Filters, http://www.clearsilver.net/docs/man_filters.hdf
19. CodeIgniter/system/libraries/Security.php, http://bitbucket.org/ellislab/codeigniter/src/tip/system/libraries/Security.php
20. CodeIgniter User Guide Version 1.7.2: Input Class, http://codeigniter.com/user_guide/libraries/input.html
21. Ctemplate: Guide to Using Auto Escape, http://google-ctemplate.googlecode.com/svn/trunk/doc/auto_escape.html
22. django: Built-in template tags and filters, http://docs.djangoproject.com/en/dev/ref/templates/builtins
23. Django sites : Websites powered by django, http://www.djangosites.org/
24. The Django Book: Security, http://www.djangobook.com/en/2.0/chapter20/
25. Finifter, M., Wagner, D.: Exploring the Relationship Between Web Application Development Tools and Security. In: Proceedings of the 2nd USENIX Conference on Web Application Development. USENIX (June 2011)
26. Finifter, M., Weinberger, J., Barth, A.: Preventing capability leaks in secure javascript subsets. In: Proc. of Network and Distributed System Security Symposium (2010)
27. Guha, A., Krishnamurthi, S., Jim, T.: Using static analysis for ajax intrusion detection. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 561–570. ACM, New York (2009)
28. Google Web Toolkit: Developer's Guide – SafeHtml, http://code.google.com/webtoolkit/doc/latest/DevGuideSecuritySafeHtml.html
29. Hansen, R.: XSS cheat sheet (2008)
30. Hickson, I.: HTML 5 : A vocabulary and associated apis for html and xhtml, http://www.w3.org/TR/html5/
31. HTML Purifier Team: Css quoting full disclosure (2010), http://htmlpurifier.org/security/2010/css-quoting
32. HTML Purifier : Standards-Compliant HTML Filtering, http://htmlpurifier.org/
33. Huang, Y.W., Yu, F., Hang, C., Tsai, C.H., Lee, D.T., Kuo, S.Y.: Securing web application code by static analysis and runtime protection. In: Proceedings of the 13th International Conference on World Wide Web, WWW 2004, pp. 40–52. ACM, New York (2004)
34. Jean, J.: Facebook CSRF and XSS vulnerabilities: Destructive worms on a social network, http://seclists.org/fulldisclosure/2010/Oct/35
35. JiftyManual, http://jifty.org/view/JiftyManual
36. Jovanovic, N., Krügel, C., Kirda, E.: Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In: IEEE Symposium on Security and Privacy (2006)
37. Kirda, E., Kruegel, C., Vigna, G., Jovanovic, N.: Noxes: a client-side solution for mitigating cross-site scripting attacks. In: Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 330–337. ACM, New York (2006)
38. KSES Developer Team: Kses php html/xhtml filter, http://sourceforge.net/projects/kses/
39. Livshits, B., Lam, M.S.: Finding security errors in Java programs with static analysis. In: Proceedings of the Usenix Security Symposium (2005)
40. Livshits, B., Martin, M., Lam, M.S.: SecuriFly: Runtime protection and recovery from Web application vulnerabilities. Tech. rep., Stanford University (September 2006)
41. Martin, M., Lam, M.S.: Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In: 17th USENIX Security Symposium (2008)
42. The Mason Book: Escaping Substitutions, http://www.masonbook.com/book/chapter-2.mhtml

43. Nadji, Y., Saxena, P., Song, D.: Document structure integrity: A robust basis for cross-site scripting defense. In: NDSS (2009)
44. Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically hardening web applications using precise tainting. In: 20th IFIP International Information Security Conference (2005)
45. XSS Prevention Cheat Sheet, `http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet`
46. Pullicino, J.: Google XSS Flaw in Website Optimizer Explained (December 2010), `http://www.acunetix.com/blog/web-security-zone/articles/google-xss-website-optimizer-scripts/`
47. Robertson, W., Vigna, G.: Static enforcement of web application integrity through strong typing. In: Proceedings of the 18th Conference on USENIX Security Symposium, SSYM 2009, pp. 283–298. USENIX Association, Berkeley (2009)
48. Ruby on Rails Security Guide, `http://guides.rubyonrails.org/security.html`
49. Saxena, P., Akhawe, D., Hanna, S., Mao, F., McCamant, S., Song, D.: A symbolic execution framework for javascript. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 513–528. IEEE Computer Society, Washington, DC, USA (2010)
50. Saxena, P., Hanna, S., Poosankam, P., Song, D.: FLAX: Systematic discovery of client-side validation vulnerabilities in rich web applications. In: 17th Annual Network & Distributed System Security Symposium NDSS (2010)
51. Saxena, P., Molnar, D., Livshits, B.: Scriptgard: Preventing script injection attacks in legacy web applications with automatic sanitization. Tech. rep., Microsoft Research (September 2010)
52. Schmidt, B.: Google Analytics XSS Vulnerability, `http://spareclockcycles.org/2011/02/03/google-analytics-xss-vulnerability/`
53. Schwartz, E.J., Avgerinos, T., Brumley, D.: All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 317–331. IEEE Computer Society, Washington, DC, USA (2010)
54. Seo, J., Lam, M.S.: Invisitype: Object-oriented security policies (2010)
55. Smarty Template Engine: escape, `http://www.smarty.net/manual/en/language.modifier.escape.php`
56. Stamm, S.: Content security policy (2009), `https://wiki.mozilla.org/Security/CSP/Spec`
57. Swamy, N., Corcoran, B., Hicks, M.: Fable: A language for enforcing user-defined security policies. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2008)
58. Template::Manual::Filters, `http://template-toolkit.org/docs/manual/Filters.html`
59. Mike, T.L., Venkatakrishnan, V.N.: BluePrint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers. In: Proceedings of the IEEE Symposium on Security and Privacy (2009)
60. TwitPwn: DOM based XSS in Twitterfall (2009), `http://www.twitpwn.com/2009/07/motb-08-dom-based-xss-in-twitterfall.html`
61. Twitter: All about the “onMouseOver” incident, `http://blog.twitter.com/2010/09/all-about-onmouseover-incident.html`
62. UTF-7 XSS Cheat Sheet, `http://openmya.hacker.jp/hasegawa/security/utf7cs.html`
63. Venema, W.: Taint support for PHP (2007), `ftp://ftp.porcupine.org/pub/php/php-5.2.3-taint-20071103.README.html`

64. Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Cross site scripting prevention with dynamic data tainting and static analysis. In: Proceeding of the Network and Distributed System Security Symposium (NDSS), vol. 42. Citeseer (2007)
65. Weinberger, J., Saxena, P., Akhawe, D., Finifter, M., Shin, R., Song, D.: An empirical analysis of xss sanitization in web application frameworks. Tech. Rep. UCB/EECS-2011-11, EECS Department, University of California, Berkeley (February 2011)
66. Xie, Y., Aiken, A.: Static detection of security vulnerabilities in scripting languages. In: Proceedings of the Usenix Security Symposium (2006)
67. xssterminate, `http://code.google.com/p/xssterminate/`
68. Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In: Proceedings of the 15th USENIX Security Symposium, pp. 121–136 (2006)
69. Yii Framework: Security, `http://www.yiiframework.com/doc/guide/1.1/en/topics.security`
70. Zalewski, M.: Browser security handbook. Google Code (2010), `http://code.google.com/p/browsersec/wiki/Part1`
71. Zend Framework: Zend_Filter, `http://framework.zend.com/manual/en/zend.filter.set.html`

A Transductions in the Browser

Table 5 details browser transductions that are automatically performed upon reading or writing to the DOM. The DOM property denotes the various aspects of an element accessible through the DOM APIs, while the access method describes the specific part of the API through which a developer may edit or examine these attributes. Excepting "specified in markup", the methods are all fields or functions of DOM elements.

Table 6 describes the specifics of the transducers employed by the browser. Except for "HTML entity decoding", the transductions all occur in the parsing and serialization processes triggered by reading and writing these properties as strings. When writing to a property, the browser parses the string to create an internal AST representation. When reading from a property, the browser recovers a string representation from the AST.

Textual values are HTML entity decoded when written from the HTML parser to the DOM via edge 1 in Figure 1. Thus, when a program reads a value via JavaScript, the value is entity decoded. In some cases, the program must re-apply the sanitization to this decoded value or risk having the server's sanitization negated.

One set of DOM read access APIs creates a serialized string of the AST representation of an element, as described in Table 6. The other API methods simply read the text values of the string versions (without serializing the ASTs to a string) and perform no canonicalization of the values.

The transductions vary significantly for the DOM write access API as well, as detailed in Table 5. Some writes cause input strings to be parsed into an internal AST representation, or apply simple replacements on certain character sequences (such as URI percent-decoding), while others store the input as is.

In addition, the parsers in Figure 1 apply their own transductions internally on certain pieces of their input. The CSS and JavaScript parsers unescape certain character sequences within string literals (such as Unicode escapes), and the URI parser applies some of its own as well (undoing percent-encoding).

Table 5. Transductions applied by the browser for various accesses to the document. These summarize transductions when traversing edges connected to the "Document" block in Figure 1.

DOM property	**Access method**	**Transductions on reading**	**Transductions on writing**
`data-*` attribute	`get/setAttribute`	None	None
	`.dataset`	None	None
	specified in markup	N/A	HTML entity decoding
src, href attributes	`get/setAttribute`	None	None
	`.src`, `.href`	URI normalization	None
	specified in markup	N/A	HTML entity decoding
id, alt, title, type, lang, class dir attributes	`get/setAttribute`	None	None
	.[attribute name]	None	None
	specified in markup	N/A	HTML entity decoding
style attribute	`get/setAttribute`	None	None
	`.style.*`	CSS serialization	CSS parsing
	specified in markup	N/A	HTML entity decoding
HTML contained by node	`.innerHTML`	HTML serialization	HTML parsing
Text contained by node	`.innerText`, `.textContent`	None	None
HTML contained by node, including the node itself	`.outerHTML`	HTML serialization	HTML parsing
Text contained by node, surrounded by markup for node	`.outerText`	None	None

Table 6. Details regarding the transducers mentioned in Table 5. They all involve various parsers and serializers present in the browser for HTML and its related sub-grammars.

Type	**Description**	**Illustration**
HTML entity decoding	Replacement of character entity references with the actual characters they represent.	`&amp;` → `&`
HTML parsing	Tokenization and DOM construction following the HTML parsing rules, including entity decoding as appropriate.	`<p>&gt;</p>` → *HTML element* P *with body* `>`
HTML serialization	Creating a string representation of an HTML node and its children.	*HTML element* P *with body* `>` → `<p>&gt;</p>`
URI normalization	Resolving the URI to an absolute one, given the context in which it appears.	`/article title` → `http://www.example.com/article%20title`
CSS parsing	Parsing CSS declarations, including character escape decoding as appropriate.	`color: \72\65\64` → `color: red`
CSS serialization	Creating a canonical string representation of a CSS style declaration.	"`color:#f00`" → "`color: rgb(255, 0, 0);` "

Who Wrote This Code? Identifying the Authors of Program Binaries

Nathan Rosenblum, Xiaojin Zhu, and Barton P. Miller

University of Wisconsin,
Madison, Wisconsin
{nater,jerryzhu,bart}@cs.wisc.edu

Abstract. Program authorship attribution—identifying a programmer based on stylistic characteristics of code—has practical implications for detecting software theft, digital forensics, and malware analysis. Authorship attribution is challenging in these domains where usually only binary code is available; existing source code-based approaches to attribution have left unclear whether and to what extent programmer style survives the compilation process. Casting authorship attribution as a machine learning problem, we present a novel program representation and techniques that automatically detect the *stylistic features* of binary code. We apply these techniques to two attribution problems: identifying the precise author of a program, and finding stylistic similarities between programs by unknown authors. Our experiments provide strong evidence that programmer style is preserved in program binaries.

1 Introduction

Program authorship attribution has immediate implications for the security community, particularly in its potential to significantly impact applications like plagiarism detection [17] and digital forensics [13]. The central thesis of authorship attribution is that authors imbue their works with an individual style; while attribution research has historically focused on literary documents [7], computer programs are no less the product of a creative process, one in which opportunities for stylistic expression abound. Previous studies of program authorship attribution have been limited to source code [6, 10], and rely on surface characteristics like spacing and variable naming, both of which reflect the essentially textual nature of program source. In many domains, such as analysis of commercial software or malware, source code is usually unavailable. Program binaries, however, retain none of the surface characteristics used in source code attribution; such details are stripped away in the compilation process. Adapting program authorship attribution to the binary domain—to identify known malware authors or detect new ones, e.g., or to discover theft of commercial software—requires new ways to recognize the style of individual authors.

We have developed novel authorship attribution techniques that automatically discover the stylistic characteristics of binary code. We adopt a *machine learning approach*, defining a large number of simple *candidate features* and using *training*

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 172–189, 2011.

data to automatically discover which features are indicative of programmer style. This approach avoids the problem of choosing good stylistic features *a priori*, which has been the focus of source code attribution [18], and which is the primary challenge for attribution in the binary domain. We apply our techniques to two related binary code authorship problems: identifying the author of a program out of a set of candidates, and grouping programs by stylistic similarity, respectively developing *classification* and *clustering* models that build on stylistic features of binary code.

In this paper, we explore various aspects of these previously unstudied problems, examining trade-offs in different program representations and several attribution scenarios. This study demonstrates that programmer style is reflected in binary code, and lays the groundwork for authorship attribution applications in a variety of domains. Our paper makes the following contributions:

- We introduce the problem of binary code authorship attribution and define a program representation in terms of *stylistic features* that differentiate different programmers; we provide an algorithm for automatically selecting stylistic features using a set of simple *feature templates* that cover a broad range of program details.
- We formulate two program authorship tasks: (1) discriminating between programs written by different authors (*authorship identification*), and (2) grouping together stylistically similar programs (*authorship clustering*). We use information derived from the authorship identification task to improve the performance of authorship clustering.
- We evaluate binary program authorship attribution on several large sets of programs from the Google Code Jam programming competition[1] and from student projects from an undergraduate operating systems course at the University of Wisconsin. Our results show that programmer style is preserved through the compilation process; a classifier trained on stylistic features can discriminate among programs written by ten different authors with 81% accuracy.

1.1 Overview

Our authorship attribution techniques are based on the hypothesis that programmer style is preserved throughout the compilation process, as suggested by the differences depicted in Figure 1 between implementations of the same functionality by two different programmers. Evaluating this hypothesis requires solving two problems: (1) choosing a program representation broad enough to capture any residual stylistic characteristics, and (2) selecting those representational elements that actually reflect programmer style. The second problem is particularly important for authorship clustering; author identity is just one property of many for a given program, and if the representation reflects more than

[1] `http://code.google.com/codejam/`

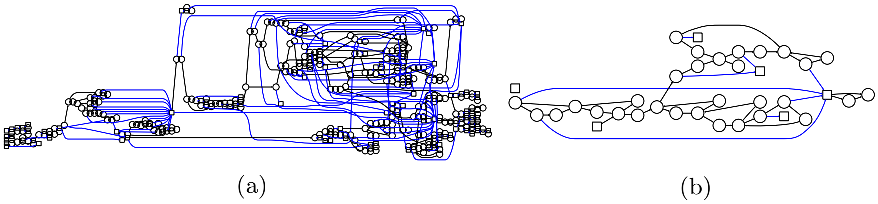


Fig. 1. The control flow graphs for two implementations of the same program by different authors. Program (a) is implemented as many small subroutines and makes use of several C++ STL classes; program (b) is almost entirely implemented as a monolithic C function.

just stylistic characteristics, a clustering algorithm may group programs according to some other property, such as program functionality. Rather than design complicated features to capture specific facets of programmer style, we define a large number of simple features that capture local and global code details at the instruction and control flow level. We adopt a machine learning approach to the problem, letting the data determine the features which best capture authorship; this data-driven policy informs our high-level workflow:

1. We collect several large corpora of programs with known authorship; these programs provide a *ground truth*, which is used to discover the stylistically important features of binary code, as well as reference points on which to evaluate authorship attribution techniques.

2. Using existing software for recursive traversal parsing [14], we extract a control flow graph and the instruction sequence for each binary, which we use as a basis for the features we describe in the following section.

3. A subset of the features that correlate with programmer style is selected. We compute the *mutual information* between features and programmer identity on a *training set* of labeled programs, ranking features according to their correlation with particular programmers. This approach is heuristic and does not take into consideration the interaction between multiple features; the learning algorithms we apply to this feature representation are responsible for refining the stylistic importance of these features.

4. We use the training set of labeled programs to build an authorship classifier based on support vector machines [3]. The classifier chooses the most likely author of a program based on its stylistic feature representation.

5. Classification is not possible for collections of programs with no training data; instead, we use the k-means clustering algorithm [1] to group programs together by stylistic similarity. To avoid clustering according to the wrong property (e.g. program functionality), we *transfer* knowledge between a *supervised* domain (a set of programs with different authors) to this *unsupervised* domain: we use the *large margin nearest neighbors* algorithm [20] to

learn a *distance metric* over a labeled set of programs, then used this metric to transform the unlabeled data prior to clustering.

In the following sections, we describe our binary code representation (2) and formally state the models and procedures we use for author classification (3) and clustering (4). We evaluate our techniques over several large program data sets (5), exploring several trade-offs inherent in binary authorship attribution. We conclude with a discussion of issues raised by this study and future directions for attribution research (6) and a review of the related literature (7).

2 Binary Code Representation

We base our binary code representation on instruction-level and structural characteristics of programs. The first step in obtaining this representation is to parse the program binary. We use the ParseAPI [14] library to extract instructions and build interprocedural *control flow graphs* from binaries, where a CFG is a directed graph $G = (V, E, \tau)$ defined by:

- the *basic block* nodes V comprising the executable code,
- the edges $E \subseteq V \times V$ representing control flow, and
- a labeling function $\tau : E \rightarrow \mathcal{T}$ corresponding to the type of the edge.

The control flow graph and underlying machine code form the basis for *feature templates*: patterns that instantiate into many concrete features of a particular binary. We first describe two feature templates, *idioms* and *graphlets*, used in our previous work on toolchain provenance [16], and then introduce new templates that capture additional properties of the binary. We stress that these features are not designed to capture any specific notion of programmer style, but rather to express many different characteristics of binary code; we use machine learning algorithms to pick out the stylistically significant features.

2.1 Idioms

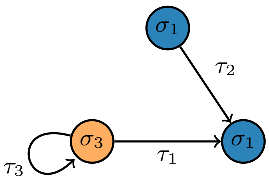
The idiom feature template captures low-level details of the instruction sequence underlying a program. Idioms are short sequences of instructions, possibly with wildcards, which we have previously used to recognize compiler-specific code patterns; for example, the idiom

$$u_1 = (\texttt{push ebp | * | mov esp,ebp})$$

describes a stack frame set-up operation. Idioms are an abstraction of the true instruction sequence, insofar as instruction details such as immediate operands and memory addresses are elided. The idiom template we use for authorship attribution describes all possible sequences of 1–3 instructions, and is intended to capture stylistic characteristics that are reflected in the order of instructions output by the compiler.

```
    cpuid
    jmp L2
    ...
L1:
    cmp ecx,edx
    jle L1
L2:
    mov eax, 0x5
    sysenter
```

(a)



(b)

Fig. 2. A code example and a corresponding graphlet. The node colors σ are determined by the instructions in each block (for example, both of the blocks represented by (●) nodes contain system instructions). Edge labels τ indicate control flow edge type (for example, τ_3 represents the jle conditional branch and τ_1 is its fall-through edge).

2.2 Graphlets

While idioms capture instruction-level details, graphlet features represent details of program structure. Graphlets are three-node subgraphs of the control flow graph that reflect the *local* structure of the program. A graphlet feature template also defines a *coloring function* $\sigma : V \to \mathcal{C}$, where $\mathcal{C}$ is the set of possible colors for a particular graphlet template. For example, the *instruction summary graphlets* we use for toolchain provenance recovery (and which we adopt here) color nodes according the various classes of instructions occurring in a basic block, as illustrated in Figure 2. We refer the reader to our previous work for details of the instruction summary coloring function and algorithms for efficient graphlet matching [16].

In the current study, graphlet features are a bridge between the instruction-level representation (using colors based on instruction classes) and the program structure (the local control flow); however, these features may miss stylistic characteristics that are visible only in high-level program structure. We could attempt to capture such characteristics by defining graphlet-like features using larger subgraphs, but there is an essential tension between the expressiveness of such features and the computational complexity of the subgraph matching problem. Instead, we introduce two additional graphlet-based features, *supergraphlets* and *call graphlets*, that are defined over transformations of the original control flow graph.

2.3 Supergraphlets

Supergraphlets are analogous to instruction summary graphlets defined over a *collapsed* control flow graph, as illustrated in Figure 3. The graph collapse operation merges each node in the graph with a random neighbor. The edge set and color of the collapsed node represent the union of the edge sets and colors of the original nodes. A three-node graphlet instantiated from the collapsed graph is thus an approximate representation of six nodes in the original CFG. This process can be repeated recursively to obtain the desired long-range structural coverage. Note that because random neighbors are selected, we do not obtain

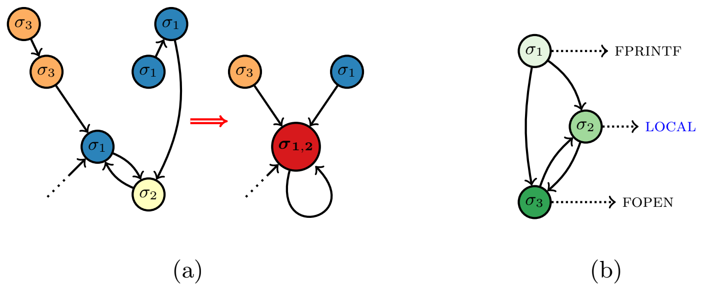


Fig. 3. Graphlet-based features of transformations of the control flow graph. Supergraphlets (a) represent control flow relationships in a graph where the neighbors of the middle three nodes have been *collapsed*; the color of one collapsed node (●) reflects the union of two nodes with different colors. Call graphlets (b) are defined over a graph reduced to blocks containing `call` instructions.

all possible supergraphlets of the original graph; in keeping with our general approach to code representation, we rely on the vast number of features to capture any authorship characteristics in the program. Selecting neighbors to collapse at random avoids systematically biasing the collapse operation towards particular control flow structure.

2.4 Call Graphlets

Recursively collapsing the control flow graph and extracting supergraphlet features only loosely approximates arbitrarily long-range program structure. Call graphlets are designed to directly capture both *interprocedural* control flow and a program's interaction with external libraries. Call graphlets are defined over a new graph G^c containing only those nodes that contain call instructions, with edges $E^c = \{(v, v') : v \rightsquigarrow v'\}$, where $\rightsquigarrow$ indicates the existence of a path in the original control flow graph. Call graphlets admit the coloring function $\sigma_c : V^c \rightarrow \{\mathcal{L}, \textsc{local}\}$, where $\mathcal{L}$ is a predefined set of *external library functions* and LOCAL is a special value meaning any internal function within the program binary. Internal functions receive a single, generic color because, unlike calls to external libraries, they are not comparable across different programs. While $\mathcal{L}$ could be restricted to a set of specific library functions, in practice we let it extend to the entire set of library routines called by programs in our corpus and rely on feature selection to eliminate irrelevant call graphlet features.

2.5 N-grams and External Interaction

To cast as wide a net as possible in our search for good authorship features, we define several more features that are relaxations of those described above. Byte *n-grams* are short strings of three or four bytes, and can be thought of as a relaxation of the idiom instruction abstractions: using the raw bytes, n-grams capture specific instruction opcodes and immediate and memory operands. *Library call*

Table 1. The number of concrete features instantiated by each feature template for a representative corpus of 1,747 C and C++ binaries comprising 27MB of code. Each template captures one or more instruction-level, control-flow, or external library interaction properties of the code.

		Code Property		
Feature	#	Instruction	Control flow	External
N-grams	391,056	✓		
Idioms	54,705	✓		
Graphlets	37,358	✓	✓	
Supergraphlets	117,997	✓	✓	
Call graphlets	8,062		✓	✓
Library calls	152			✓

features simply count the number of invocations of the set of $\mathcal{L}$ external library functions used in the call graphlet features, eliminating structural characteristics.

Table 1 summarizes our binary code feature templates and the number of each instantiated in a typical corpus. Our algorithms automatically select a subset of these features based on the training data, as we describe in the following section.

3 Author Classification

In author classification, we assume that there exists a known set of programmers of interest, and that training data are available in the form of samples of programs written by each programmer. We model program binaries as a collection of the features described in the previous section in order to discriminate between programs written by different authors. To be precise, given a known set of program authors $\mathcal{Y}$ and a set of M training programs $\mathcal{P}_1, \cdots, \mathcal{P}_M$ with author labels $y_1, \cdots, y_M$, the task of the classifier is to learn a decision function that assigns a label $y \in \mathcal{Y}$ to a new program, indicating the identity of the most likely author.

A program $\mathcal{P}_m$ is represented by a integral-valued *feature vector* $\mathbf{x}_m$ describing the features that occur in the program. Feature vectors summarize a set of *feature functions* $f \in \Phi$ that indicate the presence of that feature evaluated over a feature-specific domain in the binary. For example, the function

$$f_{\text{FPRINTF}}(\mathcal{P}_m, c_j) = \begin{cases} 1 & \text{if call site } c_j \text{ in } \mathcal{P}_m \text{ calls FPRINTF} \\ 0 & \textit{otherwise} \end{cases}$$

tests for a particular library call and is defined over the domain of *call sites* in the program; idiom feature functions

$$f_\iota(\mathcal{P}_m, a_j) = \begin{cases} 1 & \text{if idiom } \iota \text{ exists at instruction offset } a_j \text{ in } \mathcal{P}_m \\ 0 & \textit{otherwise} \end{cases}$$

are defined over the domain of instruction offsets in the binary. The feature vector $\mathbf{x}_m$ for a program counts up the $|\Phi|$ features

$$\mathbf{x}_m = \begin{pmatrix} \sum_{Dom(f_1)} f_1(\mathcal{P}_m, \cdot) \\ \sum_{Dom(f_2)} f_2(\mathcal{P}_m, \cdot) \\ \cdots \\ \sum_{Dom(f_n)} f_{|\Phi|}(\mathcal{P}_m, \cdot) \end{pmatrix}$$

evaluated at every point in the domain $Dom(f_i)$ of the particular feature.

The number of feature functions in Φ is quite large; using feature vectors that summarize all possible features would increase both training cost and the risk that the learned parameters would *overfit* the data—that is, that the resulting classifier would fail to generalize to new programs. Because our feature templates are not designed to highlight particular stylistic characteristics, we expect that many features will be of little value for authorship attribution. We therefore perform a simple form of feature selection, ranking features by the *mutual information* between the feature and the true author label. More precisely, we compute

$$I(\Phi, \mathcal{Y}) = \sum_{f \in \Phi} \sum_{y \in \mathcal{Y}} p(f, y) \log \left(\frac{p(f, y)}{p(f)p(y)} \right)$$

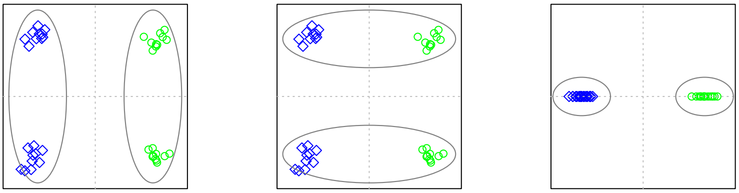
on the training set, where $p(f)$ and $p(y)$ are the empirical expectations of features and author labels, respectively, and $p(f, y)$ is the co-occurrence of these variables. Mutual information measures the decrease in uncertainty about one variable as a function of the other; features that are positively correlated with only a single programmer will score high under this criterion, while features that are distributed uniformly over programs by all authors will have low mutual information. The number of features to retain is chosen through cross-validation: we split the training data into ten *folds*, reserving one fold as a *tuning set*, then train a classifier on the remaining folds and evaluate its accuracy on the tuning set. By performing cross-validation on data represented by varying numbers of the features ranked highest by mutual information, we automatically select a subset of features that produce good authorship classifiers.

There are many different models that can be used to classify data such as ours. We use linear support vector machines (SVMs) [3], which scale well with high-dimensional data and have shown good performance in our experience with other classification tasks for binary programs. Two-class SVMs are usually formulated with labels $y \in \{-1, +1\}$, and compute a weight vector $\mathbf{w}$ that solves the following optimization problem:

$$\min_{\mathbf{w},\xi,b} \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i}^{n} \xi_i \qquad \text{s.t. } y_i(\mathbf{w}^T\mathbf{x} - b) \geq 1 - \xi_i, \;\; \xi_i \geq 0.$$

Such binary SVMs can be easily extended to the case of K classes by training K different binary classifiers with weight vectors $\mathbf{w}_1, \cdots, \mathbf{w}_K$; the classifier assigns a new example the label $k \in [1, K]$ that leads to the largest margin, i.e.

$$\operatorname*{argmax}_k \mathbf{w}_k^T \mathbf{x}.$$



(a) (b) (c)

Fig. 4. The hazards of unsupervised clustering. Assuming that the data belong to true classes y_1 (◇) and y_2 (○) and two clusters are formed, the correct cluster partition (a) is no more likely than the alternative (b). Using the distance metric $\left(\begin{smallmatrix}1 & 0\\ 0 & 0\end{smallmatrix}\right)$ is equivalent to transforming the data as in (c), where the clustering decision is unambiguous.

We use the LIBLINEAR support vector machine implementation [4] for authorship classification. We scale the feature vectors to the interval $[0, 1]$; scaling prevents frequently occurring features from drowning the contribution of rarer ones, while preserving the sparsity of the feature vectors. In our evaluation section, we examine the contribution of each feature template to overall classifier performance.

4 Author Clustering

Clustering is an unsupervised learning technique that groups data by similarity. For authorship attribution, clustering corresponds to the task of finding stylistically similar programs without assuming particular authors are present. In many ways, clustering is harder than classification: without training data, it is generally not possible to tell whether particular features are more or less useful for relating the data, which leads to the possibility that clustering algorithms will arrive at clusters that reflect a different property than what was desired. This issue is particularly challenging for authorship clustering, where we have a large number of features and no assurance that they reflect only programmer style and not, for example, program functionality.

One way to encourage the formation of authorship clusters is to transform the feature space such that stylistically similar programs are closer to one another; equivalently, we can define a $d \times d$ *distance metric* A such that the *Mahalanobis distance* [12] between two feature vectors $\mathbf{x}_a, \mathbf{x}_b$ in $\mathbb{R}^d$ is

$$D_A(\mathbf{x}_a, \mathbf{x}_b) = \sqrt{(\mathbf{x}_a - \mathbf{x}_b)^T A (\mathbf{x}_a - \mathbf{x}_b)}.$$

If a particular metric can be found such that stylistically similar programs are close under that metric, then clustering techniques will do better at forming authorship clusters. Figure 4 illustrates this solution with a simple example.

We observe that stylistic features, if they are general, can be learned from *any* set of authors; although the programs to be clustered may have no training data, we can derive a metric from a different collection of programs with author labels. More precisely, consider two sets of programs $\{\mathcal{P}_1, \cdots, \mathcal{P}_\ell\}$ and $\{\mathcal{P}_{\ell+1}, \cdots, \mathcal{P}_{\ell+u}\}$, with known author labels $\{y_1, \cdots, y_\ell\}$; the authors for the unlabeled programs may or may not coincide with those of the labeled programs. Both sets of programs are represented using the feature vectors we describe in the previous section. We define a two part algorithm for transferring stylistic knowledge from the labeled data to the unlabeled data:

1. Learn a metric A over ℓ labeled programs $\mathcal{P}_1, \cdots, \mathcal{P}_\ell$ such that the distance in the feature space between two programs with the same author is always less than the distance between two programs with different authors.
2. Cluster u unlabeled programs $\mathcal{P}_{\ell+1}, \cdots, \mathcal{P}_{\ell+u}$ using the distance function D_A.

We use the *large margin nearest neighbors* (LMNN) algorithm [20] to learn the style metric. LMNN learns the metric by optimizing the margin for nearby programs in the feature space, making it complementary to the k-means algorithm we use for clustering.

5 Evaluation

We investigate several aspects of authorship attribution in our evaluation: (1) the extent to which our techniques recover author style in program binaries, (2) the trade-offs involved in imprecise classification (i.e., tolerating some *false positives*), and (3) whether and how much stylistic clustering of one set of programs can be improved by using information derived from another, unrelated set. Our evaluation shows that:

– The binary code features we introduce effectively capture programmer style. Our classifier achieves accuracies of 81% for ten distinct authors (10% accuracy is expected for labels selected by random chance) and 51% when discriminating among almost 200 authors (0.5% for random chance). These results show that a strong author style signal survives the compilation process.

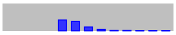
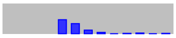
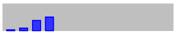
– The authorship classifier offers practical attribution with good accuracy, if a few false positives can be tolerated. The correct author is among the top five 95% of the time for a data set of 20 authors, and 81% of the time when 100 authors are represented.

– Stylistic knowledge derived from supervised authorship classification can be transferred to authorship clustering, improving cluster quality. The cluster assignments improve by 20% when clustering uses a stylistic metric.

5.1 Methodology

We obtain training and evaluation programs from the Google Code Jam programming competition and from an undergraduate operating systems course at

Table 2. Corpora used for model training and evaluation. Each binary is the implementation by a particular author of one of the program types for a given corpus.

Corpus	Authors	Program Types	Binaries	Prog./Author Dist.
				4 16
Code Jam 2010	191	23	1,747	
Code Jam 2009	93	22	834	
CS537 Fall 2009	32	7	203	

the University of Wisconsin (CS537). These data sets have author labels for each program, which can be challenging to obtain for other data sources like open source projects. They are also *parallel* corpora: each data set contains implementations by different authors of a small number of *program types* representing particular functionality (i.e., contest solutions for Code Jam, and programming projects for CS537). Parallel corpora allow us to control for confounding variables like program functionality during evaluation. Table 2 summarizes our data sets.

To create a data representation suitable for learning and evaluation, we process the binaries in each corpus with the ParseAPI parsing library to obtain control flow graphs and the underlying instructions. We eliminate statically linked library functions and other known binary code snippets that are unrelated to the program author.[2] We then exhaustively enumerate all of the features described in Section 2, using the occurrence of these features along with the known authorship labels to compute the mutual information for each feature. We select a subset of features using the cross-validation procedure described in Section 3. We use the top 1,900 features for modeling and evaluation of the Code Jam data; 1,700 features are used for CS537.

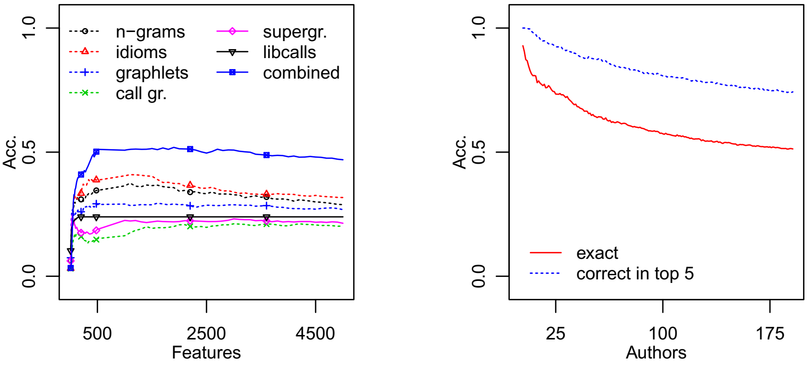
Our evaluation methodology involves both standard ten-fold cross-validation and random subset testing, depending on the experiment:

- For classification of the entire data set (e.g. 191-way classification for the Code Jam 2010 data), we use ten-fold cross-validation.
- When evaluating how classification accuracy behaves as a function of the number of authors represented in the data, we randomly draw a subset $\mathcal{Y}_s \subseteq \mathcal{Y}$ of authors and use their programs in the test. We cannot test all possible combinations of $|\mathcal{Y}_s|$ authors; instead, we repeat the test 20 times and expect relatively high variance for small sets of authors. We approach the clustering evaluation similarly.

5.2 Classification

We evaluate authorship classification to determine (1) how much each feature template contributes to attribution, and (2) how accurately the identity of a

[2] Our data preparation procedure is fully described in our supplementary materials at http://pages.cs.wisc.edu/~nater/esorics-supp/



(a) (b)

Fig. 5. Evaluation of authorship classification on the Code Jam 2010 data set. In Figure (a) we show cross-validation accuracy over all 191 authors for classifiers trained using individual feature templates, as well as the combined classifier. Figure (b) depicts accuracy using the best combination of features as the true number of authors in the data set is increased for both the exact (—) and relaxed (···) evaluations.

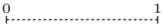
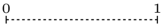
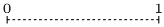
particular author can be inferred using a model based on our feature templates. For the former question, our experience led us to expect that simple feature templates that instantiate large numbers of features (e.g., idioms) would be more useful in authorship modeling. For the latter question, we hypothesized that discriminating among authors would become increasingly difficult with larger author populations.

Figure 5a depicts the cross-validation accuracy of models trained with varying numbers of the best features (by mutual information) derived from each template. Our intuition is borne out by these results: the individual contributions of simple idiom and n-gram features exceed those of the other templates. The best classifier uses a combination of all of the feature templates, achieving 51% accuracy on the full Code Jam data set.

Experiments confirm our hypothesis that author classification becomes harder for larger populations. Figure 5b depicts classifier performance as a function of the number of authors included in a subset of the data; classifier accuracy decreases as the author population size grows. In cases where precise author identification is infeasible, predicting a small set of likely authors can help to focus further investigation and analysis. In Figure 5b, this relaxed accuracy measure is plotted for a classifier that returns the top five most likely authors.

Table 3 lists exact and relaxed cross-validation accuracy for authorship classification on each corpus. The CS537 data present a significantly harder challenge for authorship attribution, due to two factors. First, there are fewer programs per author (4–7) than in the other data sets (8–16), making this a fundamentally harder learning problem. More importantly, the programs in this data set do not reflect only the work of individual programmers; students in the course were often provided with substantial amounts of partially implemented skeleton code, and also worked closely with the course professor follow an often rigid

Table 3. Classification results averaged over 20 randomly selected subsets of 20 authors

	Code Jam 2009		Code Jam 2010		CS537	
	Acc.	spread	Acc.	spread	Acc.	spread
						
Exact	.778		.768		.384	
Top 5	.947		.937		.843	

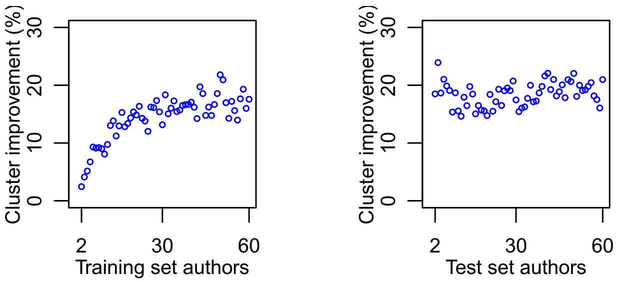
specification at the sub-module level. Despite these challenges, our attribution techniques recover significant stylistic characteristics in this data set.

5.3 Clustering

We evaluated authorship clustering to determine (1) how well the clusters reflect the ground truth program authorship, and (2) whether stylistic characteristics learned from one set of authors can improve the clustering of programs written by different authors (i.e., how well stylistic knowledge generalizes). Unlike classifiers, clustering algorithms have no notion of candidate labels, so cluster assignments are evaluated against the ground truth authors with measures based on cluster *agreement*: whether (a) programs by the same author are assigned to the same cluster, and (b) programs by different authors are assigned to different clusters. We computed several common measures of cluster agreement, including Adjusted Mutual Information (AMI), Normalized Mutual Information (NMI), and the Adjusted Rand Index (ARI); we prefer AMI because it is stable across different numbers of clusters, easing comparison of different data sets [19]. All of the measures we use take values in the range $[0, 1]$, where higher scores indicate better cluster agreement.

We performed several experiments to evaluate authorship clustering:

1. We randomly selected N authors from the Code Jam 2010 corpus and used LMNN to learn a distance metric over the feature space. We then randomly selected 30 different authors and clustered their programs using k-means with and without transforming the data with the learned metric. Since there are multiple sources of randomness in this experiment (both in selecting the data sets and in the k-means clustering algorithm), we repeated the experiment 20 times and computed the average AMI. Figure 6a depicts clustering improvement over the un-transformed data as a function of N. As expected, using more training authors to compute a metric leads a greater improvement. We conclude that stylistic information derived from one set of authors can be transferred to improve clustering of programs written by a different set of authors.
2. We performed a similar set of experiments with the number of authors used to compute the metric fixed at 30 to evaluate whether the clustering improvement is affected by the number of test set authors. Figure 6b shows that that the improvement due to incorporation of the stylistic metric is nearly invariant for a range of test set sizes.



(a)

(b)

Fig. 6. Clustering with metric learning. The improvement over the original clustering $(\text{AMI}_{metric} - \text{AMI}_{orig.})/\text{AMI}_{orig.}$ is illustrated as a function of the number of training authors (a) and the true number of testing authors (b).

Table 4. Cluster evaluation measures for 10 test authors, using metrics learned from 30 different authors

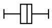
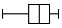
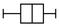
	AMI	AMI spread	NMI	spread	ARI	spread
		0 1		0 1		0 1
no transformation	.510		.637		.406	
learned metric	.606		.723		.480	

Table 4 compares the results of clustering 10 authors' programs with and without metric transformation. The cluster quality measures we compute are highly variable, due to the random nature of training and test set selection and the inherent randomness in the clustering algorithm; nonetheless, the improvement offered by the learned metric is significant at a 95% confidence level for all measures.

6 Discussion

Our evaluation shows that programmer style is preserved in program binaries, and can be recovered using techniques that automatically select stylistic code features with which to model program authorship. The SVM-based classifier we introduce can identify the correct author out of tens of candidates with good accuracy, though discriminating among a large number of authors is likely to be more limited. Nonetheless, we argue that our techniques offer a practical solution to program author identification: when discriminating among programs written by 100 authors, the correct author is ranked among the top five most likely 81% of the time, reducing the number of candidates by 95%. Moreover, our evaluation of unsupervised author clustering using stylistic metrics derived from the classification problem shows that programs can be effectively clustered by programmer style even when no training data are available for the authors in question.

The conclusions we draw are subject to limitations inherent in empirical studies. In particular, threats to internal validity apply to our claim that our techniques isolate programmer style, rather than some other program property like program functionality. We addressed this issue by using a parallel corpus, where each author implemented the same programs; the fact that our authorship classifier is able to learn to recognize an author's programs despite differing functionality mitigate this threat. Our domain transfer results for authorship clustering provide further evidence that our techniques recover programmer style.

In this study, we assume that a program has a single author. This assumption may be violated in many scenarios, such as when programmers collaborate or when programs are assembled from commodity components. The binary code representation we use is not inherently restricted to representing the program as a single unity; our features could just as easily describe individual compilation units, functions, or arbitrary sequences of binary code, for example using the sequential model we have previously used to recover program provenance [15, 16]. The extension of authorship attribution to multiple authors and a sub-program model is an open question, and is the focus of our ongoing research.

The nature of the features underlying our authorship models suggests several additional directions for future research. Our use of many simple, uninformed binary code features provides much of the power of our approach, but makes understanding the resulting models difficult: it is not clear how to map from instruction idioms and control flow graphlets to conceptual notions of high-level programmer style (the appendix illustrates this difficulty with some example features). A related concern is whether malicious agents can obfuscate programs in such a way that our techniques become ineffective. The former question informs the latter: understanding how binary code features map to source-level constructs could facilitate techniques to obscure stylistic variations.

7 Related Work

Previous work on program authorship attribution has focused almost exclusively on source code-level attribution. The use of code metrics like variable naming conventions, comment style, and program organization has been proposed several times [5, 18]; Krsul and Spafford [10] show the feasibility of this approach in a small pilot study. More recently, Hayes and Offutt [6] found further evidence that programmers can be distinguished through aggregate textual characteristics like average use of particular operators, placement of semicolons, and comment length.

Structural malware classification and behavioral clustering share many challenges with authorship attribution, as all three techniques involve extracting salient characteristics from binary code. The instruction-level features we use are similar to those used in malware classification [2, 8, 9], particularly n-grams; our idiom features differ from features based on instruction sequences through the use of wildcards and the abstraction of low-level details like the opcode and immediate values. The instruction summary colors we use in the graphlet features are inspired by a technique to identify polymorphic malware variants [11].

Although some of the binary code representations we use are similar to existing work, our techniques are largely orthogonal: malware classification seeks to extract characteristics specific to a program or a family of programs with related behavior, while our authorship attribution techniques must discover more general properties of author style.

Authorship falls into the broad category of *program provenance*: those details that characterize the process through which the program was produced. Our previous investigation of *toolchain provenance* [15, 16] heavily informs this work, providing a general framework for extracting the characteristics of program binaries as well as providing the base representations on which we build more sophisticated authorship features. The current paper investigates a higher level of the *provenance hierarchy*, moving beyond those program properties that are attributable to the production toolchain.

8 Conclusion

We have presented techniques to extract stylistic characteristics from program binaries to perform authorship attribution and to cluster programs according to programmer style. Our authorship attribution techniques identify the correct author out of a set of 20 candidates with 77% accuracy, and rank the correct author among the top five 94% of the time. These techniques enable analysts to determine, for example, whether a new program sample is likely to have been written by a person of interest, or to test for the existence of multiple, stylistically dissimilar authors in a collection of programs. Framing authorship attribution and clustering as machine learning problems, we designed instruction- and structure-based representations of binary code that automatically capture binary code details that reflect programmer style. We developed program clustering techniques that transfer stylistic knowledge across different domains, assigning new programs to clusters based on stylistic similarity with no training data. The results of our evaluation strongly support our claim that programmer style is preserved through the compilation process, and can be recovered from characteristics of the code in program binaries. Our approach to discovering stylistic features builds on our previous research into recovering toolchain provenance, and is part of a general framework for information retrieval in program binaries, with applications in security and software forensics.

Acknowledgments. We thank the anonymous reviewers for their insightful comments and suggestions. This research funded in part by Department of Homeland Security grant FA8750-10-2-0030 (funded through AFRL), National Science Foundation grants CNS-0716460 OCI-1032341, and Department of Energy grants DE-SC0004061 and DE-SC0002154.

References

[1] Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)

[2] Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E.: Semantics-aware malware detection. In: IEEE Symposium on Security and Privacy (S&P 2005), Oakland, CA (May 2005)

[3] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20 (1995)

[4] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)

[5] Gray, A., Sallis, P., MacDonell, S.: Software forensics: Extending authorship analysis techniques to computer programs. In: 3rd Biennial Conference of the International Association of Forensic Linguists, Durham, NC (September 1997)

[6] Hayes, J.H., Offutt, J.: Recognizing authors: an examination of the consistent programmer hypothesis. Software Testing, Verification and Reliability (2009)

[7] Juola, P.: Authorship attribution. Foundations and Trends in Information Retrieval (December 2006)

[8] Karim, M., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. Journal in Computer Virology 1, 13–23 (2005)

[9] Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 7, 2721–2744 (2006)

[10] Krsul, I., Spafford, E.H.: Authorship analysis: identifying the author of a program. Computers & Security 16(3), 233–257 (1997)

[11] Krügel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006)

[12] Mahalanobis, P.C.: On the generalised distance in statistics. In: Proceedings National Institute of Sciences of India, vol. 2 (1936)

[13] Palmer, G.: A road map for digital forensic research. Technical Report DTR-T001-01 FINAL, Digital Forensics Research Workshop, DFRWS (2001)

[14] Paradyn Project. ParseAPI: An application program interface for binary parsing (2011), http://paradyn.org/html/parse0.9-features.html

[15] Rosenblum, N.E., Miller, B.P., Zhu, X.: Extracting compiler provenance from program binaries. In: 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2010), Toronto, Ontario, Canada (June 2010)

[16] Rosenblum, N.E., Miller, B.P., Zhu, X.: Recovering the toolchain provenance of binary code (2011). In: 20th International Symposium on Software Testing and Analysis (ISSTA), Toronto, Ontario, Canada (July 2011)

[17] Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: ACM SIGMOD International Conference on Management of Data, San Diego, CA (June 2003)

[18] Spafford, E.H., Weeber, S.A.: Software forensics: Can we track code to its authors? Technical Report CSD-TR-92-010, Purdue University (February 1992)

[19] Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada (June 2009)

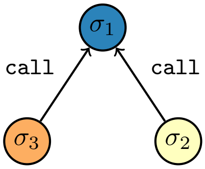
[20] Weinberger, K.Q., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. Journal of Machine Learning Research (February 2009)

A Stylistic Feature Examples

The binary code features we describe in this paper are a powerful tool for capturing characteristics of programmer style, but can be difficult to interpret. To illustrate this difficulty, consider the following idiom features, ranked by the magnitude of their weights w in an SVM-based classifier that discriminates among ten different program authors (these idioms are among the top twenty features by magnitude):

<table>
<tr><th>Idiom</th><th>w</th></tr>
<tr><td><code>push rbx | * | lea rax, [mem]</code></td><td>0.213638</td></tr>
<tr><td><code>push rbx | sub rsp, [imm] | lea rax, [mem]</code></td><td>0.213638</td></tr>
<tr><td><code>sub rsp, [imm] | * | mov [mem], rsi</code></td><td>0.180291</td></tr>
<tr><td><code>mov rdi, rax | * | mov rax, [imm]</code></td><td>0.179203</td></tr>
<tr><td><code>call rip | mov xmm, rax | mov rdi, rax</code></td><td>0.173018</td></tr>
<tr><td><code>jnz rip | mov xmm, rax | mov rdi, rax</code></td><td>0.165779</td></tr>
<tr><td><code>mov rax, [imm] | call rip | mov xmm, rax</code></td><td>0.153061</td></tr>
<tr><td><code>sub rsp, [imm] | * | mov rsi, [imm]</code></td><td>0.148890</td></tr>
<tr><td><code>mov xmm, eax | * | mov rdi, [imm]</code></td><td>-0.133057</td></tr>
<tr><td><code>* | sub rsp, [imm] | mov [mem], [imm]</code></td><td>0.131137</td></tr>
</table>

While some patterns emerge (IP-relative branching and calls appear to be a telling characteristic), it is difficult to translate from simple instruction patterns to a meaningful, high-level understanding of programming style; our models require hundreds or thousands of these features to capture program authorship. Other highly-weighted features offer similar barriers to interpretation; there are several three- and four-byte N-grams among the most discriminative features in this model, and the highest-weighted feature is the instruction summary graphlet



where $\sigma_1 = \{\texttt{jump}\}$, $\sigma_2 = \{\texttt{call,lea,mov}\}$, and $\sigma_3 = \{\texttt{arith}, \texttt{call}, \texttt{mov}, \texttt{stack}\}$, which is hardly sufficient to reach any conclusion about programming style.

The contribution of some features is easier to judge, for example the external library features (Section 2.5). The use of `sprintf` and several of the C++ `iostream` operators are heavily weighted (within the top ten), suggesting that programmer preference for particular APIs or standard library functions may be a good indicator of style. Nonetheless, this feature type alone is insufficient (see Section 5.2) for authorship attribution and extracting high-level interpretations of programmer style from low-level code features is an open problem.

Secure and Efficient Protocols for Iris and Fingerprint Identification

Marina Blanton[1] and Paolo Gasti[2]

[1] Department of Computer Science and Engineering, University of Notre Dame
[2] Department of Information and Computer Science, University of California, Irvine

Abstract. Recent advances in biometric recognition and the increasing use of biometric data prompt significant privacy challenges associated with the possible misuse, loss, or theft of biometric data. Biometric matching is often performed by two mutually distrustful parties, one of which holds one biometric image while the other owns a possibly large biometric collection. Due to privacy and liability considerations, neither party is willing to share its data. This gives rise to the need to develop secure computation techniques over biometric data where no information is revealed to the parties except the outcome of the comparison or search. To address the problem, in this work we develop and implement the first privacy-preserving identification protocol for iris codes. We also design and implement a secure protocol for fingerprint identification based on FingerCodes with a substantial improvement in the performance compared to existing solutions. We show that new techniques and optimizations employed in this work allow us to achieve particularly efficient protocols suitable for large data sets and obtain notable performance gain compared to the state-of-the-art prior work.

1 Introduction

Recent advances in biometric recognition make the use of biometric data more prevalent for authentication and other purposes. Today large-scale collections of biometric data include face, fingerprint, and iris images collected by the US Department of Homeland Security (DHS) from visitors through its US-VISIT program [21], iris images collected by the United Arab Emirates (UAE) Ministry of Interior from all foreigners and also fingerprints and photographs from certain types of travelers [23], and several others. While biometry serves as an excellent mechanism for authentication and identification of individuals, such data is undeniably extremely sensitive and must be well protected. Furthermore, once leaked biometric data cannot be revoked or replaced. For these reasons, biometric data cannot be easily shared between organizations or agencies. However, there could be legitimate reasons to carry out computations on biometric data belonging to different entities. For example, a non-government agency may need to know whether a biometric it possesses appears on the government watch-list. In this case the agency would like to maintain the privacy of the individual if no matches are found, and the government also does not want to release its database to third parties.

The above requires carrying out computation over biometric data in a way that keeps the data private and reveals only the outcome of the computation. In particular, we study

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 190–209, 2011.

the problem of *biometric identification*, where a client C is in a possession of a biometric X and a server S possesses a biometric database D. The client would like to know whether X appears in the database D by comparing its biometric to the records in D. The computation amounts to comparing X to each $Y \in D$ in a privacy-preserving manner. This formulation is general enough to apply to a number of other scenarios, ranging from a single comparison of X and Y to the case where two parties need to compute the intersection of their respective databases. We assume that the result of comparing biometrics X and Y is a bit, and no additional information about X or Y should be learned by the parties as a result of secure computation. With our secure protocols, the outcome can be made available to either party or both of them; for concreteness in our description, we have the client learn the outcome of each comparison.

In this work we assume that both the client's and the server's biometric images have been processed and have representations suitable for biometric matching, i.e., each raw biometric image has been processed by a feature extraction algorithm. For the types of biometric considered in this work, this can be performed for each image independently and we do not discuss this further.

Prior Work. Literature on secure multi-party computation is extensive. Starting from the seminal work on garbled circuit evaluation [39], it has been known that any function can be securely evaluated by representing it as a boolean circuit. Similar results are also known for securely evaluating any function using secret sharing techniques (e.g., [36]) or homomorphic encryption (e.g., [11]). In the last several years a number of tools have been developed for automatically creating a secure protocol from a function description written in a high-level language. Examples include Fairplay [30], VIFF [14], TASTY [18], and others. It is, however, well-known that custom optimized protocols are often constructed for specific applications due to the inefficiency of generation solution. Such custom solutions are known for a wide range of application (e.g., set operations [29,17], DNA matching [38], k-means clustering [9], etc.), and this work focuses on secure biometric identification using iris codes and fingerprints. Furthermore, some of the optimizations employed in this work can find their uses in protocol design for other applications, as well as general compilers and tools such as TASTY [18].

With the growing prevalence of applications that use biometrics, the need for secure biometric identification was recognized in the research community. A number of recent publications address the problem of privacy-preserving face recognition [16,37,33]. This problem was first treated by Erkin et al. [16], where the authors designed a privacy-preserving face recognition protocol based on the Eigenfaces algorithm. The performance of that solution was consequently improved by Sadeghi et al. [37]. More recently, Osadchy et al. [33] designed a new face recognition algorithm together with its privacy-preserving realization called SCiFI. The design targeted to simultaneously address robustness to different viewing conditions and efficiency when used for secure computation. As a result, SCiFI is currently recognized as the best face identification algorithm with efficient privacy-preserving realization. SCiFI takes 0.31 sec (during the online phase) [33] to compare two biometrics, and therefore would take about 99 sec to compare a biometric to a database of 320 images (which is the database size used in the experiments in several prior publications).

Another very recent work by Barni et al. [3] designs a privacy-preserving protocol for fingerprint identification using FingerCodes [25]. FingerCodes use texture information from a fingerprint to compare two biometrics. The algorithm is not as discriminative as fingerprint matching techniques based on location of minutiae points, but it was chosen by the authors as particularly suited for efficient realization in the privacy-preserving framework. As of the time of this writing, similar results for other types of biometrics or other fingerprint matching techniques are not available in the literature. We narrow this gap by providing a secure two-party protocol for widely used iris identification, as well as address fingerprint identification. Our protocols follow the standard algorithms for comparing two biometrics, yet they are very efficient and outperform the state-of-the-art protocols with a notable reduction in the overhead.

Bringer et al. [8] describe a biometric-based authentication mechanism with privacy protection of biometric, where the Hamming distance is used as the distance metric. The authentication server is composed of three entities that must not collude, and one of them, the matcher, learns the computed Hamming distance. In our work, however, no information beyond the outcome of the comparison is revealed, the computation itself is more complex and corresponds to the actual algorithm used for iris code comparisons, and there is no need for additional or third-party entities. Barbosa et al. [2] extend the framework with a classifier to improve authentication accuracy and propose an instantiation based on Support Vector Machine using homomorphic encryption.

Our Contributions. In this work we treat the problem of privacy preserving biometric identification. We develop new secure protocols for two types of biometric, iris and fingerprints, and achieve security against semi-honest adversaries. While iris codes are normally represented as binary strings and use very similar matching algorithms, there is a variety of representations and comparison algorithms for fingerprints. In this paper, we study FingerCodes that use fixed-size representations and an efficient comparison algorithm.[1] Our protocols were designed with efficiency in mind to permit their use on relatively large databases, and possibly in real time. While direct performance comparison of our protocols and the results available in the literature is possible only in the case of FingerCode, we can use complexity of the computation to draw certain conclusions. The results we achieve in this work are as follows:

1. Our secure FingerCode protocol is extremely fast and allows the parties to compare two fingerprints X and Y using a small fraction of a second. For a database of 320 biometrics, the online computation can be carried out in 0.45 sec with the communication of 279KB. This is an over 30-fold improvement in both communication and computation over the privacy-preserving solution of [3], as detailed in Section 5, and a significant improvement over an adaptation of [37] to this context.
2. Iris codes use significantly longer representations (thousands of bits) and require more complex transformation of the data. Despite the length and complexity, our solution allows two iris codes to be compared in 0.15 sec. With respect to the state-of-the-art face recognition protocol SCiFI, which also relies on Hamming distance

[1] We also construct a secure protocol for minutiae-based fingerprint comparisons, but its description and implementation appear in the full version of this work [7] due to space constraints.

computation, our protocol achieves lower overhead despite the fact that the computation involves an order of magnitude larger number of more complex operations.

2 Description of Computation

In what follows, we assume that client C holds a single biometric X and server S holds a database of biometrics D. The goal is to learn whether C's biometric appears in S's database without learning any additional information. This is accomplished by comparing X to each biometric $Y \in D$, and as a result of each comparison C learns a bit that indicates whether the comparison resulted in a match.

Iris. Let an iris biometric X be represented as an m-bit binary string. We use X_i to denote i-th bit of X. In iris-based recognition, after feature extraction, biometric matching is normally performed by computing the Hamming distance between two biometric representations. Furthermore, the feature extraction process is such that some bits of the extracted string X are unreliable and are ignored in the matching process. Information about such bits is stored in an additional m-bit string, called *mask*, where its i-th bit is set to 1 if the i-th bit of X should be used in the matching process and is set to 0 otherwise. For biometric X, we use $M(X)$ to denote the mask associated with X. Often, a predetermined number of bits (e.g., 25% in [20] and 35% in [4]) is considered unreliable in each biometric template. Thus, to compare two biometric representations X and Y, their Hamming distance takes into account the respective masks. That is, if the Hamming distance between two iris codes without masks is computed as: $HD(X, Y) = (||X \oplus Y||)/m = (\sum_{i=1}^{m} X_i \oplus Y_i)/m$, the computation of the Hamming distance that uses masks becomes [15]:

$$HD(X, M(X), Y, M(Y)) = \frac{||(X \oplus Y) \cap M(X) \cap M(Y)||}{||M(X) \cap M(Y)||} \tag{1}$$

In other words, we have $HD(X, M(X), Y, M(Y)) = \frac{\sum_{i=1}^{m}((X_i \oplus Y_i) \wedge M(X_i) \wedge M(Y_i))}{\sum_{i=1}^{m}(M(X_i) \wedge M(Y_i))}$. Throughout this work, we assume that the latter formula is used and simplify the notation to $HD(X, Y)$. Then the computed Hamming distance is compared with a specific threshold T, and the biometrics X and Y are considered to be a match if the distance is below the threshold, and a mismatch otherwise. The threshold T is chosen based on the distributions of authentic and impostor data. (In the likely case of overlap of the two distributions, the threshold is set to achieve the desired levels of false accept and false reject rates based on the security goals.)

Two iris representations can be slightly misaligned. This problem is caused by head tilt during image acquisition. To account for this, the matching process attempts to compensate for the error and rotates the biometric representation by a fixed amount to determine the lowest distance. Each biometric is represented as a two-dimensional array, therefore a circular shift is applied to each row by shifting its representation by a small fixed number of times, which we denote by c. The minimum Hamming distance across all runs is then compared to the threshold. That is, if we let $\mathsf{LS}^j(\cdot)$ (resp., $\mathsf{RS}^j(\cdot)$) denote a circular left (resp., right) shift of the argument by a fixed number of bits (2 bits in experiments conducted by the biometrics group at our institution, where application of the Gabor filter during feature extraction results in a complex number, which is quantized into a 2-bit value), the matching process becomes:

$$\begin{aligned}\min(HD(X, \mathsf{LS}^c(Y)), \ldots, HD(X, \mathsf{LS}^1(Y)), HD(X, Y),\\ HD(X, \mathsf{RS}^1(Y)), \ldots, HD(X, \mathsf{RS}^c(Y))) \overset{?}{<} T\end{aligned} \tag{2}$$

Throughout this work we assume that the algorithms for comparing two biometrics are public, as well as any constant parameters such as T. Our protocols, however, maintain their security and performance guarantees if the (fixed) thresholds are known only to the server who owns the database.

Fingerprints. Work on fingerprint identification dates many years back with a number of different approaches currently available (see, e.g., [31] for an overview). The most popular and widely used techniques extract information about minutiae from a fingerprint and store that information as a set of points in the two-dimensional plane. Fingerprint matching can also be performed using a different type of information extracted from a fingerprint image. One example is FingerCode [25] which uses texture information from a fingerprint scan to form fingerprint representation X. While FingerCodes are not as distinctive as minutiae-based representations and are best suited for use in combination with minutiae to improve the overall matching accuracy [31], FingerCode-based identification can be implemented very efficiently in a privacy-preserving protocol. In particular, each FingerCode consists of a fixed number m elements of ℓ bits each. Then FingerCodes $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$ are considered a match if the Euclidean distance between their elements is below the threshold T:

$$\sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \overset{?}{<} T \tag{3}$$

Barni et al. [3] was the first to provide a privacy-preserving protocol for FingerCode-based biometric identification. We show that the techniques employed in this work improve both computation and communication of the protocol of [3] by a large factor.

3 Preliminaries

Security Model. We use the standard security model for secure two-party computation in presence of semi-honest participants (also known as honest-but-curious or passive). In particular, it means that the parties follow the prescribed behavior, but might try to compute additional information from the information obtained during protocol execution. Security in this setting is defined using simulation argument: the protocol is secure if the view of protocol execution for each party is computationally indistinguishable from the view simulated using that party's input and output only. This means that the protocol execution does not reveal any additional information to the participants. The definition below formalizes the notion of security for semi-honest participants:

Definition 1. *Let parties* P_1 *and* P_2 *engage in a protocol* π *that computes function* $f(\mathsf{in}_1, \mathsf{in}_2) = (\mathsf{out}_1, \mathsf{out}_2)$*, where* in_i *and* out_i *denote input and output of party* P_i*, respectively. Let* $\mathrm{VIEW}_\pi(P_i)$ *denote the view of participant* P_i *during the execution* *of protocol π. More precisely, P_i's view is formed by its input, internal random coin tosses r_i, and messages $m_1, \ldots, m_t$ passed between the parties during protocol execution, i.e.,* $\mathrm{VIEW}_\pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_t)$. *We say that protocol π is secure against semi-honest adversaries if for each party P_i there exists a probabilistic polynomial time simulator S_i such that* $\{S_i(\mathsf{in}_i, f(\mathsf{in}_1, \mathsf{in}_2))\} \equiv \{\mathrm{VIEW}_\pi(P_i), \mathsf{out}_i\}$, *where "$\equiv$" denotes computational indistinguishability.*

Homomorphic Encryption. Our constructions use a semantically secure additively homomorphic encryption scheme. In an additively homomorphic encryption scheme, $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$ which also implies that $\mathsf{Enc}(m)^a = \mathsf{Enc}(a \cdot m)$. While any encryption scheme with the above properties (such as the well known Paillier encryption scheme [34]) suffices for the purposes of this work, the construction due to Damgård et al. [13,12] (DGK) is of particular interest here. We also note that in Paillier encryption scheme, a public key consists of a k-bit RSA modulus $N = pq$, where p and q are prime, and an element g whose order is a multiple of N in $\mathbb{Z}^*_{N^2}$. Given a message $m \in \mathbb{Z}_N$, encryption is performed as $\mathsf{Enc}(m) = g^m r^n \bmod N^2$, where $r \overset{R}{\leftarrow} \mathbb{Z}_N$ and notation $a \overset{R}{\leftarrow} A$ means that a is chosen uniformly at random from the set A. In DGK encryption scheme [13,12], which was designed to work with small plaintext spaces and has shorter ciphertext size than other randomized encryption schemes, a public key consists of (i) a (small, possibly prime) integer u that defines the plaintext space, (ii) k-bit RSA modulus $N = pq$ such that p and q are $k/2$-bit primes, v_p and v_q are t-bit primes, and $uv_p|(p-1)$ and $uv_q|(q-1)$, and (iii) elements $g, h \in \mathbb{Z}^*_N$ such that g has order uv_pv_q and h has order v_pv_q. Given a message $m \in \mathbb{Z}_u$, encryption is performed as $\mathsf{Enc}(m) = g^m h^r \bmod N$, where $r \overset{R}{\leftarrow} \{0,1\}^{2.5t}$. We refer the reader to the original publications [34] and [13,12], respectively, for any additional information.

Garbled Circuit Evaluation. Originated in Yao's work [39], garbled circuit evaluation allows two parties to securely evaluate any function represented as a boolean circuit. The basic idea is that, given a circuit composed of gates, one party P_1 creates a garbled circuit by assigning to each wire two randomly chosen keys. P_1 also encodes gate information in a way that given keys corresponding to the input wires (encoding specific inputs), the key corresponding to the output of the gate on those inputs can be recovered. The second party, P_2, evaluates the circuit using keys corresponding to inputs of both P_1 and P_2 (without learning anything in the process). At the end, the result of the computation can be recovered by linking the output keys to the bits which they encode.

Recent literature provides optimizations that reduce computation and communication overhead associated with circuit construction and evaluation. Kolesnikov and Schneider [27] describe an optimization that permits XOR gates to be evaluated for free, i.e., there is no communication overhead associated with such gates and their evaluation does no involve cryptographic functions. This optimization is possible when the hash function used for creating garbled gates can be assumed to be correlation robust (see [28,27] for more detail). Under the same assumptions, Pinkas et al. [35] additionally give a mechanism for reducing communication complexity of binary gates by 25%: now each gate can be specified by encoding only three outcomes of the gate instead of all four. Finally, Kolesnikov et al. [26] improve the complexity of certain commonly used operations such as addition, multiplication, comparison, etc. by reducing the number of non-XOR gates: adding two n-bit integers requires $5n$ gates, n of which are non-XOR gates; comparing two n-bit integers requires $4n$ gates, n of which are non-XOR gates; and computing the minimum of t n-bit integers (without the location of the minimum value) requires $7n(t-1)$ gates, $2n(t-1)$ of which are non-XOR gates.

With the above techniques, evaluating a non-XOR gates involves one invocation of the hash function (which is assumed to be correlation robust). During garbled circuit evaluation, P_2 directly obtains keys corresponding to P_1's inputs from P_1 and engages in the oblivious transfer (OT) protocol to obtain keys corresponding to P_2's inputs.

Oblivious Transfer. In 1-out-of-2 Oblivious Transfer, OT_1^2, one party, the sender, has as its input two strings m_0, m_1 and another party, the receiver, has as its input a bit b. At the end of the protocol, the receiver learns m_b and the sender learns nothing. Similarly, in 1-out-of-N OT the receiver obtains one of the N strings held by the sender. There is a rich body of research literature on OT, and in this work we use its efficient implementation from [32] as well as techniques from [24] that reduce a large number of OT protocol executions to κ of them, where κ is the security parameter. This, in particular, means that obtaining the keys corresponding to P_2's inputs in garbled circuit evaluation by P_2 incurs only small overhead.

4 Secure Iris Identification

As indicated in equation 1, computing the distance between two iris codes involves performing the division operation. While techniques for carrying out this operation using secure multi-party computation are known (see, e.g., [1,9,6,10]), their performance in practice even using very recent results is far from satisfactory for this application As an example, Blanton [5] reports that two-party evaluation of garbled circuits produced by Fairplay takes several seconds for numbers of length 24–28 bits, but circuits for longer integers could not be constructed due to the rapidly increasing memory requirements of Fairplay. Hoens et al. [19] report that building a multi-party division protocol using homomorphic encryption alone requires on the order of an hour to carry out the operation for 32-bit integers. Fortunately, in our case the computation can be rewritten to completely avoid this operation and replace it with multiplication. That is, using the notation $HD(X, Y) = ||(X \oplus Y) \cap M(X) \cap M(Y)|| \,/\, ||M(X) \cap M(Y)|| = D(X, Y) \,/\, M(X, Y)$, instead of testing whether $HD(X, Y) \overset{?}{<} T$, we can test whether $D(X, Y) \overset{?}{<} T \cdot M(X, Y)$. While the computation of the minimum distance as used in equation 2 is no longer possible, we can replace it with equivalent computation that does not increase its cost. Now the computation becomes:

$$\Big(D(X, \mathsf{LS}^c(Y)) \overset{?}{<} T \cdot M(X, \mathsf{LS}^c(Y))\Big) \vee \cdots \vee \Big(D(X, \mathsf{RS}^c(Y)) \overset{?}{<} T \cdot M(X, \mathsf{RS}^c(Y))\Big) \tag{4}$$

When this computation is carried over real numbers, T lies in the range [0, 1]. In our case, we need to carry the computation over the integers, which means that we "scale up" all values with the desired level of precision. That is, by using ℓ bits to achieve desired precision, we multiply $D(X,Y)$ by 2^ℓ and let T range between 0 and 2^ℓ. Now $2^\ell D(X,Y)$ and $T \cdot M(X,Y)$ can be represented using $\lceil \log m \rceil + \ell$ bits.

Security. Due to space constraints, we defer the security analysis of our iris identification protocol, described in Sections 4.1 and 4.2 below, to Appendix A.

4.1 Base Protocol

In what follows, we first describe the protocol in its simplest form. Section 4.2 presents optimizations and the resulting performance of the protocol.

In our solution, the client C generates a public-private key pair (pk, sk) for a homomorphic encryption scheme and distributes the public key pk. This is a one-time setup cost for the client for all possible invocations of this protocol with any number of servers. During the protocol itself, the secure computation proceeds as specified in equation 4. In the beginning, C sends its inputs encrypted with pk to the server S. At the server side, the computation first proceeds using homomorphic encryption, but later the client and the server convert the intermediate result into a split form and finish the computation using garbled circuit evaluation. This is due to the fact that secure two-party computation of the comparison is the fastest using garbled circuit evaluation [26], but the rest of the computation in our case is best performed on encrypted values.

To compute $D(X,Y) = \sum_{i=1}^{m} (X_i \oplus Y_i) \wedge M(X_i) \wedge M(Y_i)$ using algebraic computation, we use $X_i \oplus Y_i = X_i(1-Y_i) + (1-X_i)Y_i$ and obtain $D(X,Y) = \sum_{i=1}^{m} (X_i(1-Y_i) + (1-X_i)Y_i)M(X_i)M(Y_i)$. $M(X,Y)$ is computed as $\sum_{i=1}^{m} M(X_i)M(Y_i)$. Then if S obtains encryptions of $X_i M(X_i)$, $(1-X_i)M(X_i)$, and $M(X_i)$ for each i from C, the server will be able to compute $D(X,Y)$ and $M(X,Y)$ using its knowledge of the Y_i's and the homomorphic properties of the encryption. Figure 1 describes the protocol, in which after receiving C's encrypted values S produces $\mathsf{Enc}(M(X_i))$'s and proceeds to compute $D(X,Y^j)$ and $M(X,Y^j)$ in parallel for each Y in its database, where Y^j denotes biometric Y shifted by j positions and j ranges from $-c$ to c. At the end of steps 3(a).i and 3(a).ii the server obtains $\mathsf{Enc}(2^\ell D(X,Y^j) + r_S^j)$ for a randomly chosen r_S^j of its choice, and at the end of step 3(a).iii S obtains $\mathsf{Enc}(T \cdot M(X,Y^j) + t_S^j)$ for a random t_S^j of its choice. The server sends these values to the client who decrypt them. Therefore, at the end of step 3(a) C holds $r_C^j = 2^\ell D(X,Y^j) + r_S^j$ and $t_C^j = T \cdot M(X,Y^j) + t_S^j$ and S holds $-r_S^j$ and $-t_C^j$, i.e., they additively share $2^\ell D(X,Y^j)$ and $T \cdot M(X,Y^j)$.

What remains to compute is $2c+1$ comparisons (one per each Y^j) followed by $2c$ OR operations as specified by equation 4. This is accomplished using garbled circuit evaluation, where C enters r_C^j's and t_C^j's and S enters r_S^j's and t_S^j's and they learn a bit, which indicates whether Y was a match.

Note that since r_C^j's, r_S^j's, t_C^j's and t_S^j's are used as inputs to the garbled circuit and will need to be added inside the circuit, we want them to be as small as possible. Therefore, instead of providing unconditional hiding by choosing t_S^j and r_C^j from $\mathbb{Z}_N^*$ (where N is from pk), the protocol achieves statistical hiding by choosing these random values to be κ bits longer than the values that they protect, where κ is a security parameter.

Input: C has biometric X, $M(X)$ and key pair (pk, sk); S has a database D composed of Y, $M(Y)$ biometrics.

Output: C learns what records in D resulted in match with X if any, i.e., it learns a bit as a result of comparison of X with each $Y \in D$.

Protocol steps:

1. For each $i = 1, \ldots, m$, C computes encryptions $\langle a_{i1}, a_{i2}\rangle = \langle \mathsf{Enc}(X_i M(X_i)), \mathsf{Enc}((1-X_i)M(X_i))\rangle$ and sends them to S.
2. For each $i = 1, \ldots, m$, S computes encryption of $M(X_i)$ by setting $a_{i3} = a_{i1} \cdot a_{i2} = \mathsf{Enc}(X_i M(X_i)) \cdot \mathsf{Enc}((1-X_i)M(X_i)) = \mathsf{Enc}(M(X_i))$.
3. For each record Y in the database, S and C perform the following steps in parallel:
   (a) For each amount of shift $j = -c, \ldots, 0, \ldots, c$, S rotates the bits of Y by the appropriate number of positions to obtain Y^j and proceeds with all Y^j's in parallel.
      i. To compute $(X_i \oplus Y_i^j)M(X_i)M(Y_i^j) = (X_i(1-Y_i^j) + (1-X_i)Y_i^j)M(X_i)M(Y_i^j)$ in encrypted form, S computes $b_i^j = a_{i1}^{(1-Y_i^j)M(Y_i^j)} \cdot a_{i2}^{Y_i^j M(Y_i^j)} = \mathsf{Enc}(X_i M(X_i)(1-Y_i^j)M(Y_i^j) + (1-X_i)M(X_i)Y_i^j M(Y_i^j))$.
      ii. S adds the values contained in b_i^j's to obtain $b^j = \prod_{i=1}^m b_i^j = \mathsf{Enc}(\sum_{i=1}^m (X_i \oplus Y_i^j)M(X_i)M(Y_i^j)) = \mathsf{Enc}(||(X \oplus Y^j) \cap M(X) \cap M(Y^j)||)$. S then "lifts up" the result, blinds, and randomizes it as $c^j = (b^j)^{2^\ell} \cdot \mathsf{Enc}(r_S^j)$, where $r_S^j \xleftarrow{R} \{0,1\}^{\lceil \log m \rceil + \ell + \kappa}$, and sends the resulting c^j to C.
      iii. To obtain $T(||M(X) \cap M(Y^j)||)$, S computes $d_i^j = a_{i3}^{M(Y_i^j)} = \mathsf{Enc}(M(X_i) \cdot M(Y_i^j))$ and $d^j = (\prod_{i=1}^m d_i^j)^T = \mathsf{Enc}(T(\sum_{i=1}^m M(X_i)M(Y_i^j)))$. S blinds and randomizes the result as $e^j = d^j \cdot \mathsf{Enc}(t_S^j)$, where $t_S^j \xleftarrow{R} \{0,1\}^{\lceil \log m \rceil + \ell + \kappa}$, and sends e^j to C.
      iv. C decrypts the received values and sets $r_C^j = \mathsf{Dec}(c^j)$ and $t_C^j = \mathsf{Dec}(e^j)$.
   (b) C and S perform $2c+1$ comparisons and OR of the results of the comparisons using garbled circuit. C enters r_C^j's and t_C^j's, S enters $-r_S^j$'s and $-t_S^j$'s, and C learns bit b computed as $\bigvee_{j=-c}^{c}((r_C^j - r_S^j) \overset{?}{<} (t_C^j - t_S^j))$. To achieve this, S creates the garbled circuit and sends it to C. C obtains keys corresponding to its inputs using OT, evaluates the circuit, and S sends to C the key-value mapping for the output gate.

Fig. 1. Secure two-party protocol for iris identification

4.2 Optimizations

Pre-computation and Offline Communication. Similar to prior literature on secure biometric identification [16,37,33,3], we distinguish between offline and online stages, where any computation and computation that does not depend on the inputs of the participating parties can be moved to the offline stage. In our protocol, first notice that most modular exponentiations (the most expensive operation in the encryption scheme) can be precomputed. That is, the client needs to produce $2m$ encryptions of bits. Because both m and the average number of 0's and 1's in a biometric and a mask are known, the client can produce a sufficient number of bit encryptions in advance. In particular, X normally will have 50% of 0's and 50% of 1's, while 75% (or a similar number) of $M(X)$'s bits are set to 1 and 25% to 0 during biometric processing. Let p_0 and p_1 (q_0 and q_1) denote the fraction of 0's and 1's in an iris code (resp., its mask), where $p_0 + p_1 = q_0 + q_1 = 1$. Therefore, to have a sufficient supply of ciphertexts to form tuples $\langle a_{i1}, a_{i2} \rangle$, the client needs to precompute $(2q_0 + q_1(p_1 + \varepsilon) + q_1(p_0 + \varepsilon))m = (1 + q_0 + 2q_1\varepsilon)m$ encryptions of 0 and $(q_1(p_1 + \varepsilon) + q_1(p_0 + \varepsilon))m = q_1(1 + 2\varepsilon)m$ encryptions of 1, where ε is used as a cushion since the number of 0's and 1's in X might not be exactly p_0 and p_1, respectively. Then at the time of the protocol the client simply uses the appropriate ciphertexts to form its transmission.

Similarly, the server can precompute a sufficient supply of encryptions of r_S^j's and t_S^j's for all records. That is, the server needs for produce $2(2c + 1)|D|$ encryptions of different random values of length $\lceil \log m \rceil + \ell + \kappa$, where $|D|$ denotes the size of the database D. The server also generates one garbled circuit per record Y in its database (for step 3(b) of the protocol) and communicates the circuits to the client. In addition, the most expensive part of the oblivious transfer can also be performed during the offline stage, as detailed below.

Optimized Multiplication. Server's computation in steps 3(a).i and 3(a).iii of the protocol can be significantly lowered as follows. To compute ciphertexts b_i^j, S needs to calculate $a_{i1}^{(1-Y_i^j)M(Y_i^j)} \cdot a_{i2}^{Y_i^j M(Y_i^j)}$. Since the bits Y_i^j and $M(Y_i^j)$ are known to S, this computation can be rewritten using one of the following cases:

- $Y_i^j = 0$ and $M(Y_i^j) = 0$: in this case both $(1 - Y_i^j)M(Y_i^j)$ and $Y_i^j M(Y_i^j)$ are zero, which means that b_i^j should correspond to an encryption of 0 regardless of a_{i1} and a_{i2}. Instead of having S create an encryption 0, we set b_i^j to the empty value, i.e., it is not used in the computation of b^j in step 3(a).ii.
- $Y_i^j = 1$ and $M(Y_i^j) = 0$: the same as above.
- $Y_i^j = 0$ and $M(Y_i^j) = 1$: in this case $(1 - Y_i^j)M(Y_i^j) = 1$ and $Y_i^j M(Y_i^j) = 0$, which means that S sets $b_i^j = a_{i1}$.
- $Y_i^j = 1$ and $M(Y_i^j) = 1$: in this case $(1 - Y_i^j)M(Y_i^j) = 0$ and $Y_i^j M(Y_i^j) = 1$, and S therefore sets $b_i^j = a_{i2}$.

The above implies that only $q_1 m$ ciphertexts b_i^j need to be added in step 3(a).ii to form b^j (i.e., $q_1 m - 1$ modular multiplications to compute the hamming distance between m-element strings).

Similar optimization applies to the computation of d_i^j and d^j in step 3(a).iii of the protocol. That is, when $M(Y_i^j) = 0$, d_i^j is set to the empty value and is not used in the computation of d^j; when $M(Y_i^j) = 1$, S sets $d_i^j = a_{i3}$. Consequently, $q_1 m$ ciphertexts are used in computing d^j.

To further reduce the number of modular multiplications, we can adopt the idea from [33], which consists of precomputing all possible combinations for ciphertexts at positions i and $i + 1$ and reducing the number of modular multiplications used during processing a database record in half. In our case, the value of $b_i^j b_{i+1}^j$ requires computation only when $M(Y_i^j) = M(Y_{i+1}^j) = 1$. In this case, computing $a_{i1}a_{(i+1)1}$, $a_{i1}a_{(i+1)2}$, $a_{i2}a_{(i+1)1}$, and $a_{i2}a_{(i+1)2}$, for each odd i between 1 and $m - 1$ will cover all possibilities. Note that these values need to be computed once for all possible shift amounts of the biometrics (since only server's Y's are shifted). Depending on the distribution of the set bits in each $M(Y)$, the number of modular multiplication now will be between $q_1 m/2$ (when $M(Y_i) = M(Y_{i+1})$ for each odd i) and $m(q_0 + (1 - 2q_0)/2) = m/2$ (when $M(Y_i) \neq M(Y_{i+1})$ for as many odd i's as possible). This approach can be also applied to the computation of d^j (where only the value of $a_{i3}a_{(i+1)3}$ needs to be precomputed for each odd i) resulting in the same computational savings during computation of the hamming distance. Furthermore, by precomputing the combinations of more than two values additional savings can be achieved during processing of each Y.

Optimized Encryption Scheme. As it is clear from the protocol description, its performance crucially relies on the performance of the underlying homomorphic encryption scheme for encryption, addition of two encrypted values, and decryption. Instead of utilizing a general purpose encryption scheme such as Paillier, we turn our attention to schemes of restricted functionality which promise to offer improved efficiency. In particular, the DGK additively homomorphic encryption scheme [13,12] was developed to be used for secure comparison, where each ciphertext encrypts a bit. In that setting, it has faster encryption and decryption time than Paillier and each ciphertext has size k using a k-bit RSA modulus (while Paillier ciphertext has size $2k$). To be suitable for our application, the encryption scheme needs to support larger plaintext sizes. The DGK scheme can be modified to work with longer plaintexts. In that case, at decryption time, one needs to additionally solve the discrete logarithm problem where the base is 2-smooth using Pohlig-Hellman algorithm. This means that decryption uses additional $O(n)$ modular multiplications for n-bit plaintexts. Now recall that in the protocol we encrypt messages of length $\lceil \log m \rceil + \ell + \kappa$ bits. The use of the security parameter κ significantly increases the length of the plaintexts. We, however, notice that the DGK encryption can be setup to permit arithmetic on encrypted values such that all computations on the underlying plaintexts are carried modulo 2^n for any n. For our protocol it implies that (i) the blinding values r_S^j and t_S^j can now be chosen from the range $[0, 2^n - 1]$, where $n = \lceil \log m \rceil + \ell$, and (ii) this provides information-theoretic hiding (thus improving the security properties of the protocol). This observation has a profound impact not only on the client decryption time in step 3(a).iv (which decreases by about an order of magnitude), but also on the consecutive garbled circuit evaluation, where likewise the circuit size is significantly reduced in size.

Circuit Construction. We construct garbled circuits using the most efficient techniques from [35] and references therein. By performing addition modulo 2^n and eliminating gates which have a constant value as one of their inputs, we reduce the complexity of the circuit for addition to $n - 1$ non-XOR gates and $5(n - 1) - 1$ total gates. Similarly, after eliminating gates with one constant input, the complexity of the circuit for comparison of n-bit values becomes n non-XOR gates and $4n - 2$ gates overall. Since in the protocol there are two additions and one comparison per each j followed by $2c$ OR gates, the size of the overall circuit is $14(n - 1)(2c + 1) + 2c$ gates, $(3n - 2)(2c + 1) + 2c$ of which are non-XOR gates. Note that this circuit does not

Table 1. Breakdown of the performance of the iris identification protocol. Time is expressed in milliseconds unless otherwise stated, and communication overhead in KB.

		Offline			Online		
	Setup	enc	circuit	total	enc	circuit	total
Server	$c = 5$	1398 + 71/rec	1780 + 8.5/rec	3178 + 79.5/rec	108 + 148/rec	1.2/rec	89 + 149.2/rec
	$c = 0$	1398 + 6.5/rec	1457 + 0.7/rec	2855 + 7.2/rec	108 + 13.6/rec	0.1/rec	89 + 13.7/rec
Client	$c = 5$	11.93s	1693 + 3.4/rec	13.62s + 3.4/rec	20/rec	2.6/rec	22.6/rec
	$c = 0$	11.93s	1055 + 0.3/rec	12.99s + 0.3/rec	1.8/rec	0.2/rec	2.0/rec
Comm	$c = 5$	512	11.6 + 22.1/rec	524 + 22.1/rec	0.5 + 2.7/rec	17.2/rec	0.5 + 19.9/rec
	$c = 0$	512	11.6 + 2/rec	524 + 2/rec	0.5 + 0.2/rec	1.6/rec	0.5 + 1.8/rec

use multiplexers, which are required (and add complexity) during direct computation of minimum.

Oblivious Transfer. The above circuit requires each party to supply $2n(2c+1)$ input bits, and a new circuit is used for each Y in D. Similar to [18], the combination of techniques from [24] and [32] achieves the best performance in our case. Let the server create each circuit and the client evaluate them. Using the results of [24], performing OT_1^2 the total of $2n(2c+1)|D|$ times, where the client receives a κ-bit string as a result of each OT for a a security parameter κ, can be reduced to κ invocations of OT_1^2 (that communicates to the receiver κ-bit strings) at the cost of $4\kappa \cdot 2n(2n+1)|D|$ bits of communication and $4n(2c+1)$ applications of a hash function for the sender and $2n(2c+1)$ applications for the receiver. Then κ OT_1^2 protocols can be implemented using the construction of [32] with low amortized complexity, where the sender performs $2+\kappa$ and the receiver performs 2κ modular exponentiations with the communication of $2\kappa^2$ bits and κ public keys. The OT protocols can be performed during the offline stage, while the additional communication takes place once the inputs are known.

Further Reducing Online Communication. If transmitting $2m$ ciphertexts during the online stage of the protocol (which amounts to a few hundred KB for our set of parameters) constitutes a burden, this communication can be performed at the offline stage before the protocol begins. This can be achieved using the technique of [33]. We refer the reader to the full version [7] for details of applying this technique to our solution.

4.3 Implementation and Performance

We implemented the secure iris identification protocol in C using MIRACL library [22] for cryptographic operations. The implementation used DGK encryption scheme [13,12] with a 1024-bit modulus and another security parameter t set to 160, as suggested in [13,12]. To simplify comparisons with prior work, throughout this work we use $k = 1024$ security parameter for public-key cryptography and $\kappa = 80$ for symmetric and statistical security. The experiments were run on an Intel Core 2 Duo 2.13 GHz with 3GB of RAM and *gcc* version 4.4.5 on Linux.

Table 1 shows performance of the secure iris identification protocol and its components. The performance was obtained using the following set of parameters: the size of iris code and mask $m = 2048$ (this value of m is used in commercial iris recognition software), 75% of bits are reliable in each iris code, and the length n of values is 20 bits. All optimizations described earlier in this section were implemented. In our implementation, upon receipt of client's data, the server precomputes all combinations for pairs of ciphertexts $b_i b_{i+1}$ in step 3(a).ii (one-time cost of the total of $4(m/2)$ modular multiplications) and all combinations of 4 elements $d_i d_{i+1} d_{i+2} d_{i+3}$ in step 3(a).iii (one-time cost of $11(m/4)$ modular multiplications). This cuts the server's time for processing each Y by more than a half. Furthermore, the constant overhead associated with the OT (circuit) can be reduced in terms of both communication and computation for both parties if public-key operations are implemented over elliptic curves.

The table shows performance using different configurations with the amount of rotation $c = 5$ and no rotation with $c = 0$ (this is used when the images are well aligned, which is the case for iris biometrics collected at our institution). In the table, we divide the computation and communication into offline precomputation and online protocol execution. No inputs are assumed to be known by any party at precomputation time. Some of the overhead depends on the server's database size, in which case the computation and communication are indicated per record (using notation "/rec"). The overhead associated with the part of the protocol that uses homomorphic encryption is shown separately from the overhead associated with garbled circuits. The offline and online computation for the part based on homomorphic encryption is computed as described in Section 4.2. For circuits, garbled circuit creation, communication, and some of OT is performed at the offline stage, while the rest of OT (as described in Section 4.2) and garbled circuit evaluation takes place during the online protocol execution.

It is evident that our protocol design and optimizations allow us to achieve notable performance. In particular, comparison of two iris codes, which includes computation of $2(2c+1) = 22$ Hamming distances over 2048-bit biometrics in encrypted form, is done in 0.15 sec. This is noticeably lower than 0.3 sec online time per record reported by the best currently known face recognition protocol SCiFI [33], which computes a single Hamming distance over 900-bit values. That is, despite an order of magnitude larger number of operations and more complex operations such as division, computation of minimum, etc., we are able to outperform prior work by roughly 50%. This in particular implies that using the techniques suggested in this work (and DGK encryption scheme in particular) performance of SCiFI and other existing protocols can be improved to a fraction of the previously reported time. When iris images are well aligned and no rotation is necessary our protocol requires only 14 msec online computation time and under 2KB of data to compare two biometrics.

5 Secure Fingerprint Identification

In this section we illustrate how a number of the techniques developed in this work for iris identification can be applied to other types of biometric computations such as FingerCodes. In particular, we show that the efficiency of the secure protocol for FingerCode identification [3] can be improved by an order of magnitude.

The computation involved in FingerCode comparisons is very simple, which results in an extremely efficient privacy-preserving realization. Similar to [3], we rewrite the computation in equation 3 as $\sum_{i=1}^{m}(x_i - y_i)^2 = \sum_{i=1}^{m}(x_i)^2 + \sum_{i=1}^{m}(y_i)^2$

Input: C has biometric $X = (x_1, \ldots, x_m)$ and DGK encryption key pair (pk, sk); S has a database D composed of biometrics $Y = (y_1, \ldots, y_m)$.
Output: C learns what records in D resulted in match with X if any, i.e., it learns a bit as a result of comparison of X with each $Y \in D$.
Protocol steps:

1. C computes and sends to S encryptions $\mathsf{Enc}(-2x_1), \ldots, \mathsf{Enc}(-2x_m), \mathsf{Enc}(\sum_{i=1}^{m} x_i^2)$.
2. For each $Y = (y_1, \ldots, y_m) \in D$, S and C perform in parallel:
   (a) S computes the encrypted distance d between X and Y as $d = \mathsf{Enc}(\sum_{i=1}^{m} x_i^2) \cdot \mathsf{Enc}(\sum_{i=1}^{m} y_i^2) \cdot \prod_{i=1}^{m} \mathsf{Enc}(-2x_i)^{y_i} = \mathsf{Enc}(\sum_{i=1}^{m}(x_i - y_i)^2)$, blinds it as $d' = d \cdot \mathsf{Enc}(r_S)$, where $r_S \xleftarrow{R} \{0,1\}^n$, and sends d' to C.
   (b) C decrypts the value it receives and sets $r_C = \mathsf{Dec}(d')$.
   (c) C and S engage in a secure protocol that computes $((r_C - r_S) \bmod 2^n) \overset{?}{<} T^2$ using garbled circuit evaluation. S creates the circuit and sends it to C along with the key-value mapping for the output gate. C obtains keys corresponding to its inputs from S using OT, evaluates the circuit, and learns the result.

Fig. 2. Secure two-party protocol for FingerCode identification

$-\sum_{i=1}^{m} 2x_i y_i < T^2$. In our protocol, the Euclidean distance is computed using homomorphic encryption, while the comparisons are performed using garbled circuits. The secure FingerCode protocol is given in Figure 2: the client contributes encryptions of $-2x_i$ and $\sum(x_i)^2$ to the computation, while the server contributes $\sum(y_i)^2$ and computes encryption of $-2x_i y_i$ from $-2x_i$. Note that by using $\mathsf{Enc}(-2x_i)$ instead of $\mathsf{Enc}(x_i)$, the server's work for each Y is reduced since negative values use significantly longer representations. The protocol in Figure 2 uses DGK encryption with the plaintext space of $[0, 2^n - 1]$. To be able to represent the Euclidean distance, we need to set $n = \lceil \log m \rceil + 2\ell + 1$, where ℓ is the bitlength of elements x_i and y_i. This implies that all computation on plaintexts is performed modulo 2^n; for instance, $2^n - 2x_i$ is used in step 1 to form $\mathsf{Enc}(-2x_i)$. The circuit used in step 2(c) takes two n-bit values, adds them modulo 2^n, and compares the result to a constant as described in Section 4.2.

Finally, some of the computation can be performed offline: for the client it includes precomputing the random values used in the $m + 1$ ciphertexts it sends in step 1 (computation of $h^r \bmod N$), and for the server includes precomputing $\mathsf{Enc}(r_S)$ and preparing a garbled circuit for each Y, as well as one-time computation of random values for $\mathsf{Enc}(\sum_{i=1}^{m}(y_i)^2)$ since the reuse of such randomness does not affect security. The client and the server also perform some of OT functionality prior to protocol initiation.

In the FingerCode protocol of [3], each fingerprint in the server's database is represented by c FingerCodes that correspond to different orientations of the same fingerprint, which improves the accuracy of matching. The protocol of [3], however, reports all matches within the c FingerCodes corresponding to the same fingerprint, and this is what our protocol in Figure 2 computes. If it is desirable to output only a single bit for all c instances of a fingerprint, it is easy to modify the circuit evaluated in step 2(c) of the protocol to compute the OR of the bits produced by the original c circuits.

Security. The security of this protocol is straightforward to show and we omit the details of the simulator from the current description.

Table 2. Breakdown of the performance of the FingerCode identification protocol. Time is expressed in milliseconds and communication overhead in KB.

	Offline			Online		
	enc	circuit	total	enc	circuit	total
Server	3.6 + 3.9/rec	1448 + 0.37/rec	1451.6 + 4.3/rec	0.22 + 1.37/rec	0.05/rec	0.22 + 1.42/rec
Client	61	1025 + 0.15/rec	1086 + 0.15/rec	4.7 + 0.92/rec	0.16/rec	4.7 + 1.08/rec
Comm	0	11.6 + 1.26/rec	11.6 + 1.26/rec	2.12 + 0.12/rec	0.74/rec	2.12 + 0.86/rec

Implementation and Performance. The FingerCode parameters can range as $m =$ 16–640, $\ell =$ 4–8, and $c = 5$. We implement the protocol using parameters $m = 16$ and $\ell = 7$ (the same as in [3]) and therefore $n = 19$. The performance of our secure FingerCode identification protocol is given in Table 2. No inputs (X or Y) are assumed to be known at the offline stage when the parties compute randomization values of the ciphertexts. For that reason, a small fixed cost is inquired in the beginning of the protocol to finish forming the ciphertext using the data itself. We also note that, based on our additional experiments, by using Paillier encryption instead of DGK encryption, the server's online work increases by an order of magnitude, even if packing is used.

It is evident that the overhead reported in the table is minimal and the protocol is well suited for processing fingerprint data in real time. In particular, for a database of 320 records used in prior work (64 fingerprints with 5 FingerCodes each used in [3]), client's online work is 0.35 sec and the server's online work is 0.45 sec, with online communication of 279KB. As can be seen from these results, computation is no longer the bottleneck and this secure two-party protocol can be carried out extremely efficiently. Compared to the solution in [3] that took 16 sec for the online stage with the same setup, the computation speed up is by a factor of 35. Communication efficiency, however, is what was specifically emphasized in the protocol of [3] resulting in 10101KB online overhead for a database of size 320. Our solution therefore improves such result by a factor of 35. We also would like to note that all offline work in [3] is for ciphertext precomputation (since no garbled circuits are used) and is non-interactive, while in our protocol circuit transmission and input-independent portions of OT can be done prior to the protocol itself and involve interaction. We, however, note that the overall (offline and online) computation for $|D| = 320$ is 1.48 sec for the client and 3.27 sec for the server with the total of 692KB communication, which is still at least several times lower than the online portion of the time and communication in [3].

Privacy-preserving face recognition techniques by Sadeghi et al. [37] can also be adapted to perform secure FingerCode comparisons. They were developed for a different context, but also involve computing Euclidean distances using homomorphic encryption, followed by garbled circuits-based comparisons of the results. (Comparison of face images also includes projecting a client's face to a different vector space as the first step of the protocol, but it is not needed here.) The authors of [37] use Paillier homomorphic encryption for distance computation, where packing of multiple values into a single ciphertext is used at certain points of the protocol. In particular, ciphertext d' that the server sends to the client in step 2(b) contains up to $t = \lfloor \frac{k-\kappa}{n} \rfloor$ distances (for k-bit modulus and statistical security parameter κ), where $t = 49$ in our case. This results in fewer $\mathsf{Enc}(r_S)$ to form, transmit, and decrypt.

When we compare communication of our protocol with that based on techniques of [37], we obtain that the initial transmission of client biometric is lower by a factor of 2 in our protocol. The circuit size, and thus corresponding work and communication, is slightly larger in [37] due to handling of additional κ bits per t distances. The communication associated with transmission of encrypted blinded distances, however, is significantly lower in [37] due to packing. Overall, we obtain that communication of both solutions is very similar because the communication overhead is heavily dominated by garbled circuits. For the distance computation, [37] report runtime of 6.08 sec for the client and 0.47 sec for the server for $|D| = 320$, while distance computation in our protocol (including precomputation) is 0.36 sec for the client and 1.69 sec for the server for the same $|D|$. In [37] the number of dimensions $m = 12$, while we have $m = 16$, but the length of values is $n = 50$ in [37], while we have $n = 19$. The computation itself in [37] is more expensive (including interaction between the parties, which we do not have) due to the need to transform client's data, but faster machines are used.

To obtain a better insight on how performance of Paillier encryption with packing compares to that of DGK encryption for our application, we implemented our protocol using techniques of [37] (note that distance computation is more efficient than what is described in [37]). We used a 1024-bit modulus and a number of optimizations suggested in [34] for best performance. In particular, small generator $g = 2$ was used to achieve lower encryption time, and decryption is sped up using pre-computation and Chinese remainder computation (see [34], section 7 for more detail). For optimally packed values which result in the lowest overhead per record, we obtain that the server's precomputation is 31.9 + 1.31/rec (all in msec), server's online work is 1.0 + 24.68/rec, client's precomputation is 545.4, and client's online work is 509.6 + 0.22/rec. For $|D| = 320$, we obtain the client's overall runtime of 1.13 sec and server's runtime of 8.24 sec, where the increase in time compared to the performance in [37] can be explained by larger m and slower machines (note that this is the opposite of what is reported in [37]; the server clearly performs the majority of distance computation work). We obtain that our approach is faster by a factor of almost 5 than the use of Paillier encryption with packing as suggested in [37] and online work is faster by a factor of 9.3. And as previously described, the circuit overhead of [37] is slightly larger due to the need to achieve statistical hiding of computed distances.

6 Conclusions

The protocol design presented in this work suggests certain principles that lead to an efficient implementation of a privacy-preserving protocol for biometric identification: (i) representation of client's biometric plays an important role; (ii) operations that manipulate bits are the fastest using tools other than encryption; (iii) a proper tuning of encryption tools can result in a significant speedup. Using these principles and a number of new techniques in this work we develop and implement secure protocols for iris and fingerprint identification that use standard biometric recognition algorithms. The optimization techniques employed in this work allow us to achieve notable performance results for different secure biometric identification protocols.

In particular, we develop the first privacy-preserving two-party protocol for iris codes using current biometric recognition algorithms. Despite the length of iris codes' representation and the complexity of their processing, our protocol allows a secure comparison between two biometrics to be performed in 0.15 sec with communication of under 18KB. Furthermore, when the iris codes are known to be well-aligned and their rotation is not necessary, the overhead decreases by an order of magnitude to 14 msec computation and 2KB communication per comparison.

Two FingerCodes used for fingerprint recognition can be compared at low cost, which allowed us to develop an extremely efficient privacy-preserving protocol. Comparing two fingerprints requires approximately 1 msec of computation, allowing thousands of biometrics to be processed in a matter of seconds. Communication overhead is also very modest with less than 1KB per biometric comparison. Compared to prior privacy-preserving implementation of FingerCode [3], we simultaneously improve online computation and communication by a factor of more than 30.

Acknowledgments. We would like to thank Keith Frikken for suggestion to reduce communication of the iris identification protocol from $3m$ to $2m$ ciphertexts. We would also like to thank Stefan Katzenbeisser and the anonymous reviewers for their insightful comments and observations. Portions of this work were sponsored by the Air Force Office of Scientific Research grant AFOSR-FA9550-09-1-0223.

References

1. Atallah, M., Bykova, M., Li, J., Frikken, K., Topkara, M.: Private collaborative forecasting and benchmarking. In: ACM Workshop on Privacy in the Electronic Society (WPES), pp. 103–114 (2004)
2. Barbosa, M., Brouard, T., Cauchie, S., de Sousa, S.M.: Secure biometric authentication with improved accuracy. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 21–36. Springer, Heidelberg (2008)
3. Barni, M., Bianchi, T., Catalano, D., Di Raimondo, M., Labati, R., Failla, P., Fiore, D., Lazzeretti, R., Piuri, V., Scotti, F., Piva, A.: Privacy-preserving fingercode authentication. In: ACM Workshop on Multimedia and Security (MM&Sec), pp. 231–240 (2010)
4. Barzegar, N., Moin, M.: A new user dependent iris recognition system based on an area preserving pointwise level set segmentation approach. EURASIP Journal on Advances in Signal Processing, 1–13 (2009)
5. Blanton, M.: Empirical evaluation of secure two-party computation models. Technical Report TR 2005-58, CERIAS, Purdue University (2005)
6. Blanton, M., Aliasgari, M.: Secure computation of biometric matching. Technical Report 2009-03, Department of Computer Science and Engineering, University of Notre Dame (2009)
7. Blanton, M., Gasti, P.: Secure and Efficient Protocols for Iris and Fingerprint Identification. Cryptology ePrint Archive, Report 2010/627 (2010), http://eprint.iacr.org/
8. Bringer, J., Chabanne, H., Izabachène, M., Pointcheval, D., Tang, Q., Zimmer, S.: An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 96–106. Springer, Heidelberg (2007)
9. Bunn, P., Ostrovsky, R.: Secure two-party k-means clustering. In: ACM Conference on Computer and Communications Security (CCS), pp. 486–497 (2007)
10. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (2010)

11. Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)
12. Damgård, I., Geisler, M., Krøigård, M.: A correction to efficient and secure comparison for on-line auctions. Cryptology ePrint Archive, Report 2008/321 (2008)
13. Damgård, I., Geisler, M., Krøigård, M.: Homomorphic encryption and secure comparison. Journal of Applied Cryptology 1(1), 22–31 (2008)
14. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
15. Daugman, J.: How iris recognition works. IEEE Transactions on Circuits and Systems for Video Technology 14(1), 21–30 (2004)
16. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009)
17. Frikken, K.B.: Privacy-preserving set union. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 237–252. Springer, Heidelberg (2007)
18. Henecka, W., Kogl, S., Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-partY computations. In: ACM Conference on Computer and Communications Security (CCS), pp. 451–462 (2010)
19. Hoens, T., Blanton, M., Chawla, N.: A private and reliable recommendation system using a social network. In: IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT), pp. 816–825 (2010)
20. Hollingsworth, K., Bowyer, K., Flynn, P.: The best bits in an iris code. IEEE Transactions on Pattern Analysis and Machine Intelligence 31(6), 964–973 (2009)
21. U.S. DHS US-VISIT, http://www.dhs.gov/files/programs/usv.shtm
22. Multiprecision Integer and Rational Arithmetic C/C++ Library, http://www.shamus.ie/
23. IrisGuard Press Release, http://cl.ly/3KIB
24. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
25. Jain, A., Prabhakar, S., Hong, L., Pankanti, S.: Filterbank-based fingerprint matching. IEEE Transactions on Image Processing 9(5), 846–859 (2000)
26. Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009)
27. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
28. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
29. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
30. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay – a secure two-party computation system. In: USENIX Security Symposium, pp. 287–302 (2004)
31. Maltoni, D., Maio, D., Jain, A., Prabhakar, S.: Hanbook of Fingerprint Recognition, 2nd edn. Springer, Heidelberg (2009)
32. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 448–457 (2001)

33. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: SCiFI – A system for secure face identification. In: IEEE Symposium on Security and Privacy, pp. 239–254 (2010)
34. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
35. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
36. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: ACM Symposium on Theory of Computing (STOC), pp. 73–85 (1989)
37. Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: Efficient privacy-preserving face recognition. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 229–244. Springer, Heidelberg (2010)
38. Troncoso-Pastoriza, J., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient DNA searching through oblivious automata. In: ACM Conference on Computer and Communications Security (CCS), pp. 519–528 (2007)
39. Yao, A.: How to generate and exchange secrets. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 162–167 (1986)

A Security Analysis of the Iris Protocol

Security of the iris protocol relies on the security of the underlying building blocks. In particular, we need to assume that (i) the DGK encryption scheme is semantically secure (which was shown under a hardness assumption that uses subgroups of an RSA modulus [13,12]); (ii) garbled circuit evaluation is secure (which was shown assuming that the hash function is correlation robust [27], or if it is modeled as a random oracle); and (iii) the oblivious transfer is secure as well (to achieve this, techniques of [24] require the hash function to be correlation robust and the use of a pseudo-random number generator, while techniques of [32] model the hash functions as a random oracle and use the computational Diffie-Hellman (CDH) assumption). Therefore, assuming the security of the DGK encryption, CDH, and using the random oracle model for hash functions is sufficient for our solution.

To show the security of the protocol, we sketch how to simulate the view of each party using its inputs and outputs alone. If such simulation is indistinguishable from the real execution of the protocol, for semi-honest parties this implies that the protocol does not reveal any unintended information to the participants (i.e., they learn only the output and what can be deduced from their respective inputs and outputs).

First, consider the client C. The client's input consists of its biometric X, $M(X)$ and the private key, and its outputs consists of a bit b for each record in S's database D. A simulator that is given these values simulates C's view by sending encrypted bits of C's input to the server as prescribed in step 1 of the protocol. It then simulates the messages received by the client in step 3(a).iii using encryptions of two randomly chosen strings r_C^j and t_C^j of length n. The simulator next creates a garbled circuit for the computation given in step 3(b) that, on input client's r_C^j's and t_C^j's computes bit b, sends the circuit to the client, and simulates the OT. It is clear that given secure implementation of garbled circuit evaluation in the real protocol, the client cannot distinguish simulation from real protocol execution. Furthermore, the values that C recovers in step 3(a).iv of the protocol are distributed identically to the values used in the real protocol execution that uses DGK encryption (and they are statistically indistinguishable when other encryption schemes are used).

Now consider the server's view. The server has its database D consisting of Y, $M(Y)$ and the threshold T as the input and no output. In this case, a simulator with access to D first sends to S ciphertexts (as in step 1 of the protocol) that encrypt bits of its choice. For each $Y \in D$, S performs its computation in step 3(a) of the protocol and forms garbled circuits as specified in step 3(b). The server and the simulator engage in the OT protocol, where the simulator uses arbitrary bits as its input to the OT protocol and the server sends the key-value mapping for the output gate. It is clear that the server cannot distinguish the above interaction from the real protocol execution. In particular, due to semantic security of the encryption scheme S learns no information about the encrypted values and due to security of OT S learns no information about the values chosen by the simulator for the garbled circuit.

Linear Obfuscation to Combat Symbolic Execution

Zhi Wang[1], Jiang Ming[2], Chunfu Jia[1], and Debin Gao[3]

[1] College of Information Technical Science, Nankai University, China
zwang@mail.nankai.edu.cn, cfjia@nankai.edu.cn
[2] College of Information Sciences & Technology, Pennsylvania State University, USA
mingjiangpku@gmail.com
[3] School of Information Systems, Singapore Management University, Singapore
dbgao@smu.edu.sg

Abstract. Trigger-based code (malicious in many cases, but not necessarily) only executes when specific inputs are received. Symbolic execution has been one of the most powerful techniques in discovering such malicious code and analyzing the trigger condition. We propose a novel automatic malware obfuscation technique to make analysis based on symbolic execution difficult. Unlike previously proposed techniques, the obfuscated code from our tool does not use any cryptographic operations and makes use of only linear operations which symbolic execution is believed to be good in analyzing. The obfuscated code incorporates unsolved conjectures and adds a simple loop to the original code, making it less than one hundred bytes longer and hard to be differentiated from normal programs. Evaluation shows that applying symbolic execution to the obfuscated code is inefficient in finding the trigger condition. We discuss strengths and weaknesses of the proposed technique.

Keywords: Software obfuscation, symbolic execution, malware analysis.

1 Introduction

Symbolic execution was proposed as a program analysis technique more than three decades ago [21]. In recent years, symbolic execution has advanced a lot. It is usually combined with dynamic taint analysis and theorem proving, and is becoming a powerful technique in security analysis of software programs. In particular, symbolic execution has been shown to be useful in discovering trigger-based code (malicious in many cases, although not necessarily) and finding the corresponding trigger condition [3].

To fight against the state-of-the-art malware analyzers, Sharif et al. proposed a conditional code obfuscation scheme that obfuscates equality conditions that rely on inputs by introducing one-way hash functions [35]. It was shown that analyzers based on symbolic execution are hard to reason about the value of input that satisfies the equality condition. However, admitted by the authors, using cryptographic functions in the obfuscation might improve malware detection [35].

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 210–226, 2011.

$$a_i = \begin{cases} n & \text{for } i = 0 \\ f(a_{i-1}) & \text{for } i > 0 \end{cases}$$

$$\text{where } f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod{2} \\ 3n+1 & \text{if } n \equiv 1 \pmod{2} \end{cases}$$

Fig. 1. Collatz conjecture: a_i will eventually reach 1 regardless of the value of n

In this paper, we challenge the requirement of using cryptographic functions in obfuscation to make symbolic execution difficult, and propose a novel automatic obfuscation technique that makes use of linear unsolved conjectures. There are a few advantages of using only linear operations in the obfuscation without any cryptographic ones. First, the obfuscated code becomes less suspicious in malware detection. The obfuscated code produced by our technique only adds a simple loop to the code, making the resulting obfuscated code similar to legitimate programs that employ, e.g., simple number sorting algorithms. Second, such simple obfuscated code makes it possible for our technique to be combined with other obfuscation and polymorphism techniques to achieve stronger protection. Third, the size of the obfuscated code is less than one hundred bytes longer than the original program.

Many unsolved conjectures (e.g., the Collatz conjecture [23], see Figure 1) involve some simple linear operations on integers that loop for an unknown number of times. Such operations are usually fast and commonly used in basic algorithms in computer science. They are perfect candidates to be used in obfuscations to make symbolic execution difficult because symbolic execution is usually weak and inefficient in analyzing loops, in particular, the number of times the loop body executes [25,19,9,10,8,27,38,30,3].

Another advantage of using these unsolved conjectures is that they can be used to obfuscate inequality conditions, a case the previous proposal is unable to handle [35]. Although some inequality conditions could be transformed to (a set of) equality conditions, it might become impractical when the inequality range is big.

We propose and implement an automatic obfuscator to incorporate unsolved conjectures into trigger conditions in program source code. Extensive evaluations show that symbolic execution would take hundreds of hours in order to figure out the trigger condition.

The rest of the paper is organized as follows. Section 2 discusses the background of symbolic execution, unsolved conjectures, and related work. We detail our threat model and give an overview of the steps in our obfuscation technique in Section 3. Detailed implementation of our obfuscator is explained in Section 4. We show the evaluation results of the obfuscated code and discuss strengths and weaknesses in Section 5. Finally we conclude in Section 7.

2 Background and Related Work

In this section, we briefly discuss existing work on symbolic execution, its application, and limitations in handling loops. We also discuss related work on obfuscating software programs. At the end of the section, we outline some unsolved conjectures in mathematics which we make use of in our obfuscator.

2.1 Symbolic Execution and Its Applications

Forward symbolic execution has been extensively utilized in various security analysis techniques [33]. Automatic testing leverages forward symbolic execution to achieve high code coverage and automatic input generation [9,10,8,19,27,25,34,18]. Most of these applications automatically generate inputs to trigger well-defined bugs, such as integer overflow, memory errors, null pointer dereference, etc. Recent work shows that forward symbolic execution can be used to generate succinct and accurate input signatures or filters to block exploits [6,14,4,5]. Previous work has also proposed several improvements to enhance white-box exploration on the programs that rely on string operations [39,7] and lift the symbolic constraints from the byte level to the protocol level [6]. Malware analysis leverages forward symbolic execution to capture information flows through binaries [12,28,40,2]. Brumley et al. proposed MineSweeper [3] that utilizes static analysis and symbolic execution to detect trigger conditions in malware and trigger-based behavior.

2.2 Limitation of Symbolic Execution in Unrolling Loops

Most existing forward symbolic execution techniques have limitations in traversing branches in a loop, particularly when symbolic variables are used as the bound. Typically, only a fixed number of times or a fixed amount of time is spent to approximate the analysis [25,19,9,10,8,27,38,30,3]. Several approaches improve this loop unrolling strategy. LESE [32] introduces new symbolic variables to represent the number of times each loop executes and links symbolic loop variables to symbolic inputs with known input grammar. RWset [1] prunes redundant loop paths by tracking all the reads and writes performed by the checked code. We exploit this weakness of symbolic execution in handling loops to propose our novel obfuscator that uses only linear operations. In Section 6, we discuss the resilient of our obfuscator to these advancements in dealing with loops.

2.3 Binary Obfuscation

Different approaches for binary obfuscation have been developed, and the main purpose is to improve resistance to static analysis [26,31,29,35,24]. Collberg et al. performs binary obfuscation by code transformation [11]. Popov et al. obfuscates binary by replacing branch instructions with trap and bogus code [31]. Moser et al. propose opaque constants to evade static analysis. The main difference between our approach and existing work is that our goal is to impede forward symbolic execution. Sharif et al. presented an advanced work to attack symbolic execution by encrypting code that is conditionally dependent on input [35], which is the closest to our approach. However encrypting the original code introduces data bytes rarely observed. Our work introduces only linear operations and is less susceptible to statistical de-obfuscation techniques.

2.4 Unsolved Conjectures

Unsolved conjectures are unproven propositions or theorems that appear to be correct and have not been disproven. The Collatz conjecture, also known as the $3x + 1$ conjecture [23] asserts that starting from any positive integer n (see Figure 1), repeated iterations of this function eventually produces the value 1. The $3x + 1$ conjecture and its variations are simple to state but hard to be proven [15,23,20]. Conway proved that such $3x + 1$ problems can be formally undecidable [13]. The $3x + 1$ conjecture has been tested and found to always reach 1 for all integers $\leq 20 \cdot 2^{58}$ in 2009 [37].

Some other examples of unsolved conjectures that we can use in our obfuscator include

$5x + 1$ conjecture:

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \ (mod\ 2) \\ n/3 & \text{if } n \equiv 0 \ (mod\ 3) \\ 3n + 1 & \text{else} \end{cases}$$

$7x + 1$ conjecture:

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \ (mod\ 2) \\ n/3 & \text{if } n \equiv 0 \ (mod\ 3) \\ n/5 & \text{if } n \equiv 0 \ (mod\ 5) \\ 7n + 1 & \text{else} \end{cases}$$

Matthews conjecture:

$$f(n) = \begin{cases} 7n + 3 & \text{if } n \equiv 0 \ (mod\ 3) \\ (7n + 2)/3 & \text{if } n \equiv 1 \ (mod\ 3) \\ (n - 2)/2 & \text{if } n \equiv 2 \ (mod\ 3) \end{cases}$$

Juggler Sequence:

$$a_i = \begin{cases} \lfloor a_{i-1}^{1/2} \rfloor & \text{if } a_{i-1} \equiv 0 \ (mod\ 2) \\ \lfloor a_{i-1}^{3/2} \rfloor & \text{if } a_{i-1} \equiv 1 \ (mod\ 2) \end{cases}$$

These conjectures are similar to the Collatz conjecture in that they all converge to a fixed value regardless of the value of the starting integer, see Table 1.

Table 1. Convergence of unsolved conjectures

Conjecture	Convergence	Modular	Operand
$3x + 1$	1	mod 2	Integer
$5x + 1$	1	mod $2, 3$	Integer
$7x + 1$	1	mod $2, 3, 5$	Integer
Matthews	0	mod 3	Integer
Juggler	1	mod 2	Floating point, Integer

3 Overview of Our Obfuscator

Our proposed obfuscation technique complicates symbolic execution by introducing a spurious input variable and a loop from unsolved conjectures. The additional spurious input variable affects the control flow of the program in such a way that the trigger condition of the malicious behavior depends on this newly added input variable. Therefore, the additional input variable has to be modeled as a symbol in symbolic execution. A loop introduced by an unsolvable conjecture is added to the control flow, typically at the trigger condition. This loop adds a huge number of possible execution paths (growing exponentially) to the program execution, and takes symbolic execution a long time to figure out the original trigger condition.

Note that the introduction of the spurious input variable and the unsolved conjecture do not hide the malicious behavior, a goal some existing obfuscator tries to achieve [35]. What our obfuscator tries to do is to hide the condition under which the malicious behavior is triggered, but not the behavior itself, although our obfuscator could be used in conjunction with other tools to achieve both. In addition, we try to hide the trigger condition without using cryptographic operations so that the obfuscated code is less suspicious.

Figure 2 demonstrates the idea with a simple example. Figure 2a shows a simple code segment where `do_m()` is some malicious behavior with a trigger condition `x==30`. This is a program easily analyzed with symbolic execution. Figure 2b shows a code segment with a while loop. In the loop body, the variable `y` is updated in different ways according to certain condition on `y`. This code segment is hard to be analyzed by symbolic execution because the value of `y` depends on the number of times the loop body gets executed, which is hard to be figured out.

Now if we try to obfuscate the code segment in Figure 2a by introducing a spurious variable and a loop as shown in Figure 2b, we can see that the trigger condition of the malicious behavior is no longer static but depends on the spurious variable, whose value depends on the number of times the body executes (see Figure 2c). Intuitively, this is hard to analyze with symbolic execution.

However, there is one important issue we have not discussed — how do we make sure that the semantics of the program does not change after the obfuscation. In other words, although symbolic execution has a hard time figuring out the number of times the body executes in the loop, are we (as the programmer)

```
if (x == 30) {
  do_m();
}
```

(a) Malicious code with trigger condition

```
while(y > 1){
  if (y % 2 == 1){
    y = 3 * y + 1;
  }
  else{
    y = y / 2;
  }
}
```

(b) Collatz conjecture

```
// x: user input
// y: spurious variable

y = x + 1000;
while(y > 1){
  if (y % 2 == 1){
    y = 3 * y + 1;
  }
  else{
    y = y / 2;
  }
  if ((x - y > 28) &&
     (x + y < 32)){ // cond.
     do_m();
     break;
  }
}
```

(c) Obfuscated code

Fig. 2. An example

able to figure that out? Answer is yes, thanks to the unsolved conjecture as shown in Figure 1. This unsolved conjecture simply says that `y` will converge to 1 regardless of its initial value. With this, we can work out the condition (`cond.` as in Figure 2c) to be introduced in the obfuscated code under which the malicious behavior executes.

Note that our proposed obfuscator might be susceptible to pattern recognition, assuming that the unsolved conjecture we use is known. This could be solved by randomly choosing various unsolved conjectures, variations to `cond.` according to the particular unsolved conjecture used, or combining with other existing obfuscation techniques (e.g., opaque constants [29]). We discuss this further in Section 6.

4 Implementation

Having explained the basic idea as introducing a spurious input variable and adding a loop from unsolved conjectures, we turn to the implementation details in this section. Here we assume that the source code is available for obfuscation. The same idea can be easily applied to binaries since the obfuscated code we insert is simple and involves linear operations only.

4.1 Adding a Spurious Variable

In most cases, only variables derived from program inputs are taken as symbolic variables in symbolic execution [18,10,2]. We therefore have to make inserted spurious variables dependent upon program inputs.

This is not difficult since the Collatz conjecture hold regardless of the initial value of the variable. For example, if we assume that x represents a program input, our spurious variable y can be made dependent on it by $y = x + c$ where c is a constant, or $y = x + gettimeofday()$, or $y = x + rand()$ where the relation between y and x is more complicated.

However, it is not the case that the more complicated the relationship between y and x is, the longer symbolic execution takes. Symbolic execution does not support some complex operations, e.g., pointer manipulations, floating point operations, etc. When it is obvious that the variable is impossible to reason about symbolically, concrete values will be used to simplify the constraints to continue the symbolic execution. Therefore, we want to make the dependence complicated but not to the extent of being skipped by symbolic execution. We use linear polynomial with normal operations in our experiments (see Section 5). In the example shown in Figure 2, we simply use $y = x + 1000$.

4.2 Choosing an Unsolved Conjecture

Some requirements when choosing an unsolved conjecture include

- **Convergent:** the loop converges.
- **Partially decidable:** although no proof exists, it has been tested that the terminating condition is known under certain range.

- **Machine implementable:** it can be easily implemented in common programming languages.
- **Simple/Linear:** the implementation is simple and involves linear operations only.

All the examples shown in Section 2 satisfy these requirements.

Although the objective of our obfuscation is to confuse symbolic execution but not to combat pattern recognition, program analysts who know our obfuscation technique might create signatures of the unsolved conjectures and the corresponding convergence values to de-obfuscate the program. We discuss two ways of introducing variations to make such pattern recognition difficult. Section 6 also discusses the similarity of an implementation of the unsolved conjecture with the implementation of simple arithmetic algorithms.

In order to use the unsolved conjecture to obfuscate the malicious code, we need to insert a trigger condition within the loop under which the malicious behavior will execute. Intuitively, this trigger condition is related to the converge value of the conjecture (see Table 1), which is a constant regardless of the starting integer value. We can introduce variations to the trigger condition, $y == 1$ in the case of Collatz conjecture, by backtracking a few rounds before the loop actually terminates. For example, $y == 2$ can be used as the trigger condition when we backtrack one round, and $y == 4$ for two rounds.

Another variation we can introduce is for the condition of the loop. This condition is actually unimportant as long as it allows the loop to continue before reaching the trigger condition. Therefore, it can be chosen from a large number of options, including conditions on the converge value (e.g., `while (y > 1)`), conditions on the max stopping time of loops (e.g., `for (i=0; i<1000; i++)`), etc. For Collatz conjecture, the stopping times of positive integers from 1 to 2^{31} are all less than 1000.

4.3 Inserting Trigger-Based Malicious Code into the Unsolved Conjecture

Now we have introduced a new spurious variable $y = x + 1000$ (Section 4.1) and an unsolved conjecture with a trigger condition $y == 1$ (Section 4.2). The next is to insert the malicious code into the unsolved conjecture and to modify the trigger condition accordingly to preserve the semantics of the original code. Depending on the original trigger condition of the malicious code, we modify it in three different ways.

- **$>$ or $\geq$ (e.g., $x \geq 30$):** Since the spurious variable is always greater than or equal to 1 in the loop, $x - y \geq 30 - 1$.
- **$<$ or $\leq$ (e.g., $x \leq 30$):** Similarly, we have $x + y \leq 30 + 1$.
- **$==$ (e.g., $x == 30$):** This is equivalent to the intersection of two inequalities $x \geq 30 \cap x \leq 30$, and therefore we have $x+y \geq 31 \cap x-y \leq 29$ (also equivalent with the code segment shown in Figure 2c).

We implement the automatic obfuscator which takes input the original source code in C and output obfuscated code to be compiled. We apply the obfuscator on

Table 2. Overhead in size of obfuscated binaries

Malware	Size of original binary	Increase in size (bytes) after obfuscation	
		Before memory alignment	After memory alignment
Blaster	29,426	72	64
Mydoom	28,240	46	64
NetSky	36,182	60	64

three malware samples, Blaster [22], MyDoom [16], and NetSky [36] to evaluate the overhead in size in the obfuscated binaries. Table 2 shows the results.

Blaster is a worm that exploits the DCOM RPC vulnerability. It only triggers its malicious behavior (DoS attack against `windowsupdate.com`) if the system date falls into the range of Aug 16 and Aug 31. The trigger condition in the original code is implemented by two `if` statements which are both obfuscated by our technique. We find that the obfuscated binary code increases by 72 and 64 bytes in its size before and after memory alignment, respectively.

Mydoom is a mass-mailing worm that performs a DoS attack on Feb 1, 2004 starting at 16:09:18 UTC. NetSky is another mass-mailing worm that uses its own SMTP engine to send itself to the email addresses it finds on compromised systems. It only launches the attack on Oct 11 2004. We obfuscate the trigger condition (date and time) for these two malware samples, and find that the increase in size is also 64 bytes after memory alignment, which is hardly noticeable.

5 Security Evaluation

In this section, we first test the effectiveness of our obfuscation on an example with one branch condition, the running example as shown in Figure 2, to evaluate its resistance to symbolic execution. Although this example is very simple, results show that there is very little likelihood to find the trigger input by reasoning about the obfuscated code symbolically. We then continue to discuss if the evaluation results on such a simple example are general in other more complicated programs. We used a machine with a six-core processor running at 3.0 GHz and 4 GB of RAM to do symbolic execution.

5.1 Strategy Used by Program Analyzers

In order to evaluate the effectiveness of our obfuscator in confusing automatic program analyzers that employ symbolic execution, we first discuss the strategy used by the automatic program analyzer.

Recall that the objective of our obfuscation is to hide the trigger condition but not the malicious behavior. Also recall we assume that the program analyzer does not find out the trigger condition by pattern matching due to the variations we introduce to the trigger condition; see Section 4.2. An automatic program analyzer's strategy is described as follows.

1. Pick an initial program input y_0;
2. Dynamically monitor the execution of the program under the chosen input y_i. If the malicious behavior is observed, the trigger condition is found to be y_i;
3. Collect the branch conditions along the execution trace and negate the last condition on the trace;
4. Use a solver to solve for a new program input that satisfies the new sequence of conditions (with the last one negated) as well as the immediate condition of the malicious behavior (`cond.` as in Figure 2c). Let y_{i+1} equal to the new input if it can be found and go to step 2; if the new input cannot be found, negate the next (towards the start of the sequence) condition and send it to the solver until a new input can be found.

In the example as shown in Figure 2c, the trigger condition is `y == 1030` (i.e., `x == 30`). Assume that the program analyzer picks `y = 1158` as the initial input, which will result in a sequence of true/false results in evaluating the condition `y % 2 == 1` in each iteration of the loop. Table 3 shows the value of `y`, the evaluation result of the condition in some of the iterations for the trigger condition `y == 1030` and an initial input of `y == 1158`.

Table 3. Dynamic traces with an initial input of 1030 and 1158

y = 1030			y = 1158			
iteration	y	y % 2 == 1	iteration	y	y % 2 == 1	STP result
1	1030	false	1	1158	false	
2	515	true	2	579	true	
3	1546	false	3	1738	false	
...	...	...	...	...	...	
9	145	true	9	163	true	
10	436	false	10	490	false	
11	218	false	11	245	true	true
...	...	...	...	...	...	...
123	4	false	30	4	false	false
124	2	false	31	2	false	false
125	1	true	32	1	true	false

Table 3 also shows the result of the solver when the program analyzer tries to find the next input. The STP solver keeps returning false, i.e., cannot find a valid input satisfying the given condition sequence, until iteration 11. Once STP returns the next program input, the program analyzer goes back to step 2 and tries again.

This process terminates until the malicious behavior is observed and the trigger condition is found. The reason why the last condition is negated first is because we assume that the program analyzer is able to guess an initial input that is close to the trigger condition. This is a reasonable assumption since the trigger condition is usually context dependent. Under such an assumption, the program analyzer would like to choose the next y as one that results in a very similar program execution trace as the earlier one, which has a higher probability of getting closer and closer to the actual trigger condition during the experiment.

5.2 Probability of Finding the Correct Trigger Condition

Assuming that the trigger condition is unique, and the solver always manages to find the next input if there exists one that satisfies the given condition sequence (if multiple ones satisfy the condition sequence, a random one will be returned), we notice that the solver finds the next input exactly at iteration i where for all $j \leq i$ the corresponding looping condition $c_j = (y\%2 == 1)$ evaluates to the same result as in the trigger condition. In the example shown in Table 3, this means that for all $j \in [1, 10]$, c_j evaluates to the same value in both `y == 1030` and `y == 1158` while c_{11} evaluates to different values. This can be proven easily because if the solver finds the next input any earlier, it contradicts with our assumption that the trigger condition is unique.

To discuss the probability of finding the correct trigger condition with symbolic execution, we use the following notations in Table 5.2.

Table 4. Notations used

t	the trigger input (1030 in our example)
x	the program input used by the analyzer
$f(x)$	the number of iterations executed before x converges to 1
$g(x)$	largest i s.t. for all $j \leq i$, c_j is the same for t and x
$s(n)$	the number of different x s.t. $g(x) = n$
$z(x)$	the time taken to find the next input x

Table 5 shows the evaluation of some random x. Intuitively, $f(x)$ gives us an idea how long it takes to finish monitoring the execution of the program under input x. $g(x)$ evaluates how close x is with the trigger input t. The difference between $f(x)$ and $g(x)$ indicates the number of times the solver is invoked before it manages to find the next valid input x. An interesting observation here shows that $z(x)$ is not proportional to $f(x) - g(x)$. This is because the time taken for the solver depends on the complexity of the conditions. $z(x)$ is dominated by the last few tests with complex conditions (closer to $g(x)$), and is therefore mainly dependent on the value of $g(x)$.

The last two columns in Table 5 show the value of $s(n)$. Intuitively, the larger $s(n)$ is, the more likely the solver returns the next input x such that $g(x) = n$. There exists some $g(x)$ values that do not correspond to any possible x ($s(g(x)) = 0$), as shown in Figure 3. Since $s(n) = \sum_{i=n+1}^{g(t)} s(i)$, the nonzero $s(n)$ values decrease by half with the increase of n. Therefore,

$$\Pr\left(g(x_{k+1}) = g(x_k) + n\right) = \frac{1}{2^m}$$

Table 5. Statistics for different initial values of x picked

x	$f(x)$	$g(x)$	$z(x)$	$s(g(x))$	$s(g(x)+1)$
1158	32	10	19.14s	33554431	16777215
17414	142	20	878.4s	262144	131072
1049606	153	31	878.4s	4096	2048
134218758	326	43	5178.9s	32	16
2147484678	179	50	1083.6s	2	1
1030	125	125		1	

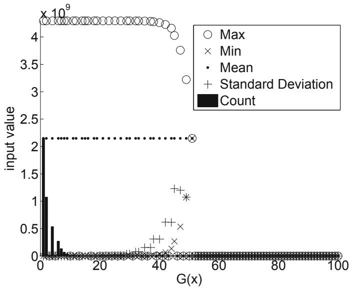


Fig. 3. Distribution of initial inputs for different $g(x)$

where m is the number of nonzero $s(g(x))$ values between $g(x_k)$ and $g(x_{k+1})$.

For example,

$$\Pr(x_{k+1} = 17414 | x_k = 1158) = 262144/33554431 = 0.0078$$

where x_{k+1} is the return of our solver after monitoring the execution with input x_k.

Appendix A shows the continuous scripts of the program analyzer for initial inputs $x = 1158$ and $x = 1034$, which confirms our intuition in the probability of $g(x_{k+1})$ shown above. This shows that it is unlikely the program analyzer gets lucky and the solver returns the trigger condition in the beginning of the study. More likely the program analyzer will take one step closer each time the solver returns an input, just like the scripts shown in Appendix A. The total times needed to find the correct trigger condition are 16871.24 and 21709.1 seconds, respectively, for the two initial inputs.

5.3 Choice of Initial Input

Section 5.2 shows that the program analyzer most likely will take one step closer to the trigger condition t (with an input x that has a slightly larger $g(x)$) every time the solver returns an input for some particular initial input. In this section, we show that the result presented in Section 5.2 can be generalized to other initial inputs. Figure 3 shows the distribution of initial inputs $x \in [1, 2^{32}]$ for different $g(x)$ for the trigger condition $t = 1030$. Appendix B shows that similar distribution is found for eight random values of t.

It is obvious from Figure 3 that there are more initial values with smaller $g(x)$. The number of initial input values continues to drop until it reaches zero for

$g(x) > 50$, with a single exception when $x = t$ which results in $g(x) = g(t) = 125$. Recall that $g(x)$ is an indication of how close x is with t, this means that there are fewer possible values of x when it comes closer to t, which means that the strategy of randomly picking other values of x does not usually give the program analyzer an advantage in finding t. The strategy shown in Section 5.2 is still a reasonably good strategy.

Looking at the mean values, we notice that it is not a continuous line, although the mean is always around 2×10^8. It is not a close line mainly because there exists many $g(x)$ values that do not correspond to any possible x. As we explained earlier, there are many scenarios where the solver might not be able to find any inputs. A closer look into the original data reveals that the mean is always around 2^{16}. This shows that there is very little bias in the distribution of x when t is small, and therefore the program analyzer could not get much advantage by choosing smaller/larger initial values.

This analysis shows that choose different initial inputs does not give the program analyzer significant advantages, and the analysis results in Section 5.2 can be extended to different initial inputs, as well as different trigger conditions (see Appendix B).

6 Limitations

Our obfuscator is designed to make symbolic execution difficult in finding out a trigger condition of malicious code. We show its effectiveness in some examples and its security in Section 5. However, this obfuscator is not designed to solve all obfuscation problems and there are some limitations to it.

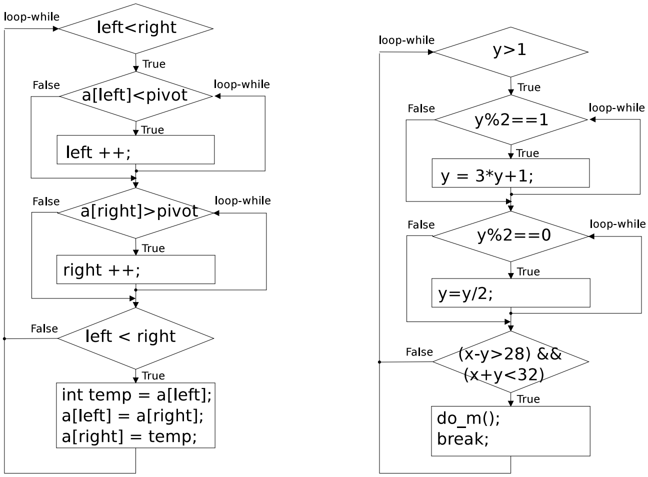
Constants. Our obfuscator is not designed to obfuscate constants. In fact, we introduce additional constants into the obfuscated code. To handle this problem, our obfuscator can be used in conjunction with opaque constants [28] to hide special characteristic of the obfuscation.

Malicious behavior. Our obfuscator is not designed to hide the malicious code, but the condition under which the malicious code will be executed. A malware author can introduce existing code mutation techniques, such as polymorphism and metamorphism, to make it difficult to analyze the malicious behavior.

Pattern matching. The unsolved conjectures introduced by our obfuscator might introduce special patterns that can be identified. Besides using different conjectures shown in the section 2.4 and introducing variations as discussed in Section 4, here we show that the control flow of our unsolved conjectures is very similar to some common program algorithm, which makes pattern matching difficult.

Figure 4 shows that the control flow of the two code segments are similar. We also use one of the most sophisticated binary difference analyzer, BinHunt [17] to analyze the similarity of various binary codes, and show the results in Table 6. Results show that our obfuscated code is very similar to the code of quick sort.

Larger set of triggered inputs. In our analysis we assume that there is a single integer that satisfies the trigger condition, and show that symbolic execution has



(a) A quick sort algorithm (b) Our obfuscated code

Fig. 4. Control flow comparison

Table 6. Binary difference analysis (larger matching strength indicates higher similarity)

Matching strength	Our obfuscated code	Select sort	Bubble sort
Quick sort	0.85	0.49	0.38

a hard time figuring it out. However, the probability results may change when there is a larger set of inputs that satisfy the trigger condition.

Execution overhead. Our obfuscator introduces some additional overhead to the execution of the program due to the loop added. This is usually not a concern when it is applied in a malicious program. However, it may be an issue when the technique is used to obfuscate legitimate programs.

7 Conclusion

In this paper, we introduce a novel obfuscator that makes symbolic execution difficult in finding trigger conditions. Our obfuscator applies the concept of unsolved conjectures and adds a loop to the obfuscated code. Experiments show that symbolic execution will have a hard time unrolling the loop and therefore inefficient in figuring out the trigger condition under which certain code segment will be executed. Our security analysis shows that there does not exist other analyzing strategy in making the analysis simpler, even when different initial inputs are used or when the trigger condition is different.

Acknowledgments. The authors thank the anonymous reviewers for their suggestions. This work was supported by the National Natural Science Foundation of China under grant 60973141, the Natural Science Foundation of Tianjin under grant 09JCYBJ00300 and the Specialized Research Fund for the Doctoral Program of Higher Education of China under grant 20100031110030.

References

1. Boonstoppel, P., Cadar, C., Engler, D.: RWset: Attacking path explosion in constraint-based test generation. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 351–366. Springer, Heidelberg (2008)
2. Brumley, D., Hartwig, C., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Song, D., Yin., H.: Bitscope: Automatically dissecting malicious binaries. Technical report cs-07-133, School of Computer Science, Carnegie Mellon University (March 2007)
3. Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Poosankam, J., Song, D., Yin, H.: Automatically identifying trigger-based behavior in malware. In: Botnet Analysis in Defense, vol. 36 (2007)
4. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: Proceedings of the 2006 IEEE Symposimu on Security and Privacy (2006)
5. Brumley, D., Wang, H., Jha, S., Song, D.: Creating vulnerability signature using weakest preconditions. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy (2007)
6. Caballero, J., Liang, Z., Poosankam, P., Song, D.: Towards generating high coverage vulnerability-based signatures with protocol-level constraint-guided exploration. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (2009)
7. Caballero, J., McCamant, S., Barth, A., Song, D.: Extracting models of security-sensitive operations using string-enhanced white-box exploration on binaries. Tech. rep., Technical Report UCB/EECS-2009-36, EECS Department, University of California, Berkeley (March 2009)
8. Cadar, C., Dunbar, D., Engler, D.: Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 2008 USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008 (2008)
9. Cadar, C., Engler, D.: Execution generated test cases: How to make systems code crash itself. In: Proceedings of the 12th SPIN Workshop (2005)
10. Cadar, C., Ganesh, V., Pawlowski, P., Dill, D., Engler, D.: EXE:automatically generating inputs of death. In: Proceedings of the 2006 ACM Conference on Computer and Communications Security (CCS 2006) (2006)
11. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical report 148, Department of Computer Sciences, The University of Auckland (1997)
12. Comparetti, P.M., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., Zanero, S.: Identifying dormant functionality in malware programs. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy (2010)
13. Conway, J.H.: Unpredictable iterations. In: Proceedings of the 1972 Number Theorey Conference (1972)
14. Costa, M., Castro, M., Zhou, L., Zhang, L., Peinado, M.: Bouncer: Securing software by blocking bad input. In: Proceedings of the 2007 ACM Symposium on Operating Systems Principles (SOSP) (2007)

15. Crandall, R.E.: On the "3x + 1" problem. Mathematics of Computation 32, 1281–1292 (1978)
16. Ferrie, P.: W32.Mydoom (2004), `http://www.symantec.com/security_response/writeup.jsp?docid=2004-012612-5422-99&tabid=2`
17. Gao, D., Reiter, M.K., Song, D.: BinHunt: Automatically finding semantic differences in binary programs. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 238–255. Springer, Heidelberg (2008)
18. Godefroid, P., Klarlund, N., Sen, K.: DART: Directed automated random testing. In: Proceedings of the ACM Conference on Programming Lanuguage Design and Implementation (2005)
19. Godefroid, P., Levin, M.Y., Molnar, D.: Automated whitebox fuzz testing. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008) (2008)
20. Guy, R.K.: Unsolved problems in number theory. Problem Books in Mathematics (2004)
21. King, J.C.: Symbolic execution and program testing. Commun. ACM 19, 385–394 (1976), `http://doi.acm.org/10.1145/360248.360252`
22. Knowles, D., Perriott, F.: W32.Blaster (2003), `http://www.symantec.com/security_response/writeup.jsp?docid=2003-081113-0229-99&tabid=2`
23. Lagarias, J.C.: The 3x+1 problem and its generations. Amer. Math. Monthly 92, 3–23 (1985)
24. Lee, B., Kim, Y., Kim, J.: binOb+: a framework for potent and stealthy binary obfuscation. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (2010)
25. Lee, G., Morris, J., Parker, K., Bundell, G., Lam, P.: Using symbolic execution to guide test generation. Software Testing, Verification & Reliability 15(1), 41–61 (2005)
26. Linn, C., Debray, S.: Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (2003)
27. Molnar, D., Li, X., Wagner, D.: Dynamic test generation to find integer bugs in x86 binary linux programs. In: Proceedings of the 2009 USENIX Security Symposium (2009)
28. Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proceedings of the 2007 USENIX Security Symposium (2007)
29. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (2007)
30. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: Automatic protocol replay by binary analysis. In: Proceedings of the 13th ACM Conference on Computer and and Communications Security (CCS 2006) (2006)
31. Popov, I.V., Debray, S.K., Andrews, G.R.: Binary obfuscation using signals. In: Proceedings of the 2007 USENIX Security Symposium (2007)
32. Saxena, P., Poosankam, P., McCamant, S., Song, D.: Loop-extended symbolic execution on binary programs. In: Proceedings of the 18th International Symposium on Software Testing and Analysis (2009)
33. Schwartz, E.J., Avgerinos, T., Brumley, D.: All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: Proceedings of the 2010 IEEE Symposium on Security and Privacy (2010)

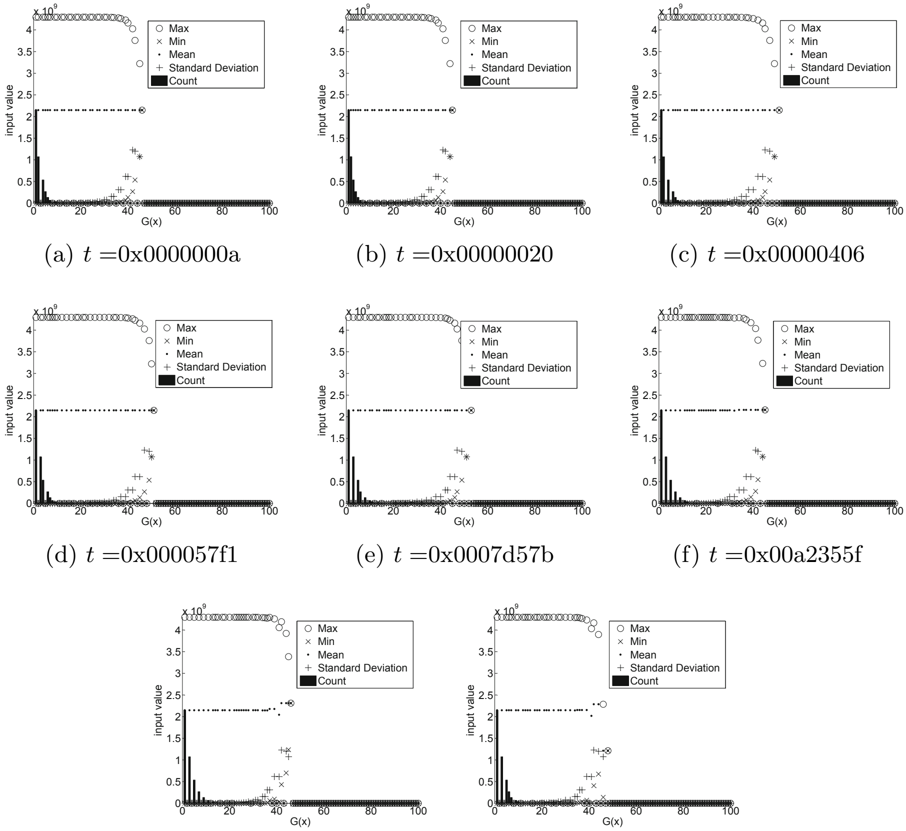
34. Sen, K., Marinov, D., Agha, G.: CUTE: A concolic unit testing engine for c. In: Proceedings of 13th International Symposium on the Foundations of Software Engineering, FSE 2005 (2005)
35. Sharif, M., Lanzi, A., Giffin, J., Lee, W.: Impeding malware analysis using conditional code obfuscation. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008) (2008)
36. Shinotsuka, H.: W32.NetSky (2004), `http://www.symantec.com/security_response/writeup.jsp?docid=2004-030717-4718-99&tabid=2`
37. Silva, T.O.: Computational verification of the 3x+1 conjecture. Tech. rep., Electronics, Telecommunications, and Informatics Department,University of Aveiro (November 2010), `http://www.ieeta.pt/~tos/3x+1.html`
38. Wang, T., Wei, T., Lin, Z., Zou, W.: Intscope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution. In: Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009) (2009)
39. Xu, R.G., Godefroid, P., Majumdar, R.: Testing for buffer overflows with length abstraction. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis (2008)
40. Yin, H., Song, D.: Panorama: capturing system-wide information flow for malware detection and analysis. In: ACM Conference on Computer and Communications Security (CCS 2007) (2007)

A Contineous Scripts of the Program Analyzer When $x = 1158$ and $x = 1034$

Table 7. Complexity of symbolic formulas and average solving time

Round	x	g(x)	f(x)	z(x)	# of STP nodes	x	g(x)	f(x)	z(x)	# of STP nodes
1	1158	10	32	19.14	19107	1034	3	125	707.6	127543
2	1286	11	27	13.6	17613	1022	5	63	127.6	45602
3	1542	13	35	24.2	24076	550	7	93	464.4	124144
4	2054	14	38	31.2	26042	1222	8	40	38.4	24682
5	3078	15	36	27.3	26119	646	10	101	345.8	65050
6	5126	17	55	72.2	40000	4358	11	47	64.8	29976
7	9222	18	110	469.2	84462	518	13	124	621.6	125150
8	17414	20	142	878.4	110404	7174	15	120	598.5	128339
9	33798	22	60	98.8	44059	25606	18	127	654	131941
10	66566	24	56	70.4	41148	50182	20	66	124.2	57248
11	132102	26	101	345	76491	99334	22	116	413.6	85067
12	263174	28	102	288.6	86298	197638	24	148	892.8	145560
13	525318	29	90	244	68720	4326406	26	181	1286.5	150866
14	1049606	31	153	878.4	118753	1573894	29	76	141	58221
15	2098182	32	105	350.4	76268	6292486	32	228	1999.2	185416
16	4195334	34	137	710.7	105035	12583942	34	105	319.5	81474
17	8389638	36	169	824.6	147493	25166854	36	230	1978.8	188020
18	16778246	37	82	126	71580	100664326	39	139	750	138716
19	33555462	39	171	1188	129648	1946158086	41	135	639.2	137371
20	67109894	41	172	956.3	146783	2013266950	43	183	1162	161645
21	134218758	43	326	5178.9	281345	1879049222	44	306	4034.8	243945
22	268436486	44	174	975	147574	1610613766	46	249	2151.8	199218
23	536871942	46	175	993.3	148022	3221226502	48	177	1109.4	153689
24	1073742854	48	176	1024	147881	2147484678	50	179	1083.6	150026
25	2147484678	50	179	1083.6	150026					
Sum of z(x)				16871.24					21709.1	

B Distribution of Initial Inputs for Different Trigger Input t



(a) t =0x0000000a (b) t =0x00000020 (c) t =0x00000406

(d) t =0x000057f1 (e) t =0x0007d57b (f) t =0x00a2355f

(g) t =0x09c45b3f (h) t =0xc8435f73

DriverGuard: A Fine-Grained Protection on I/O Flows

Yueqiang Cheng, Xuhua Ding, and Robert H. Deng

School of Information Systems,
Singapore Management University
{yqcheng.2008,xhding,robertdeng}@smu.edu.sg

Abstract. Most commodity peripheral devices and their drivers are geared to achieve high performance with security functions being opted out. The absence of security measures invites attacks on the I/O data and consequently threats those applications feeding on them, such as biometric authentication. In this paper, we present the design and implementation of DriverGuard, a hypervisor based protection mechanism which dynamically shields I/O flows such that I/O data are not exposed to the malicious kernel. Our design leverages a composite of cryptographic and virtualization techniques to achieve fine-grained protection. DriverGuard is lightweight as it only needs to protect around 2% of the driver code's execution. We have tested DriverGuard with three input devices and two output devices. The experiments show that DriverGuard induces negligible overhead to the applications.

1 Introduction

Device drivers are often blamed as the main cause for system failures and security breaches, mainly due to their enormous code size and the much higher bug rate than other kernel code [5]. Various schemes have been proposed to improve system reliability by isolating driver errors (e.g., Nook [25] and SafeDrive [32]), or to defend against device I/O misuses for illegal memory accesses (e.g., BitVisor [24] and schemes in [29]). In this paper, we study the other side of the coin: how to protect driver operations, which is motivated by attacks on sensitive I/O data, such as password keystrokes, fingerprint templates, sensor readings and confidential print-outs.

As compared to applications and other kernel components such as system call functions, driver operations or I/O flows are more attractive to malware targeted at sensitive data for the following reasons. Firstly, there exist more loopholes to exploit due to the complexity of I/O mechanisms and the abundance of driver bugs. For instance, IRQ number sharing allows a malicious interrupt handler to easily access another handler's data. Another reason is that more drivers handle raw data generated by or for hardware. In many applications, raw data are more favorable to attackers as compared to derived data. For instance, a user's fingerprint template is life-long valid whereas a secret key derived from the fingerprint template may remain valid only for a few hours. Furthermore,

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 227–244, 2011.

most commodity I/O devices nowadays are not encryption capable and raw data are exposed to any code accessing them.

We aim to protect application-device data flows against the untrusted kernel throughout the entire I/O lifecycle. In particular, we focus on those devices that render raw data, e.g., sound cards and printers, or generate raw data for applications, e.g., seismic sensors and fingerprint scanners. We are less concerned with disks and network adaptors, because these devices deal with derived data from applications. Therefore, a simple solution to protect disk I/O and network I/O is to encrypt the application data before and after I/O operations. In this work, we present DriverGuard, a holistic and compact I/O protection system making use of a combination of cryptographic and virtualization techniques. We have implemented DriverGuard with slight changes on the drivers and the hypervisor. Our experiments with several I/O devices demonstrate that DriverGuard imposes little overhead to the system and causes unnoticeable delays to user applications. DriverGuard is complementary to many user space protection schemes such as BIND [23], Overshadow [4], PRECIP [26] and Terra [8]. A composition of DriverGuard and a user-space protection scheme can protect the whole lifecycle of data processing.

Our work is remarkably different from secure I/O [16,24,29] and driver code security. Secure I/O copes with those attacks misusing the I/O mechanism (especially DMA operations) for illegal memory accesses. Driver code security tackles software attacks, such as return-address attacks [3] and code injection attacks [11], which gain the root privilege by attacking drivers. Although these attacks do not necessarily target on I/O data, they are one of the threats considered in our study.

Organization. The next section discusses the related work. We present the design of DriverGuard in Section 3, and the implementation details in Section 4. Section 5 shows the evaluation of DriverGuard through our experiments. We conclude the paper in Section 6.

2 Related Work

Data Flow Security. BIND [23] binds data and code and uses cryptographic techniques to guarantee the integrity of data. However BIND is limited to derived data. TERRA [8] builds an application specific domain with a trusted path from the hypervisor to an application specific kernel, then to the application. Trusted path schemes [10,31] focus on providing a trusted GUI to user, protecting user inputs to the intended applications. These two schemes only address security issues at the driver-applications interface, whereas the battlefield of DriverGuard is the entire I/O path. Bumpy [14] proposes to protect user keyboard inputs by building a trust environment using Flicker [13]. It requires an encryption-capable keyboard and therefore is not applicable to generic devices.

Secure I/O. Most existing results on secure I/O deal with I/O misuses where an adversary attacks the system by exploiting the flexibility of I/O operations, especially DMA. The schemes described below serve a different purpose from ours and are not applicable to I/O flow protection. *d*Anubis [16] is a system monitoring and analyzing device drivers using virtual machine introspection techniques. BitVisor [24] is a hypervisor dedicated to I/O management and supports only one VM. It uses a parapass-through mechanism whereby most I/O operations from the guest pass through the hypervisor with some of them being intercepted. The interception allows the hypervisor to protect itself and to perform security functions. DMA security receives more attention since DMA-capable devices can access memory without involving the CPU. In [30], a software based approach is proposed whereby the hypervisor validates all DMA descriptors before they are issued to the device. This approach is then extended to a hardware-based approach by utilizing I/O memory management units (IOMMUs) in [29], which deals with the bad-address fault, the invalid-use fault and the bad-device fault.

Hypervisor-Based System Security. Our scheme is also relevant to hypervisor-based security systems. SecVisor [21] utilizes a small hypervisor to prevent kernel code injection. Overshadow [4] protects device memory from untrusted software in user space. In HyperShield [17], a thin layer hypervisor is plugged into a running OS without rebooting, so that it prevents illegal code execution. Lares [19] is an architecture which establishes a secure environment for security tools to actively monitor a guest domain. HookSafe [28] uses a hypervisor to prevent kernel hooks from being hijacked. TrustVisor [12] is a tiny trusted computing base which protects code and data integrity by leveraging hardware features. Although these schemes take the rootkit as the adversary, they only provide a generic protection, not geared for I/O protection. Therefore, an attack on the I/O path might not be considered as adversarial in these schemes.

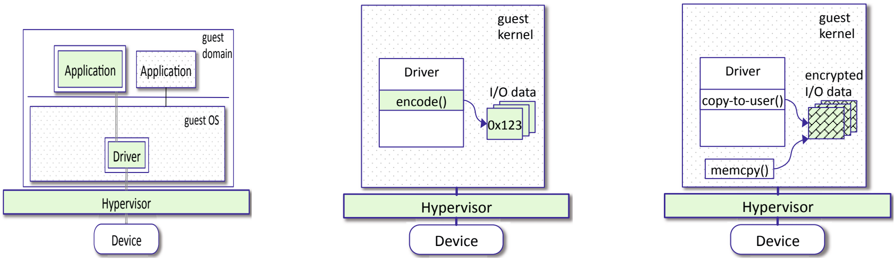
3 Design of DriverGuard

The objective of DriverGuard is to protect the confidentiality of a driver's I/O data from being attacked by a corrupted kernel. We remark that since I/O operations are heavily used, the low-overhead requirement is vital for the practicality of DriverGuard.

3.1 Trust Model

The bedrock of our scheme is a trusted hypervisor beneath the guest domain. Although there are known rootkit attacks on the hypervisor, we suppose that secure boot-up and load-time attestation with the support of TPM [9] can ensure the hypervisor's security in the bootstrapping phase. The hypervisor then loads itself into an isolated memory region with the highest privilege so as to block any illegal accesses from a guest domain [1]. Other techniques such as HyperSafe [27] can also be applied to ensure the hypervisor's security. We assume the presence of IOMMU protecting the hypervisor's memory territory from being invaded by DMA devices under the adversary's control.

We do not trust the guest kernel since it is vulnerable to various attacks such as return-oriented attacks [22,2,3] and code injection [11,21]. In our attack



(a) An illustration of the trust model. The colored boxes represent trusted components.

(b) A PCB accesses the I/O data in plaintext.

(c) A non-PCB, e.g., the memcpy() function, accesses the encrypted I/O data.

Fig. 1. Illustrations of DriverGuard's trust model and the concept of PCB

model, we consider the subverted guest kernel as the adversary whose goal is to acquire I/O data transferred by a driver. The malicious kernel can read or write any memory region and any I/O port within the guest domain, but can not subvert the hypervisor. The driver is treated as a benign executable which actively demands I/O protection. We consider the scenarios that I/O requests are issued from an application well-protected in user space. Existing schemes such as Overshadow [4] can safeguard the application data against attacks from other applications and the guest kernel. Figure 1(a) depicts the trust model used in this paper. Note that hardware attacks such as bus sniffing are not investigated in this paper. Neither is the denial-of-service attack whereby the adversary attempts to sabotage I/O flows, instead of compromising the data.

Attacks. The attacks from the corrupted kernel are classified into two categories. The malicious kernel can attack the device I/O controls such that the device receives a manipulated I/O command and reads/writes data from/to regions to the adversary's advantage. This type of attacks requires the kernel to tamper with I/O control related regions such as I/O ports and DMA descriptors to inject commands or to modify the control region locations. The other type of attacks are directly targeted on the I/O data. The kernel can attempt to access data regions such as I/O data ports, MMIO data buffers, DMA buffers, the driver's memory regions and the application's data buffer. Alternatively, the kernel can attack the driver's execution flow or runtime states, such as the stack.

3.2 Design Rationale

A straightforward approach is that the hypervisor arbitrates whether a control flow can access the I/O data. It requires the hypervisor to introspect driver operations, which is difficult to implement due to the semantic gap (e.g., lack of details of driver operations) between the hypervisor and the driver. Considering the complexity of I/O operations, the workload on the hypervisor will significantly downgrade the whole system performance.

Isolation is a widely used method to protect program executions. To apply isolation on driver protection, one may propose location isolation or execution isolation. Location isolation is to place drivers and the kernel's I/O subsystem a separated domain, e.g., an I/O domain and Domain 0, or the hypervisor's space, such that malware in the guest kernel can not attack them directly. These approaches are efficient in terms of I/O performance. Nonetheless, the resulting protection is weak because the TCB size is increased significantly due to the drivers and the I/O subsystem.

In execution isolation, the drivers still reside in the untrusted guest kernel while their executions are in a secure environment established by the hypervisor, similar to TrustVisor [12] and Overshadow [4]. The generic execution isolation is not applicable for driver protection, because I/O operations are featured with frequent hardware interrupts and intensive driver-kernel interactions. Note that if the I/O subsystem is also enclosed in execution isolation, it suffers from the same drawback as in the location isolation approach.

We adopt the idea of execution isolation, however, at a micro-level. It is well-known that most of the driver code is for housekeeping purposes, such as error handling, resource allocation and cleaning up [7], with only a small portion dealing with I/O data transferring. We further observe that among the code for data transferring, only a few code blocks, e.g., an encoding function, need to process the I/O data while the majority of them just move the data from one memory location to another without necessarily knowing the content. Based on these observations, we design DriverGuard as a fine-grained I/O protection mechanism which distinguishes those security-sensitive driver code from the rest. The hypervisor only needs to protect the executions of security-sensitive code blocks (around 1% of the driver code according to our experiments) because of the aforementioned driver code characteristics. The fine-grained protection is coupled with a hypervisor-based access control mechanism. Different from hypervisor introspection, access control does not impose comprehensive logics on the hypervisor. Hence, the overall cost of DriverGuard is remarkably low. Its performance is on par with the location isolation solution, however, the security strength is much stronger.

3.3 Overview

By and large, DriverGuard is constructed using three lightweight protection techniques as the building blocks: cryptography, access control and runtime protection. We use cryptographic techniques to protect all I/O data without interfering with most of the driver and the kernel executions. For regions holding data which cannot be protected by encryption, we resort to the hypervisor to enforce access control. These plaintext data can only be accessed by a few designated driver code blocks, whose executions are safeguarded by our runtime protection mechanism. We refer to these code blocks as *privileged code blocks* (PCBs) in the rest of the paper. By protecting the execution of PCBs, we successfully ensure the whole I/O data security with minimal overhead since PCBs only constitute a tiny fraction of the driver code.

Privileged Code Block. We consider three types of PCBs in a driver. One type of PCBs is the driver code blocks which make computation on the I/O data, e.g., an encoding function. We call them *computation-PCBs*. The second type of PCBs is the driver code blocks which issue I/O commands and parameters to a device. We call them *command-PCBs*. This type of code is security sensitive because their executions determine the locations of plaintext I/O data. The third type of PCBs is the driver code blocks which initialize the driver's cryptographic key. Each driver generates its own key in the driver initialization step, such as in `module_init`. We call them *key-PCBs*.

The critical property of PCBs is that they must access critical information in plaintext. It is desirable for a PCB's size to be small without making any function call to non-PCBs, because non-PCBs are unprotected and may compromise security. When a driver is loaded, the hypervisor is notified with the locations of the driver PCBs and sets them as read-only in order to protect the code integrity. A PCB is delimited by two hypercalls to request for and relinquish runtime protection. The runtime protection of a PCB means that when the PCB is scheduled off from the CPU, the hypervisor seals its context and cordons off all accesses to the data and states until it resumes its CPU control.

A high level view of DriverGuard's protection mechanism is as follows. A driver initially generates a secret encryption key in its key-PCB. The I/O data remains encrypted by this key whenever the guest domain's virtual CPU is not controlled by the driver's PCB. Within a computation-PCB, it may perform encryption, decryption or encoding functions on the data. If a computation-PCB's decrypts the data, it either re-encrypts it or requests the hypervisor's protection when it ends. For I/O controls, the hypervisor ensures that the device's I/O ports or MMIO regions can only be accessed by the driver's command-PCBs. With the assistance of the hypervisor, the command-PCB checks whether the I/O buffer address in use is legitimate before issuing the command to the hardware. Figures 1(b) and 1(c) illustrate the difference between a PCB's and a non-PCB's I/O data accesses. Next, we explain the design of three building blocks and leave the discussion of their integration in Section 4, since it involves the details of I/O operations.

3.4 Access Control over Critical Regions

Since we do not rely on encryption-capable devices, encryption is not applicable for data accessed by the hardware. To cordon off illicit accesses to those data, we utilize the hypervisor's access control mechanism. In general, there are two types of regions for access control: the data regions and the control regions. The former holds the raw data generated for or by the hardware while the latter holds the I/O parameters for the hardware. According to their address spaces, these regions are classified into memory regions and I/O ports, for which we apply different access control methods by leveraging the hardware features and the virtualization techniques available in the platform.

To intercept accesses to a protected memory region, DriverGuard sets the attribute bits in the corresponding page table entries (PTEs), while to intercept accesses to an I/O port, it clears the corresponding IOPL bits and sets up the I/O bitmap. We use *checkpoints*[1] in the rest of the paper to refer to both the IOPL bits and the PTEs marked by the hypervisor for the purpose of access interception. Although the aforementioned protection techniques are used in many existing schemes, e.g., [4, 19], we are confronted with two new problems. First, given a memory buffer address, the hypervisor must make sure that the kernel can not bypass the checkpoint to access the region, which is challenging for memory regions allocated by the kernel. Secondly, the hypervisor must ensure that the sensitive I/O data is indeed placed in the region *with* a checkpoint. The first problem demands a careful page table walk checking while the second demands the I/O control integrity checking. We will present our solutions to both problems in the next section.

3.5 Cryptographic Components

We introduce to the guest kernel a symmetric-key encryption function and a decryption function, both of which can be called by any code. However, any write access to these function's code is denied by the hypervisor. We also add a key generation function to the driver as a PCB. The security of the I/O data relies on the secrecy of the driver's key, rather than the secrecy of the decryption function, which complies with the famous Kerckhoffs principle. The driver's secret key is securely generated based on a secret random seed supplied by the hypervisor. The secret key is securely stored in a kernel space buffer priorly appointed by the driver and can only be accessed by the driver's PCBs escorted by the hypervisor. This prevents any unauthorized code from decrypting the driver's data, even though the decryption function can be called arbitrarily.

3.6 PCB Execution Escorting

The third building block in DriverGuard is the runtime protection mechanism that prevents a PCB's execution from deviating its expected behavior. The protection is requested at the PCB's entry and is relinquished at the exit via hypercalls. The hypervisor agrees to admit a control flow into the escorting only when the request is issued from the driver's PCB, and agrees to discharge a flow from escorting only when the request is issued from the PCB presently under escorting.

The PCB under the escorting is granted to access the critical data such as the driver's secret key and the I/O data, or to issue I/O commands. In our design, the hypervisor lifts the checkpoints on those regions accessed by the PCB, and restores them at the exit of escorting. Therefore, no duplicated exceptions or page faults will be raised despite that the PCB may access the same region multiple times within an escorted execution. An escorted PCB can be scheduled off from the CPU for various reasons. In that case, the hypervisor intercepts this event and restores all checkpoints. Meanwhile, it also securely saves the driver's

[1] Our definition of *checkpoint* has no relation with the *checkpoint* for rollback in distributed systems.

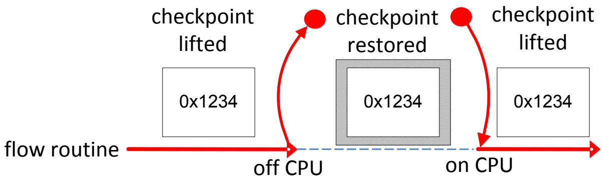


Fig. 2. An illustration of runtime protection, where $0x1234$ is an exemplary memory address with a PTE checkpoint

runtime stack and sets up a breakpoint for the PCB's upcoming CPU occupation. As a result, other code's accesses to the protected regions are denied. Figure 2 depicts a scenario of escorting. There are two methods for a PCB to restore the protection on the data. A computation-PCB encrypts the data before it exits from protection whereas a command-PCB requests the hypervisor to block all accesses on the region. In the end, a hypercall is invoked by a PCB to relinquish escorting.

4 Implementation

We build DriverGuard on top of the Xen hypervisor to protect the drivers running in a Linux guest domain. We systematically examine every step in I/O operations, from the device discovery to the application's (or device's) data fetching. In order to adaptively protect the driver operations, the hypervisor needs to store certain context information about the driver. We start with driver context initialization since it is performed by the hypervisor during the guest domain bootstrapping.

4.1 Driver Context Initialization

The context information of drivers is securely stored in three types of tables in the hypervisor space. A *device table* specifies the management relation between a driver and a device by paring their identifiers. For every protected driver, the hypervisor maintains a *PCB table* and a *region table*. The former stores the entry and exit addresses of all PCBs of the driver while the latter specifies the memory regions and the I/O ports to protect. There are five types of regions in the region table: 1) the application buffer; 2) the memory buffer allocated by the driver for data processes; 3) the I/O data buffer such as DMA buffers; 4) the control regions, including the I/O ports or MMIO regions for device control, DMA descriptor queues, the transfer descriptor queues; and 5) the buffer holding the driver's secret key.

Device Table Initialization. When a guest kernel image is uncompressed, the hypervisor inserts a hook function to the kernel to inform the hypervisor about the device-driver association via a hypercall. The hypervisor then initializes the device table accordingly. The hypervisor also sets the checkpoints for the kernel structure maintaining the device-driver association. Whenever a driver takes the ownership of a device, the hypervisor intercepts the event and updates the device table properly.

PCB Table Initialization. The hypervisor also scans the driver code to obtain the locations of PCBs. It records the addresses of escorting-entry hypercalls and the addresses for escorting-relinquish hypercalls, and puts these pairs into the PCB table. The driver's initial code are considered as intact and the scanned PCBs are therefore legitimate.

Region Table Initialization. The control regions used by a driver can either be the default ones chosen by the manufacturer/the kernel or set by the driver. In the first case, the hypervisor updates them when the driver is loaded as in the device discovery step. In the latter case, the driver informs the hypervisor via a hypercall about the protected regions or I/O ports.

4.2 Checkpoint Deployment

Given a memory region or an I/O port, the hypervisor sets up the corresponding checkpoint to intercept and examine risky accesses. The detailed deployment method is dependent on the virtualization environment. Due to the length limit, we focus on deployment for paravirtualization domains and do not elaborate the techniques in a hardware virtualization domain (HVM).

Memory Region Checkpoint. For a memory page P, the hypervisor walks through the page tables according to CR3 register to locate the PTE pointing to P. The hypervisor can set the attribute bits on the PTE to specify different access rights. To set a page "read-only", the _PAGE_RW bit is cleared; and to set a page "non-access", the _PAGE_PRESENT bit is cleared.

Because all protected regions are in kernel space, PTE updates are propagated into the kernel portion of all other page tables in order to maintain consistency. Note that in the paravirtualization setting, only the hypervisor updates page tables. Therefore, those checkpoints will not be removed by the guest kernel. The hypervisor also checks page tables to remove double mappings pointing to protected memory regions using the method in HyperSafe [27].

I/O Port Checkpoint. I/O ports do not belong to the memory address space. During the launch of a paravirtualization domain, the hypervisor clears the IOPL bits of EFLAGS of the guest's virtual CPU. Namely, it sets the I/O privilege level to 0, such that the hardware always checks the I/O bitmap for PIO instructions because the guest kernel runs in Ring 1. Then, the hypervisor sets the bits corresponding to the protected I/O ports such that a PIO instruction will cause a general protection exception.

4.3 PCB Execution Escorting

PCB Admission. A driver's PCB starts with the hypercall which takes as the parameter the buffer address it requests to access. To admit a PCB, the hypervisor checks whether the hypercall is issued from the instruction whose address is registered in the PCB table. If not, the hypervisor rejects the request.

For an admitted PCB, the hypervisor has two tasks. One is to set a *local* breakpoint at function `__switch_to`, which is the kernel function for a CPU context switch. The other task is for stack protection. The hypervisor allocates a dummy stack for the PCB. Therefore, an admitted PCB has two runtime stacks. A genuine stack is used for the PCB's execution while the dummy stack is used for untrusted code sharing the same execution flow due to interrupts. Figure 3 below describes the details of the PCB admission algorithm, where InEscorting is a flag bit indicating the current execution state.

Admission Algorithm

```
1) Fetch the EIP value stored at the top of the current guest kernel stack, which is the
return address of the hypercall.
2) If EIP does not match any entry in the PCB table, return error.
3) If the address of requested buffer is legitimate, then
      a) set InEscorting to 1;
      b) set a breakpoint at the entry of __switch_to function.
      c) If the guest's kernel stack segment is not a dummy stack, then
            (i) allocate a dummy stack at the reserved space.
            (ii) save the machine addresses of the dummy stack and the present stack as
$(MA'_{ss}, MA_{ss})$. Return 0.
      d) else, switch to the corresponding genuine stack. Return 0.
6) Return -1 as an error message for admission failure.
```

Fig. 3. Algorithm for PCB admission

Escorting. Once a PCB is admitted by the hypervisor, its execution is escorted and the checkpoints for the buffers it accesses are temporarily lifted. The essence of escorting is that the hypervisor intercedes whenever the PCB is scheduled off from the CPU, which takes place in two scenarios. One is that the PCB relinquishes the CPU and the other is due to hardware interrupts. Both cases open the door to attacks. We design a mechanism to intercept the CPU context switch and to use dummy stacks for untrusted control flows. The interception is via the interrupt handler and the exception handler as explained below.

Interrupt. To switch to a dummy stack, the hypervisor only replaces the content of the PTE for the present stack with the machine page number of the dummy stack allocated during admission. This change is transparent to any guest process, since the address in ESP register remains the same. Hence, the guest kernel is not able to access the true stack while the subsequent execution can use the dummy stack without being affected. The algorithm for stack switching and checkpoint restore is shown in Figure 4.

Interrupt Handler Algorithm

(1) If $InEscorting = 0$, return.
(2) Restore the checkpoints that are removed during escorting.
(3) Switch to the dummy stack, by setting the PTE for the guest's stack base to point to MA'_{ss}.
(4) Set $InEscorting = 0$. Remove the breakpoint at `__switch_to` function.
(5) Set a local breakpoint at the instruction pointed by EIP. Save the address pair in EIP and ESP.
(6) Return and pass the control to the default interrupt handler.

Fig. 4. Interrupt handler algorithm for stack switching and checkpoint restore

Debug Exception. All breakpoints used by the hypervisor are *local* breakpoints. Therefore, they are triggered only for the present process. There are two types of local breakpoints used in DriverGuard. One is for the CPU context switch interception. For this breakpoint, the hypervisor exits from escorting and restores checkpoints, in a similar fashion to the interrupt handling.

The other type of breakpoints is to intercept the event of PCB resuming. For this type of breakpoint, the hypervisor enters into escorting only when both EIP and ESP values match the previously saved EIP and ESP pair. The hypervisor can distinguish these two types of debug exceptions easily by checking whether it is in escorting mode. The algorithm details are shown in Figure 5.

PCB Exit. To exit from the hypervisor escorting, the PCB issues another hypercall. The hypervisor checks if InEscorting is set. If not, it returns an error message; otherwise, it clears InEscorting flag. The PCB should also issue a hypercall to protect its data if the data are left in plaintext. The hypervisor sets no more breakpoints and will handle future interrupts and exceptions in the normal way.

4.4 Region Access Control

A risky access to a memory region with a checkpoint causes a page fault and an access to an I/O port with a checkpoint throws out a general protection exception. Therefore, we modify the hypervisor's page fault routine `do_page_fault` and the general protection exception handler `do_general_protection`. In the former, the hypervisor gets the address of the trapped instruction from EIP register and the address being checked from CR2 register, while in the exception handler, the I/O port number is enclosed in the instruction.

Debug-handler Algorithm: Breakpoint address stored in *EIP*, the stack address stored in *ESP*

```
/* Exit from Escorting */
(1) If $InEscorting = 1$ and $EIP$ points to the entry of __switch_to, then
    (a) execute step (2,3,4) of the IRQ-handler algorithm.
    (c) Fetch task_struct->thread ->ip, which is the address of the next instruction
        for resuming the present flow. Denote it by $EIP_r$. Save $(EIP_r, ESP)$ tuple.
    (d) set a local breakpoint on $EIP_r$ and return 0.
/* Enter into Escorting */
(2) If there exists a saved $(EIP', ESP')$ pair, s.t. $ESP' = ESP$ and $EIP' = EIP$, then
    (a) remove the breakpoint at $EIP$;
    (b) Restore to the genuine stack by replacing the stack PTE with $MA_{ss}$.
    (c) set a local breakpoint at the entry of __switch_to function,
    (d) Set $InEscorting = 1$, and return 0.
(3) Return -1 as an error message.
```

Fig. 5. Exception handler for escorting

If the access is granted by the hypervisor, the event will not be forwarded to the guest kernel. In that case, The legitimate flow continues to execute the intercepted instruction without being re-scheduled as the guest kernel does not observe this exception. For unauthorized accesses, the page fault or exception is passed to the guest kernel. DriverGuard is compatible with memory mapping for page sharing because the checkpoints are deployed at the PTEs. A buffer mapped to two addresses has two PTE checkpoints. In the following, we elaborate the details of region access control according to all types of regions except the control region.

Application Buffer. The application data buffer is the starting or ending point of an I/O flow. We use the system call interception applied in [18] to get the buffer address. The technique used in [18] is to replace the first byte of the system call handler with instruction *HLT*, which causes a protection exception and passes the control to the hypervisor.

I/O Buffer. The addresses of I/O buffers are obtained within an escorted command-PCB. Since the I/O buffer contains the data to/from the device, they are not protected by encryption. The hypervisor blocks all accesses not from an escorted PCB. For an input buffer containing the data from the device, the driver always encrypts the data before moving them to other locations, whereas for an output buffer the driver must decrypt the data after copying them to the output buffer.

Driver Buffer. Driver buffers are for the driver to temporarily hold data for processing. When the data in those buffers are encrypted, the hypervisor does not set up checkpoints for them. Only when the escorted PCB is temporarily scheduled off from the CPU, the hypervisor sets up the checkpoints against all accesses as the data are in plaintext. In this case, the PCB notifies the hypervisor about the buffer address.

Key Buffer. The key buffer holds the secret key used by the driver. The hypervisor allows the key to be read only from the instructions from the encryption/decryption functions (i.e. key-PCBs) and is currently in escorting mode. Thus, other code can not access the secret key.

4.5 Device Control Protection

The hypervisor denies all write accesses to the region not from an escorted PCB, and maintains the consistency between the I/O buffer address specified in an I/O command and the buffer addresses requested by the device driver. This is because the kernel may manipulate the I/O command such that the device uses an unprotected I/O buffer for transferring. To defeat such attacks, the driver's command-PCB informs the hypervisor the locations of the I/O buffers in use, such as the DMA buffer and the DMA descriptor queue. The hypervisor inserts them in the region table and sets up the checkpoints accordingly. Therefore, it ensures that the I/O buffer in use is always protected.

5 Evaluation

We have implemented DriverGuard and run experiments on five char devices to evaluate the security and performance. We tested three input devices (a USB keyboard, a web camera and a fingerprint reader) and two output devices (a sound card and a printer). In principle, DriverGuard is applicable to network and disk I/O as well. Nonetheless, as argued earlier, this type of I/O can be protected using application level data encryption. Therefore, we do not experiment with them. To demonstrate the effectiveness of DriverGuard, we ran it against three kernel-level keyloggers [6, 15, 20]. None of the keyloggers is able to steal the keystroke information.

5.1 Usage of PCB

In our experiments, we manually identify all PCBs on the source code of device drivers and the drivers in the kernel's I/O subsystems, e.g., a host controller driver. It is straightforward to identify command-PCBs and key-PCBs, because key-PCBs are introduced by DriverGuard while command-PCBs are the code accessing port I/O, MMIO or structures used by devices (e.g., frame list of UHCI). Identifying computation-PCB requires the semantic knowledge of the code. We trace the I/O data to spot code segments computing on the I/O data. Note that code segments for copying or moving data are not PCBs.

Table 1 lists all the involved drivers used in our experiments and the number of PCBs in each of them. We found that a driver typically has only around ten PCBs and each PCB has approximately 15 lines of code without making function calls (except the encryption and decryption functions). The total PCB code only

Table 1. The number of PCBs and the average size for each driver used in our experiments. The drivers labeled with stars are those within the kernel's I/O subsystem. The PCB size includes the hypercalls and the calls to the encryption and decryption functions.

Driver	**Size** (LOC)	**# of PCBs**	**Avg. PCB Size** (LOC)	**Device**
keyboard driver	4964	11	17	keyboard
HID*	12771	13	10	keyboard
UVC driver	7838	7	11	camera
EHCI*	10011	6	15	camera
HDA-Intel	47825	8	6	sound card
Sound-core*	18722	5	4	sound card
devio	1628	7	12	printer, fingerprint reader
UHCI*	7600	5	14	printer, fingerprint reader

account for 1∽3% of the driver code. The tiny size of PCB and its simple logic allow for high security assurance, as compared to protecting the execution of thousands of lines of driver code.

5.2 Performance Evaluation

We experiment DriverGuard on a PC with Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz, 2GB main memory, running Xen 4.0.0 and a PV guest domain with Linux kernel 2.6.31.13. DriverGuard adds only 1727 SLOC to the Xen hypervisor. Our performance evaluation includes a cost measurement of DriverGuard's component functions and a set of application tests with five devices. We remark that the I/O characteristic is favorable to our scheme as data generation/rendering devices are usually much slower than the CPU. Therefore, DriverGuard does not affect the driver performance since the device speed is the performance bottleneck.

We choose 128-bit RC4 as the encryption cipher in our implementation, because RC4's compact code is easier to protect and does not significantly expand the PCB size. We instrument the DriverGuard code to measure the CPU cycles consumed by its main components including the escort hypercalls, the interrupt handler `do_IRQ`, the debug handler `do_debug`, the page fault handler `do_page_fault` and the general protection exception handler `do_general_protection`. Note that the encryption cost comprises the overhead of loading the secret key which incurs one page fault and the hypervisor's checkpoint removal. The results are shown in Table 2.

For each device we have experimented with, we measure the overhead and evaluate whether DriverGuard causes significant delay to the driver operation and the applications. Table 3 shows all the measured results.

Keyboard. In our experiment, we measure the time cost of the interrupt handler which moves the data from the keyboard to the tty buffer. Although the

Table 2. Cost of DriverGuard components

Components	`do_IRQ`	`do_debug`	`do_page_fault`	`do_general_protection`	*Encryption 1KB*
CPU cycles	844	739	961	1813	23355

Table 3. Performance overhead of protected keyboard, camera, fingerprint, printer and sound card I/O

	keyboard code transfer	**camera waiting**	**fingerprint collection**	**1 page print**	**sound card open**
without DriverGuard	0.053ms	33.24ms	2.61s	15.74s	7.8μs
with DriverGuard	0.138ms	33.38ms	2.63s	16.19s	12.3μs
percentage	160.40%	0.42%	0.77%	2.86%	57.7%

protected keyboard I/O is more than 2 times slower than the unprotected one, it does not affect the application because it is still negligible as compared to the speed of human keystrokes.

Camera. The web camera in our experiment is managed by the default Linux UVC driver. When the camera is opened by an application, it continuously collects video data and sends them to the application. The UVC driver's interrupt handler moves and decodes the data stream from the camera into a video frame, which resides in the driver's buffer mapped to the user space. The user application can directly use the frame data like normal user-space data.We measure the time overhead of the application's waiting time for getting new data, which is a key factor to the quality of the generated video stream. Although the interrupt handler in protection is much slower due to the encryption of four pages data, the drivers spends much more time in waiting for the camera's data generation. Thus, the cost of the interrupt handler does not cause the overall performance degradation. We have also tested video chatting using Empathy 2.30.2, which is a graphic instant messenger. We do not perceive delays in the experiments.

Fingerprint-Reader. Our fingerprint reader is the Upek Touchchip fingerprint sensor. In our evaluation experiment, we choose *Fingerprint GUI* [2] as the application which uses the default Linux driver *devio* to communicate with the fingerprint reader. When the fingerprint reader is active, the driver's interrupt handler continuously loads the collected fingerprint data into its buffers, which are then fetched by *Fingerprint GUI* by calling the *ioctl* function. In our experiments, we measure the whole I/O session of fingerprint collection.

Printer. The printer in our experiments is HP Officejet 7210 and the device driver in use is *devio*. The print-process opens the printer and issues *ioctl* to send data to the printer. After sending out the data, it waits for a signal sent

[2] `http://www.n-view.net/Appliance/fingerprint/index.php`

back by the printer to close the printer. In our experiments, we measure the turnaround time between the printer open and the printer close. Note that the relative overhead drops if more pages are printed out.

Sound Card. The sound card in our test is Intel Corporation 82801I (ICH9 Family) HD Audio and the driver in use is *HDA Intel*. We run the application *Totem* which plays MP3 files. Totem places its sound data into a user space buffer, which is mapped into the DMA buffer specified by the driver. When the music is playing, Totem directly sends data into mapped DMA region in user space, and issues *ioctl* to synchronize and update information. The hardware fetches the data from the DMA buffer directly without the driver's involvement. Hence, DriverGuard is only involved in protecting the control region so that the kernel can not change the location of the DMA buffer in use. There is no cost for DriverGuard during music playing, though the cost in opening the sound card is high.

6 Conclusion

We have proposed DriverGuard which is a hypervisor-based system protecting I/O flows between devices and applications, especially for devices generating data or rendering data. DriverGuard protects I/O device control, I/O data transfer and a driver's data processing, against attacks from an untrusted guest kernel. It is featured with fine granularity protection with strong security assurance and low overhead. It only introduces 1727 SLOC to the hypervisor and a few lines to the driver code. DriverGuard can work jointly with user-space data protection schemes to safeguard the entire data lifecycle.

The main drawback of our scheme is the need for manually discover PCBs from a driver, a process which requires the domain knowledge of the I/O data flow. In our future work, we will investigate techniques to automate PCB discovery. We will also consider extending our work to the multi-core platform.

Acknowledgement. We are grateful to Virgil Gligor and Adrian Perrig for their constructive suggestions. We also thank the anonymous reviewers for their helpful comments.

References

1. Bhargava, R., Serebrin, B., Spadini, F., Manne, S.: Accelerating two-dimensional page walks for virtualized systems. In: ASPLOS XIII: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 26–35. ACM, New York (2008)
2. Buchanan, E., Roemer, R., Shacham, H., Savage, S.: When good instructions go bad: Generalizing return-oriented programming to RISC. In: Syverson, P., Jha, S. (eds.) Proceedings of CCS 2008, pp. 27–38. ACM Press, New York (2008)
3. Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.R., Shacham, H., Winandy, M.: Return-oriented programming without returns. In: Keromytis, A., Shmatikov, V. (eds.) Proceedings of CCS 2010, pp. 559–572. ACM Press, New York (2010)

4. Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., Ports, D.R.K.: Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2008), Seattle, WA, USA (March 2008)
5. Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D.: An empirical study of operating systems errors. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, SOSP 2001, pp. 73–88. ACM, New York (2001), `http://doi.acm.org/10.1145/502034.502042`
6. Gadgetweb.de: How to: Building your own kernel space keylogger (2010), `http://www.gadgetweb.de/programming/39-how-to-building-your-own-kernel-space-keylogger.html`
7. Ganapathy, V., Renzelmann, M.J., Balakrishnan, A., Swift, M.M., Jha, S.: The design and implementation of microdrivers. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XIII, pp. 168–178. ACM, New York (2008), `http://doi.acm.org/10.1145/1346281.1346303`
8. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. In: Proceedings of the 9th ACM Symposium on Operating Systems Principles, pp. 193–206. ACM, New York (2003)
9. Trusted Computing Group: TPM main specification. Main Specification Version 1.2 rev. 85 (February 2005)
10. Langweg, H.: Building a trusted path for applications using cots components. In: In Proceedings of NATO RTO IST Panel Symposium on Adaptive Defence in Unclassified Networks (2004)
11. Lineberry, A.: Malicious code injection via /dev/mem. In: Black Hat (March 2009)
12. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: Trustvisor: Efficient tcb reduction and attestation. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 143–158. IEEE Computer Society, Washington, DC, USA (2010), `http://dx.doi.org/10.1109/SP.2010.17`
13. McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: An execution infrastructure for TCB minimization. In: Proceedings of the ACM European Conference in Computer Systems (EuroSys) (April 2008)
14. McCune, J.M., Perrig, A., Reiter, M.K.: Safe passage for passwords and other sensitive data. In: Proceedings of the Symposium on Network and Distributed Systems Security (NDSS) (February 2009)
15. Mercenary: Kernel based keylogger (2002), `http://goo.gl/7qwmr`
16. Neugschwandtner, M., Platzer, C., Comparetti, P.M., Bayer, U.: *d*anuis - dynamic device driver analysis based on virtual machine introspection. In: Proceedings of the 7th Detection of Intrusions and Malware & Vulnerability Assessment (2010)
17. Nomoto, T., Oyama, Y., Eiraku, H., Shingawa, T., Kato, K.: Using a hypervisor to migrate running operating systems to secure virtual machines. In: Proceedings of the 34th Annual IEEE Computer Software and Application Conference (2010)
18. Onoue, K., Oyama, Y., Yonezawa, A.: Control of system calls from outside of virtual machines. In: Proceedings of Symposium of Applied Computing (2008)
19. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Proceedings of the 2008 IEEE Symposium on Security and Privacy, pp. 233–247. IEEE Computer Society, Washington, DC, USA (2008), `http://portal.acm.org/citation.cfm?id=1397759.1398072`

20. Phrack: Writing linux kernel keylogger (2002), http://www.phrack.org/issues.html?issue=59
21. Seshadri, A., Luk, M., Qu, N., Perrig, A.: Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, SOSP 2007, pp. 335–350. ACM, New York (2007), http://doi.acm.org/10.1145/1294261.1294294
22. Shacham, H.: The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: De Capitani di Vimercati, S., Syverson, P. (eds.) Proceedings of CCS 2007, pp. 552–561. ACM Press, New York (2007)
23. Shi, E., Perrig, A., Doorn, L.V.: Bind: A fine-grained attestation service for secure distributed systems. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 154–168 (2005)
24. Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y., Kato, K.: Bitvisor: a thin hypervisor for enforcing i/o device security. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2009, pp. 121–130. ACM, New York (2009), http://doi.acm.org/10.1145/1508293.1508311
25. Swift, M.M., Bershad, B.N., Levy, H.M.: Improving the reliability of commodity operating systems. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 207–222. ACM, New York (2003), http://doi.acm.org/10.1145/945445.945466
26. Wang, X., Li, Z., Li, N., Choi, J.Y.: PRECIP: Towards practical and retrofittable confidential information protection. In: Proceedings of NDSS (2008)
27. Wang, Z., Jiang, X.: Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Proceedings of IEEE Symposium on Security and Privacy (2010)
28. Wang, Z., Jiang, X., Cui, W., Ning, P.: Countering kernel rootkits with lightweight hook protection. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 545–554 (2009)
29. Willmann, P., Rixner, S., Cox, A.L.: Protection strategies for direct access to virtualized i/o devices. In: Proceedings of USENIX Annual Technical Conference (2008)
30. Willmann, P., Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A.L., Zwaenepoel, W.: Concurrent direct network access for virtual machine monitors. In: Proceedings of the 13th International Symposium on High Performance Computer Architecture (2007)
31. Ye, Z.E., Smith, S., Anthony, D.: Trusted paths for browsers. ACM Trans. Inf. Syst. Secur. 8(2), 153–186 (2005)
32. Zhou, F., Condit, J., Anderson, Z., Bagrak, I., Ennals, R., Harren, M., Necula, G., Brewer, E.: Safedrive: safe and recoverable extensions using language-based techniques. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI 2006, pp. 45–60. USENIX Association, Berkeley (2006), http://portal.acm.org/citation.cfm?id=1298455.1298461

Time-Storage Trade-Offs for Cryptographically-Enforced Access Control

Jason Crampton

Royal Holloway, University of London
jason.crampton@rhul.ac.uk

Abstract. Certain classes of authorization policies can be represented as a directed graph and enforced using cryptographic techniques. Such techniques typically rely on the authorized user deriving a suitable decryption key using a secret value and public information. Hence, it is important to find enforcement schemes for which little public information is required and key derivation is efficient. These parameters are related to the number of edges and the distance between nodes in the graph associated with the authorization policy. In this paper we consider ways in which two particular types of authorization graph can be rewritten so that the number of edges and the greatest distance between any two nodes are reduced, thereby providing the basis for more efficient cryptographic enforcement.

1 Introduction

In a large multi-user computer system, it is usually the case that users will require access to resources managed by that system, such as files and printers. However, many resources may be sensitive – computerized personnel files, for example – so access to those resources should be restricted. One of the fundamental security services provided by a computer system is *access control*, a mechanism by which the interaction between users and protected resources is constrained. Generally, access control is "policy-based": an authorization policy is defined, which specifies those interactions that are authorized; every attempt by a user to interact with a protected resource is intercepted by the access control mechanism; and the interaction is checked to see whether it is authorized by the policy or not.

The increasing trend towards out-sourcing the storage of data has seen a resurgence of interest in cryptographic access control, in which data is stored encrypted (perhaps by an "honest-but-curious" third party) and may be released in that encrypted form to any user [13]. However, only authorized users possess or can derive the decryption key.

An information flow policy [15] is a particular type of authorization policy that is particularly well suited to cryptographic enforcement. There are a large number of cryptographic enforcement schemes for information flow policies in the literature (see [11] for a survey of such schemes). The security of such schemes is increasingly well understood [2,6] and attention has turned to their efficiency in recent years [4,5,10,14].

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 245–261, 2011.

The public storage requirements of an enforcement scheme are determined by the number of edges in the graph associated with the authorization policy, while the worst-case time complexity of key derivation is determined by the longest directed path in the graph. Hence, the goal of recent work has been to rewrite the graph by adding a relatively small number edges (and possibly nodes) to produce a graph that contains the same, but significantly shorter, paths as those in the original graph.

In this paper, we consider the efficiency of enforcement schemes for two related types of information flow policies. The graph of the first type, which is used to define policies for scalable multimedia formats, resembles a square grid and has not previously been studied from the perspective of efficiency. This grid has $2n(n-1)$ edges (where n^2 is the number of nodes in the grid) and the longest path is $2(n-1)$. One of our results shows that we can define an equivalent authorization graph in which the number of nodes is less than $4n(n-1)$ and the longest path is $4(\sqrt{n}-1)$. In fact, this result is a corollary of a much more powerful "master" theorem, which provides a tool for constructing many different graphs offering different trade-offs. A study of this first type of authorization policy yields fresh insights into the efficient cryptographic enforcement of temporal authorization policies, which has been the subject of considerable research in recent years [5,10,14].

There exists a large body of work on reducing the diameter of a directed graph [8,16,18,19]. Previous work on deriving new access control graphs that allow for efficient cryptographic enforcement has tended to use these "off-the-shelf" techniques. One significant advantage of our work in this paper is that we provide explicit constructions that are specific to the graphs under consideration. This has two advantages: it is clear how to construct the new graph, something that has not always been the case in related work; and we are able to provide explicit, rather than asymptotic, bounds for the number of edges and longest path, something that suggests our results will be more useful in practice.

In the next section, we briefly describe relevant background material. In Sec. 3 we introduce a number of simple constructions that provide an insight into the more complex constructions that we will describe in Sec. 4. The main contribution of Sec. 4 is to prove Theorem 1, which yields many interesting corollaries that give rise to concrete cryptographic enforcement schemes. We compare our results with related work in Sec. 5 before summarizing our contributions and identifying interesting areas for future work.

2 Graph-Based Authorization Policies

Let $G = (L, E)$, where L is a set of security labels and $E \subseteq L \times L$ is a set of directed edges. Let $\lambda : U \times O \to L$, where $\lambda(u)$ denotes the security label of a user $u \in U$ and $\lambda(o)$ denotes the security label of an object $o \in O$. Then (G, λ) defines a *graph-based authorization policy*, where $u \in U$ is authorized for $o \in O$ if and only if there exists a directed path from $\lambda(u)$ to $\lambda(o)$ in G. An information flow policy is a special case of a graph-based authorization policy, in which the graph is defined by a security lattice [15].

A graph-based authorization policy is particularly well suited to a cryptographic enforcement model. Each node $x \in L$ is associated with a secret value $\kappa(x)$, and, for each edge $(x, y) \in E$, we publish $\mathsf{Enc}_{\kappa(x)}(\kappa(y))$, where $\mathsf{Enc}_{\kappa}(M)$ denotes the encryption of message M using key κ. Then any user in possession of $\kappa(x)$ can derive $\kappa(y)$ in one step if $(x, y) \in E$, and for any z on a path from x containing d edges, $\kappa(z)$ can be (iteratively) derived in d steps.[1]

In this paper, we will consider two particular types of information flow policies, one that is used to implement access control for scalable multimedia formats and one that is used to implement temporal access control. It turns out that these types of policies have very similar sets of security labels, and results for the first type can be easily applied to policies of the second type. We discuss these policies in more detail in Sec. 2.3 and Sec. 2.4, respectively.

2.1 Derivation-Storage Trade-Offs

We could, trivially, give a user the key associated with each label for which she is authorized, but this type of approach is rarely used. Most of the literature on the cryptographic enforcement of graph-based authorization policies assumes that each user has a single secret value and the keys for which she is authorized are derived from this secret value. For schemes of this nature, the more public information required by the scheme, the smaller the number of key derivation steps required in the worst case.

Consider a partially ordered set $(L, \leqslant)$, which can be represented by a directed, acyclic graph (L, E). There are two obvious choices for the edge set E: one is the full partial order relation $\leqslant$; the second is to omit all transitive and reflexive edges from $\leqslant$ to obtain the *covering relation*, denoted $\lessdot$. The graph $(L, \lessdot)$ is called the *Hasse diagram* of L, and is the standard representation of L as a directed graph [12].

It can be seen that a cryptographic enforcement scheme for a directed graph can be used specifically to enforce an information flow policy. We may use the graph $(L, \leqslant)$, in which case key derivation can always be performed in one step. In contrast, key derivation may require a number of steps when we use the graph $(L, \lessdot)$. The trade-off here is that the second graph contains fewer edges and hence the number of items of public information that are required to support key derivation is smaller. The study of these kinds of trade-offs will be the focus of this paper.

2.2 Correctness and Security

A key assignment scheme that enforces an information flow policy for $(L, \leqslant)$ must be correct and it must be secure. Informally, we say a key assignment scheme is

[1] There exist cryptographic enforcement schemes that are designed specifically for posets, rather than arbitrary directed graphs. The Akl-Taylor scheme [1] – one of the earliest examples of cryptographic access control, which was designed to enforce the Bell-LaPadula simple security property – is one such example. For a more comprehensive survey of such schemes see [11].

- correct if each user can derive the keys for which she is authorized;
- secure if no set of users can derive a key for which none of them is authorized.

Recently, the notions of *key recovery* and *key indistinguishability* have been introduced to capture in more formal terms what it means for a key assignment scheme to be secure [2,6]. The scheme described above, in which the public information is $\left\{\mathsf{Enc}_{\kappa(x)}(\kappa(y)) : (x,y) \in E\right\}$, provides security against key recovery for any reasonable choice of encryption function. Such a scheme can be extended to one with the property of key indistinguishability by associating a secret value $\sigma(x)$ with each node x, making $\kappa(x)$ a function of $\sigma(x)$ and using $\sigma(x)$ to derive $\sigma(y)$.

For the purposes of this paper, it is sufficient to note that given a directed, acyclic graph $G = (L, E)$, there exists a key assignment scheme that has the property of key indistinguishability, the amount of storage required is proportional to $|E|$ (the cardinality of E), and the number of derivation steps required to derive $\kappa(y)$ from $\kappa(x)$ is the length of the shortest path between x and y in G. The interested reader is referred to the literature for further details [2,6].

2.3 Access Control for Scalable Multimedia Formats

Scalable multimedia formats, such as MPEG-4 [17] and JPEG2000 [9], consist of two components: a non-scalable *base* component and a scalable *enhancement* component. Decoding the base component will yield low quality results. The quality of the decoded data can be improved by decoding the enhancement component as well as the base component. The enhancement component may comprise multiple "orthogonal" layers, orthogonal in the sense that each layer controls a distinct aspect of the quality of the encoded content. The MPEG-4 FGS (fine granularity scalability) format [17], for example, has a bit-rate layer and a peak signal-to-noise ratio (PSNR) layer.

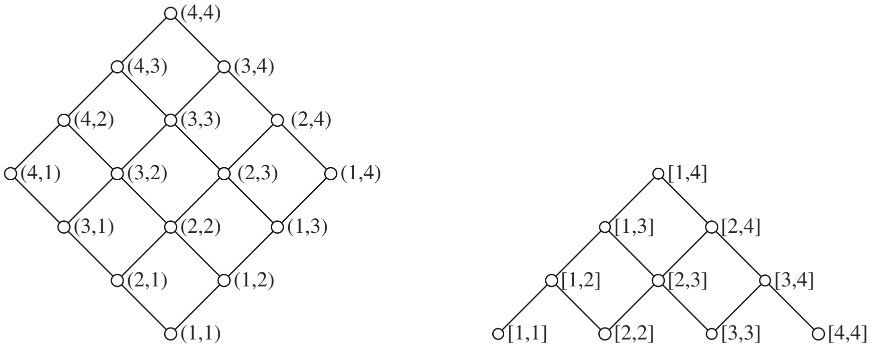
For scalable multimedia formats with two orthogonal layers, such as MPEG-4 FGS and JPEG 2000, each user and multimedia objected is associated with some pair (x, y) (see [20] for further details). A user associated with (x, y) should be able to decrypt a multimedia stream associated with (x', y'), whenever $x' \leqslant x$ and $y' \leqslant y$. We will assume that each layer has the same number of levels, although it is straightforward to generalize our results. Without loss of generality, we will associate each level with an integer between 1 and n. Hence, the set of security labels is defined by the set

$$D_n \overset{\text{def}}{=} \{(x, y) : 1 \leqslant x, y \leqslant n\}$$

with $(x, y) \leqslant (x', y')$ if and only if $x \leqslant x'$ and $y \leqslant y'$. The Hasse diagram of $(D_n, \leqslant)$ has the form of a diamond, as illustrated in Figure 1(a) for the case $n = 4$.

2.4 Temporal Authorization Policies

As its name suggests, temporal access control restricts access on the basis of time. The use of cryptographic enforcement for temporal authorization policies



(a) $(D_4, \leqslant)$

(b) $(T_4, \subseteq)$

Fig. 1. The Hasse diagrams of $(D_4, \leqslant)$ and $(T_4, \subseteq)$

was suggested by Bertino et al. [7]. We assume that time is divided into n intervals, each of which is represented by a clock "tick" identified with an integer between 1 and n. Users and objects are associated with an interval comprising $t \leqslant n$ consecutive natural numbers. We write $[x, y]$ to denote the interval $x, x+1, \dots, y-1, y$. The interval $[x, x]$ corresponds to a unit interval representing clock tick x. A user authorized for interval $[x, y]$ must be able to derive keys for all intervals of the form $[x', y']$, $x \leqslant x' \leqslant y' \leqslant y$.

Henceforth, we write T_n to denote the set of intervals in $[1, n]$: that is,

$$T_n \stackrel{\text{def}}{=} \{[x, y] : 1 \leqslant x \leqslant y \leqslant n\}.$$

We denote the set of all intervals by T_n because the Hasse diagram of the partially ordered set $(T_n, \subseteq)$ is a triangular grid, as illustrated in Fig. 1(b). We may refer to T_n as an n-triangle. A node of the form $[x, x] \in T_n$ will be called a *leaf node*. The set of leaf nodes corresponds to the totally ordered set of time points $1 < 2 < \dots < n$.

3 Basic Constructions for D_m

We first define the notion of an enforcing set of edges. Given a directed, acyclic graph $G = (L, E)$, we write E^* to denote the transitive closure of E: that is, $(x, y) \in E^*$ if and only if there exists a path from x to y in G.

Definition 1. *Given two authorization graphs $G_1 = (L, E_1)$ and $G_2 = (L, E_2)$, we say that G_1 and G_2 are* equivalent *if $E_1^* = E_2^*$. Given a partially ordered set $(L, \leqslant)$, we say $E \subseteq L \times L$ is an* enforcing set of edges for $(L, \leqslant)$ *if (L, E) is equivalent to $(L, \leqslant)$.*

In particular, (L, E) enforces the same information flow policy as $(L, \leqslant)$.

Definition 2. *Given an enforcing set of edges E for a set of nodes L, we define the* distance *between x and y to be the length of the shortest path in (L, E) between x and y. We define the* diameter *of E to be the maximum distance between any two nodes in (L, E).*

In general, we wish to find an enforcing set of edges for D_m, which we will denote by $E_\Diamond(m)$, such that the cardinality of $E_\Diamond(m)$ is small and the diameter of $E_\Diamond(m)$ is small. For a fixed method of generating a set of edges for D_m, we write $e_\Diamond(m)$ and $d_\Diamond(m)$ to denote the cardinality of $E_\Diamond(m)$ and the diameter of $(D_m, E_\Diamond(m))$, respectively. For T_m, we define $E_\vartriangle(m)$, $e_\vartriangle(m)$ and $d_\vartriangle(m)$ in an analogous fashion. When it is clear from context that we are referring to D_m, we will omit $\Diamond$; similarly we will omit $\vartriangle$ when no confusion can occur.

We first note that there exists a lower bound on $e_\vartriangle(m)$. In particular, it is easy to establish the following result.

Proposition 1. *The cardinality of any enforcing set for T_m must be at least $m(m-1)$.*

A very similar result was proved by Crampton [10]. Briefly, the proof is by contradiction: we assume that there exists an enforcing set for T_m of cardinality less than $m(m-1)$. From this assumption it follows that at least one node in T_m has out-degree less than 2, from which we can prove that this cannot be an enforcing set of edges (either because one node can reach too many leaf nodes or because one node cannot reach sufficient leaf nodes).

It follows, by symmetry, that an enforcing set of edges for D_m must have cardinality at least $2m(m-1)$. Clearly, there exists an enforcing set of edges for D_m of cardinality $2m(m-1)$ and diameter $2(m-1)$ corresponding to the graph $(D_m, \lessdot)$, illustrated in Figure 1(a) for $m = 4$.

3.1 Binary Decomposition

Proposition 2. *There exists an enforcing set of edges $E_\Diamond(m)$ for $(D_m, \leqslant)$ such that $e_\Diamond(m) = m^2 \log m$ and $d_\Diamond(m) = 2 \log m$.*[2]

Proof. We first consider D_{2m}, splitting it into four copies of D_m, as shown in Fig. 2. We label the four copies of D_m in the following way:

$$D_{0,0} = \{[x, y] : 1 \leqslant x, y \leqslant m\}, \qquad D_{1,0} = \{[x+m, y] : 1 \leqslant x, y \leqslant m\},$$
$$D_{0,1} = \{[x, y+m] : 1 \leqslant x, y \leqslant m\}, \quad D_{1,1} = \{[x+m, y+m] : 1 \leqslant x, y \leqslant m\}.$$

Evidently, for each pair (x, y), $1 \leqslant x, y \leqslant m$, there exist four related nodes in D_{2m}: $[x, y] \in D_{0,0}$, $[x+m, y] \in D_{1,0}$, $[x, y+m] \in D_{0,1}$ and $[x+m, y+m] \in D_{1,1}$. Then for each pair (x, y), where $1 \leqslant x, y \leqslant m$, we add the following four edges:

- for every node $[x+m, y+m] \in D_{1,1}$ we add two edges, one to $[x+m, m] \in D_{1,0}$ and one to $[m, y+m] \in D_{0,1}$;

[2] All logarithms are base 2, unless explicitly stated otherwise.

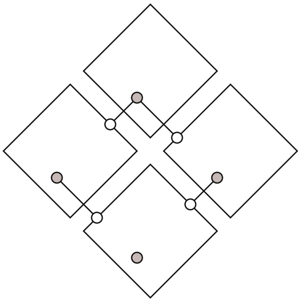


Fig. 2. Connecting nodes in copies of D_m contained in D_{2m}

- for every node $[x+m, y] \in D_{1,0}$ we add an edge to $[m, y] \in D_{0,0}$;
- for every node $[x, y+m] \in D_{0,1}$ we add an edge to $[x, m] \in D_{0,0}$.

Figure 2 illustrates these edges for one particular pair (x, y); the four related nodes are shaded. In total, therefore, we add $4m^2$ edges, since there are m^2 choices for the pair (x, y). We now recursively apply this construction to each copy of D_m. Hence, $e_{\Diamond}(2m)$ satisfies the recurrence relation

$$e_{\Diamond}(2m) = 4m^2 + 4e_{\Diamond}(m) \quad \text{and} \quad e_{\Diamond}(1) = 0,$$

from which it follows that $e_{\Diamond}(m) = m^2 \log m$.

Moreover, the diameter of D_{2m} is two greater than the diameter of D_m (since we can get from any node in $D_{1,1}$ to a node in any other copy of D_m in at most two hops). Hence, the diameter $d_{\Diamond}(2m)$ satisfies the recurrence relation

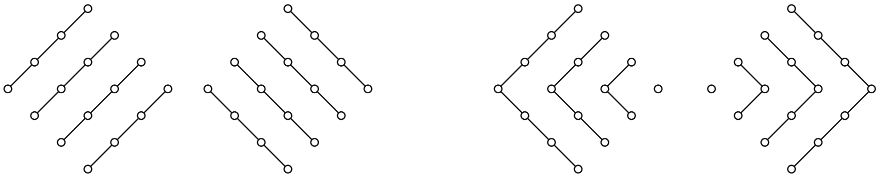
$$d_{\Diamond}(2m) = 2 + d_{\Diamond}(m) \quad \text{with} \quad d_{\Diamond}(1) = 0,$$

from which we deduce that $d_{\Diamond}(m) = 2 \log m$. □

3.2 Linear Decomposition

Atallah et al. noted that T_m can be regarded as two orthogonal sets of total orders (each set comprising total orders of cardinality $1, 2, \ldots, m$). This means that techniques to reduce the diameter of a total order can be applied to T_m, as shown to good effect by Atallah et al. In this section, we discuss how these techniques can be applied to D_m, in preparation for the schemes in Sec. 4.

We write C_m to denote the total order (or *chain*) comprising m elements. The graph $(C_m, \lessdot)$ contains $m-1$ edges and has diameter $m-1$. We will write $E_{\between}(m)$ to denote an enforcing set of edges for C_m, $e_{\between}(m)$ to denote the cardinality of $E_{\between}(m)$ and $d_{\between}(m)$ to denote the diameter of $(C_m, E_{\between}(m))$.



(a) (b)

Fig. 3. Chain partitions of D_m

Proposition 3. *Given an enforcing set of edges $E_{\between}(m)$, there exists an enforcing set of edges $E_{\Diamond}(m)$ such that*

$$e_{\Diamond}(m) = 2me_{\between}(m) \quad \textit{and} \quad d_{\Diamond}(m) = 2d_{\between}(m).$$

Proof. D_m can be viewed as two orthogonal sets of chains, each of length m, illustrated in Fig. 3(a). We may label the chain $\{(x,y) : 1 \leqslant y \leqslant m\}$ as C_x^1 and the chain $\{(y,x) : 1 \leqslant y \leqslant m\}$ as C_x^2. Then $E_{\Diamond}(m)$ is simply $2m$ copies of $E_{\between}(m)$.

Clearly this is an enforcing set of edges for D_m: if $(x,y) \leqslant (x',y')$ we simply traverse the edge set for $C_{x'}^1$ from (x',y') to (x',y) and then traverse the edge set for C_y^2 from (x',y) to (x,y). Moreover, the distance between (x',y') and (x,y) is no greater than $2d_{\between}(m)$. □

The following result was proved by Bodlaender et al. [8] and, independently, by Atallah et al. [3].

Proposition 4. *There exists an enforcing set of edges for C_m such that $e_{\between}(m) \leqslant m \log m$ and $d_{\between}(m) = 2$.*

The result is proved using an explicit, recursive construction. C_m is split into two chains of length $\left\lfloor \frac{m-1}{2} \right\rfloor$ and a single median node. An edge is added between every node in the upper chain to the median node and an edge is added from the median node to every node in the lower chain. Hence, it is possible to get from every node in the upper half to every node in the lower half in exactly two hops. The construction is then applied recursively to the two shorter chains. It is clear that $e_{\between}(m) \leqslant m + 2e_{\between}(m/2)$ and $d_{\between}(m) = 2$.

The two propositions above immediately yield the following result.

Corollary 1. *There exists a set of enforcing set of edges for D_m such that*

$$e_{\Diamond}(m) \leqslant 2m^2 \log m \quad \textit{and} \quad d_{\Diamond}(m) = 4.$$

One disadvantage with the decomposition described in Proposition 3 is that the edge set is not symmetric about the horizontal axis of symmetry, because each chain crosses this axis at different relative positions. Since T_m can be viewed as the upper half of D_m, it would be useful if we had an enforcing set of edges for D_m that was symmetric about this axis.

Hence, we consider a second decomposition of D_m into chains, illustrated in Fig. 3(b), which yields an enforcing set of edges when Bodlaender's scheme is applied to each chain. In this case, the median node of each chain is on the axis of symmetry and the resulting scheme will be symmetric about this axis, as desired. We then have the following result.

Proposition 5. *For all $m \geqslant 2$, there exists an enforcing set of edges for D_m such that*

$$e_{\Diamond}(m) = 4\left(\sum_{i=1}^{m-1} i + e_{\between}(i)\right) \quad \textit{and} \quad d_{\Diamond}(m) \leqslant 4.$$

Proof. There are two sets of chains in the partition illustrated in Figure 3(b). Each set contains chains of length $2m-1, 2m-3, \ldots, 1$. For each chain, we connect each node in the chain to the median node (which lies on the long diagonal); this requires $2m-2, 2m-4, \ldots, 2$ edges. We now apply Bodlaender's scheme to each of the chains above and below the long diagonal, which are of length $m-1, m-2, \ldots, 1$. Therefore, the total number of edges required is

$$2\left(\sum_{i=1}^{m-1} 2i + 2\sum_{i=1}^{m-1} e_{\between}(i)\right)$$

The result now follows.

Clearly, the diameter of the resulting set of edges is no greater than 4, since we require at most two hops in one set of chains and at most two hops in the other set of chains. □

Corollary 2. *For all $m \geqslant 2$, there exists an enforcing set of edges for T_m such that*

$$e_{\Diamond}(m) = 2\left(\sum_{i=1}^{m-1} i + e_{\between}(i)\right) \quad \textit{and} \quad d_{\Diamond}(m) \leqslant 4.$$

Note that, for this construction we may conclude that the number of edges required for T_m is half the number required for D_m because the construction is symmetric about the long diagonal of D_m. However, we cannot conclude that the number of hops required for T_m is half that required for D_m.

4 Additive Decomposition

In this section we explore an alternative to linear decomposition. In particular, we introduce the notion of *additive decomposition* and prove the following result.

Theorem 1. *Given an enforcing set of edges for D_a and D_b, we can construct an enforcing set of edges for D_{ab} such that*

$$e_{\Diamond}(ab) = a^2 e_{\Diamond}(b) + b^2 e_{\Diamond}(a) \quad \textit{and} \quad d_{\Diamond}(ab) = d_{\Diamond}(a) + d_{\Diamond}(b).$$

This "master" theorem immediately yields the following corollaries. Other, similar results could be derived: we choose these results because they will be used in the comparison with related work in Sec. 5.

Corollary 3. *There exists a set of enforcing set of edges for* D_m *such that* $e_\diamondsuit(m) = \frac{5}{4}m^2 \log m$ *and* $d_\diamondsuit(m) = \log m$*.*

Proof. It is easy to see that there exists an enforcing set of edges for D_2 of cardinality 5 and diameter 1. Then, for $m = 2^k$, we set $a = 2$ and $b = 2^{k-1} = m/2$ to obtain

$$e_\diamondsuit(m) = 4e_\diamondsuit\left(\frac{m}{2}\right) + \frac{5}{4}m^2 \quad \text{and} \quad d_\diamondsuit(m) = 1 + d_\diamondsuit\left(\frac{m}{2}\right),$$

from which the results follow. □

Corollary 4. *For* $m \geqslant 2$*, there exists an enforcing set of edges for* D_m *such that* $e_\diamondsuit(m) < 4m(m-1)$ *and* $d_\diamondsuit(m) = 4(\sqrt{m} - 1)$*.*

Proof. In this case, setting $a = b = \sqrt{m}$ and applying Theorem 1, we obtain

$$e_\diamondsuit(m) = me_\diamondsuit(\sqrt{m}) + me_\diamondsuit(\sqrt{m}) = 2me_\diamondsuit(\sqrt{m}) \quad \text{and} \quad d_\diamondsuit(m) = 2d_\diamondsuit(\sqrt{m}).$$

Trivially, there exists an enforcing set of edges for $D_{\sqrt{m}}$ of cardinality $2\sqrt{m}(\sqrt{m} - 1)$ and diameter $2(\sqrt{m} - 1)$. Hence, we have

$$e_\diamondsuit(m) = 4m\sqrt{m}(\sqrt{m} - 1) < 4m(\sqrt{m} - 1)(\sqrt{m} + 1) = 4m(m - 1).$$

Obviously, $d_\diamondsuit(m) = 4(\sqrt{m} - 1)$. □

Corollary 5. *For* $m \geqslant 4$*, there exists an enforcing set of edges for* D_m *such that* $e_\diamondsuit(m) \leqslant 2m^2(\log m - 1)$ *and* $d_\diamondsuit(m) = 4 \log \log m$*.*

Proof. Note, by Corollary 1, that there exists an enforcing set of edges for $D_{\sqrt{m}}$ of cardinality less than or equal to $m \log m$ (since $\log \sqrt{m} = \frac{1}{2} \log m$). Then setting $a = b = \sqrt{m}$, as in the proof of the previous corollary, we obtain the recurrence relations

$$e_\diamondsuit(m) \leqslant m^2 \log m + me_\diamondsuit(\sqrt{m}) \quad \text{and} \quad d_\diamondsuit(m) = 4 + d_\diamondsuit(\sqrt{m}).$$

Moreover, using the chain partition of Fig. 3(a), we can construct an edge set for D_4 that contains 32 edges and has diameter 4 (as illustrated in Fig. 4). The desired results immediately follow, the first by induction on m, the second directly from the recurrence relation. □

Figure 4 illustrates enforcing edge sets for D_4 and D_{16} constructed using Corollary 5. The figure also includes the 2-hop edge set for C_4 on which the whole construction is based. The enforcing set of edges for D_4 comprises 8 copies of the edge set for C_4. For clarity, the graph for D_{16} only includes the 32 edges connecting the maximal nodes in each D_4 supernode.

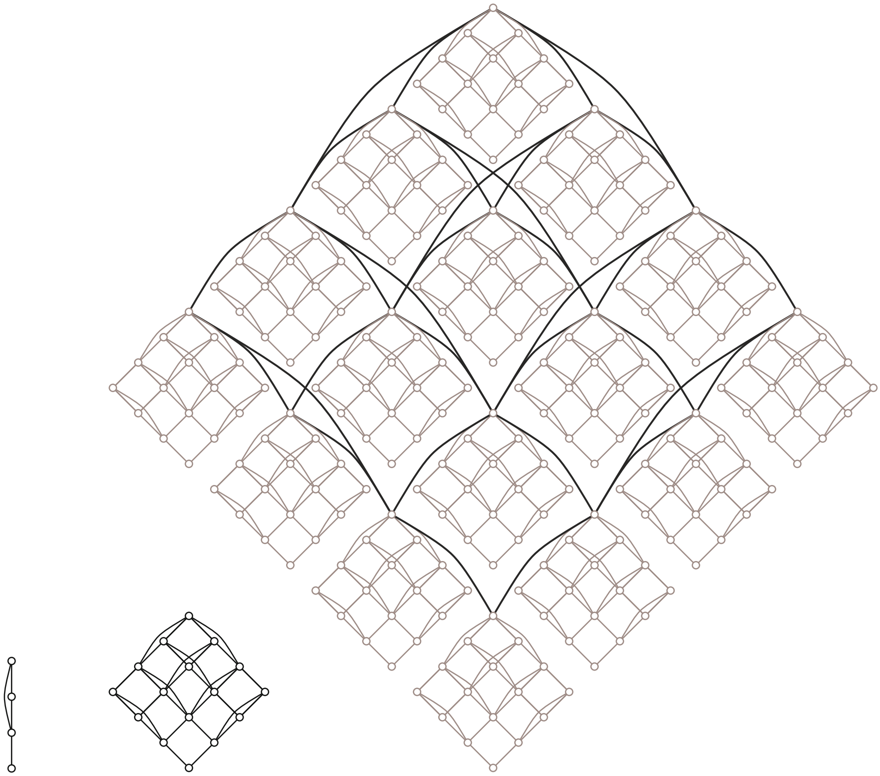
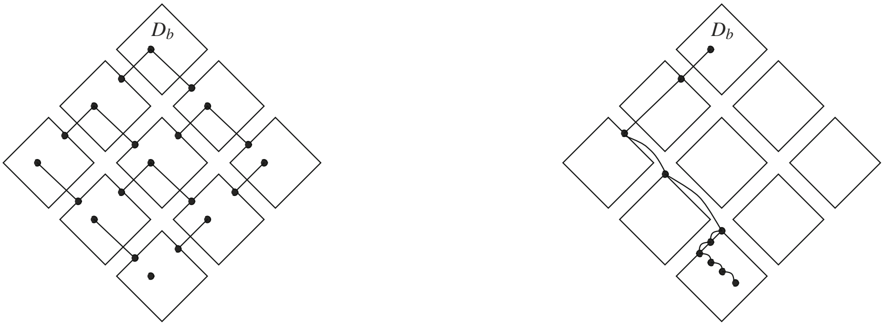


Fig. 4. Enforcing edge sets for D_4 and D_{16} with $\mathcal{O}(\log \log m)$ key derivation

We now explain the intuition behind Theorem 1. For $m = ab$, we first partition D_m into blocks of the form D_b. We call these blocks "supernodes". The set of supernodes forms a copy of D_a. Given an enforcing set of edges for D_a we can use this set of edges to connect nodes in different supernodes together. In particular, for every node in a copy of D_b we connect it to a single node in those copies of D_b to which it would be connected using the enforcing set of edges for D_a. The construction is illustrated in Fig. 5(a) for the case $m = 3b$ and a collection of nodes with the same relative position within each copy of D_b. In this illustration we are using the simplest enforcing set of edges for D_3; specifically, the one that resembles a diamond-shaped grid.[3] However, we could use any suitable set of enforcing edges to bridge the gaps between the supernodes. In Fig. 4, for example, we use a scheme for D_4 that uses 8 copies of a scheme for C_4. Finally, we construct an edge set for each of the supernodes using the scheme for D_b. We now formally prove Theorem 1.

[3] The binary decomposition described in Sec. 3 corresponds to the special case $a = 2$.



(a) Edges connecting supernodes

(b) A key derivation path

Fig. 5. D_m as copies of D_b, where $m = 3b$

Proof (of Theorem 1). Partition the set D_{ab} into copies of D_b and label each copy $D_b^{(\alpha,\beta)}$, where $1 \leqslant \alpha, \beta \leqslant a$. That is, the labels are taken from the partially ordered set $(D_a, \leqslant)$. By assumption, we have an enforcing set of edges $E_\Diamond(a)$ for $(D_a, \leqslant)$. For a node $(x, y) \in D_b^{(\alpha,\beta)}$ we add an edge to a node $(x', y') \in D_b^{(\alpha',\beta')}$ if and only if

1. $(x', y') \subseteq (x, y)$,
2. $((\alpha, \beta), (\alpha', \beta')) \in E_\Diamond(a)$, and
3. for all (x'', y'') such that $(x', y') \subset (x'', y'') \subseteq (x, y)$, $(x'', y'') \notin D_b^{(\alpha',\beta')}$.

The first condition requires that (x, y) be authorized for (x', y'); the second that there is an edge in the enforcing set for D_a; and the final condition requires that $D_b^{(\alpha',\beta')}$ contains no node larger than (x', y') for which (x, y) is authorized. A consequence of the above conditions is that $(x, y) \in D_b^{(\alpha,\beta)}$ is connected to at most one $(x', y') \in D_b^{(\alpha',\beta')}$ for all $(\alpha', \beta') \leqslant (\alpha, \beta)$.

Each copy of D_b contains b^2 nodes. Hence, we require b^2 copies of $E_\Diamond(a)$ to connect the copies of D_b. In addition, we require a^2 copies of the enforcing set of edges for D_b. In summary, $e_\Diamond(m) = a^2 e_\Diamond(b) + b^2 e_\Diamond(a)$, as required.

Note that the distance between any two nodes is determined by the distance between any two supernodes (as determined by the enforcing set of edges for D_a) and the distance between two nodes within a supernode (as determined by the enforcing set of edges for D_b). That is $d_\Diamond(m) = d_\Diamond(a) + d_\Diamond(b)$. A typical key derivation path is illustrated in Fig. 5(b). □

We can also generalize Theorem 1, as stated in the following result.

Theorem 2. *Let $m = \prod_{i=1}^{t} a_i$. Then there exists an enforcing set of edges such that*

$$e_\Diamond(m) = m^2 \sum_{i=1}^{t} \frac{1}{a_i^2} e_\Diamond(a_i) \quad \text{and} \quad d_\Diamond(m) = \sum_{i=1}^{t} d_\Diamond(a_i).$$

Proof. We prove the result by induction on t. Clearly the result holds for $t = 1$. Now suppose that the result holds for all $t < T$ and consider $m = \prod_{i=1}^{T} a_i$. Writing $a = a_1$ and $b = m/a_1$, we have, by Theorem 1,

$$\begin{aligned} e_\Diamond(m) &= \left(\frac{m}{a_1}\right)^2 e_\Diamond(a_1) + a_1^2 e_\Diamond\left(\frac{m}{a_1}\right) \\ &= \left(\frac{m}{a_1}\right)^2 e_\Diamond(a_1) + a_1^2 \frac{m^2}{a_1^2} \sum_{i=2}^{T} \frac{1}{a_i^2} e_\Diamond(a_i) \qquad \text{by inductive hypothesis} \\ &= m^2 \sum_{i=1}^{T} \frac{1}{a_i^2} e_\Diamond(a_i) \end{aligned}$$

as required. The result for $d_\Diamond(m)$ can also be proved by induction. □

We conclude this section with two results: the first is analogous to Theorem 1, the second relates enforcing sets of edges for D_m and T_m. The relationships between $E_\Diamond(m)$ and $E_\vartriangle(m)$ are not quite as straightforward as one might assume, for the following reasons:

- constructing $E_\vartriangle(m)$ by halving $E_\Diamond(m)$ (slicing away all the nodes and edges below the long diagonal) may require more than half the number of edges in $E_\Diamond(m)$;
- constructing $E_\Diamond(m)$ from two copies of $E_\vartriangle(m)$ may result in a set of edges with a diameter that is twice $d_\vartriangle(m)$, which is sub-optimal.

Theorem 3. *Given an enforcing set of edges for T_a and T_b, we can construct an enforcing set of edges for T_{ab} such that*

$$\begin{aligned} e_\vartriangle(ab) &= a^2 e_\vartriangle(b) + b^2 e_\vartriangle(a) \\ d_\vartriangle(ab) &= \max\left\{d_\vartriangle(a) + d_\vartriangle(b), d_\vartriangle(a) + 2d_\vartriangle(b) - 1\right\}. \end{aligned}$$

Proof. Given an enforcing set of edges for T_a and T_b, we can construct an enforcing set of edges for D_b such that $e_\Diamond(b) = 2e_\vartriangle(b)$ and $d_\Diamond(b) = 2d_\vartriangle(b)$, simply by creating two copies of $E_\vartriangle(b)$ and glueing the leaf nodes together. Moreover, we can partition T_{ab} into t_{a-1} copies of D_b, where t_n denotes the nth triangle number $\frac{1}{2}n(n+1)$, and a copies of T_b. Hence, we have

$$e_\vartriangle(ab) = b^2 e_\vartriangle(a) + \frac{1}{2}a(a-1)e_\Diamond(b) + a e_\vartriangle(b);$$

replacing $e_\Diamond(b)$ by $2e_\vartriangle(b)$ we obtain the required result. Now suppose that $x, y \in T_{ab}$ with $x > y$. Then

- either x and y each belong to copies of D_b, in which case we require at most $d_\vartriangle(a) - 1 + d_\Diamond(b) = d_\vartriangle(a) - 1 + 2d_\vartriangle(b)$ steps to get from x to y,
- or x belongs to a copy of D_b and y belongs to a copy of T_b, in which case we require at most $d_\vartriangle(a) + d_\vartriangle(b)$ steps to get from x to y.

The result follows. □

Theorem 4. *Let $m = ab$. Given a set of enforcing edges for T_a, T_b, D_a and D_b such that $d_\triangle(a) < d_\diamondsuit(a)$ and $d_\triangle(b) < d_\diamondsuit(b)$, there exists a set of enforcing edges $E_\triangle(m)$ such that*

$$e_\triangle(m) < b^2 e_\triangle(a) + \frac{1}{2}a(a+1)e_\diamondsuit(b) \quad \text{and} \quad d_\triangle(m) \leqslant d_\diamondsuit(a) + d_\diamondsuit(b) - 2.$$

Proof. The result for $e_\triangle(m)$ follows because T_m is contained in T_a copies of D_b. The result for $d_\triangle(m)$ follows because for $x, y \in T_m$ with $x > y$ we have

- either x and y each belong to copies of D_b, in which case we require at most $d_\triangle(a) - 1 + d_\diamondsuit(b)$ steps to get from x to y,
- or x belongs to a copy of D_b and y belongs to a copy of T_b, in which case we require at most $d_\triangle(a) + d_\triangle(b)$ steps to get from x to y.

Therefore, given enforcing sets of edges $E_\triangle(a)$, $E_\triangle(b)$ and $E_\diamondsuit(b)$, we have

$$d_\triangle(ab) = \max\{d_\triangle(a) + d_\triangle(b), d_\triangle(a) - 1 + d_\diamondsuit(b)\}.$$

Given that $d_\triangle(b) < d_\diamondsuit(b)$, then we may conclude that

$$d_\triangle(ab) = d_\triangle(a) + d_\diamondsuit(b) - 1.$$

And given $d_\triangle(a) < d_\diamondsuit(a)$, then we may also conclude that

$$d_\triangle(ab) \leqslant d_\diamondsuit(a) + d_\diamondsuit(b) - 2.$$

□

5 Comparison with Related Work

We are not aware of any results in the literature on improving the efficiency of cryptographic enforcement schemes for D_m. However, there is a considerable body of work on the efficiency of schemes for T_m [5,6,10,14]. Our results in this paper are directly comparable to this prior work, with the exception of the work of Crampton [10]. This latter work solves a slightly simpler problem, which requires that it should be possible to get from node $[x, y] \in T_m$ to all nodes of the form $[z, z]$, where $z \in [x, y]$. However, Crampton's work does provide the inspiration for the binary and additive decomposition techniques used in this paper.

In Fig. 6(a) we summarize some of our results for D_m and in Fig. 6(b) we summarize known results for T_m. Theorems 3 and 4 show that, to a good first approximation, our constructions for T_m require half the number of edges as those for D_m and the diameter is reduced by 2. For example, we can use the first row in Fig. 6(a) to deduce that there exists a scheme for T_m such that $e_\triangle(m) \approx m^2 \log m$ and $d_\triangle(m) = 4 \log\log m - 2$.

	$e_{\Diamond}(m)$	$d_{\Diamond}(m)$
Corollary 1	$2m^2 \log m$	4
Corollary 5	$2m^2(\log m - 1)$	$4 \log\log m$
Corollary 3	$\frac{5}{4}m^2 \log m$	$\log m$
Proposition 2	$m^2 \log m$	$2 \log m$
Corollary 4	$4m(m-1)$	$4(\sqrt{m} - 1)$

(a) Our results for D_m

	$e_{\triangle}(m)$	$d_{\triangle}(m)$
[14, §3.1]	$\mathcal{O}\left(m^2 \log m \log\log m\right)$	3
[5, §4]	$\mathcal{O}\left(m^2 \log m\right)$	4
[5, §4]	$\mathcal{O}\left(m^2\right)$	$\mathcal{O}\left(\log^* m\right)$
[14, §3.1]	$\mathcal{O}\left(m^2 \log m\right)$	$\mathcal{O}\left(\log^* m\right)$
[14, §3.1]	$\mathcal{O}\left(m^2\right)$	$\mathcal{O}\left(\log m \log^* m\right)$

(b) Existing results for T_m

Fig. 6. New and existing results for D_m and T_m

At first sight, our results do not appear to improve on those in the literature. However, there are a number of reasons why we believe our results are important and useful. First, they provide explicit bounds for the storage and derivation costs, something which other schemes do not. Secondly, the constructions of our enforcing edge sets are very simple to describe and implement. It is a measure of the complexity of existing work in this area that we have not been able to determine what multiplicative constants and lower terms are "hidden" by the $\mathcal{O}$ expressions in Fig. 6(b). For large values of m, it is clearly useful to understand the asymptotic behavior, but, for smaller (and arguably more relevant) values of m, our approach is more informative. And for values of m that are likely to be of practical interest, it is surely more valuable to know the exact complexity of the scheme and to have simple constructions for the enforcing set of edges.

6 Concluding Remarks

Temporal authorization policies and policies for controlling access to layered multimedia content are examples of policies that are suitable for cryptographic enforcement. The out-sourcing of data storage and the encryption of layered multimedia content makes the efficient cryptographic enforcement of such policies increasingly important. In this paper, we have studied the trade-offs that are possible between the space required for the information that is used to derive decryption keys and the time taken to derive those keys. We have specified explicit constructions for two different authorization graphs $(D_m, \leqslant)$ and $(T_m, \subseteq)$

and illustrated the relationship between these two graphs, making use of this relationship to derive schemes for T_m from those we have constructed for D_m.

There are a number of interesting questions that remain for future work. First, it would be interesting to know whether the construction of an enforcing set of edges for T_m from a set for D_m is the optimal strategy or can we do better by working directly with T_m. Second, the construction for D_{16} from copies of D_4, illustrated in Fig. 4, requires fewer edges than would be obtained by directly applying the result in Corollary 5. This is because the number of edges derived in Corollary 1, and used in Corollary 5, is not a tight bound. We hope to investigate bottom-up approaches (rather than the top-down approach used in the proof of Corollary 5) to determine tighter upper bounds on the number of edges required for a scheme with $\mathcal{O}(\log \log m)$ key derivation. Third, we have not considered the trade-offs that are possible if we assume that users may possess more than one secret value. Atallah et al. [5] and De Santis et al. [14] have each considered these issues for T_m and shown that substantial improvements in $e_\triangle(m)$ or $d_\triangle(m)$ are possible. We hope to address these questions in the near future.

Acknowledgements. The author would like to thank the anonymous referees for their helpful comments.

References

1. Akl, S., Taylor, P.: Cryptographic solution to a problem of access control in a hierarchy. ACM Transactions on Computer Systems 1(3), 239–248 (1983)
2. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM Transactions on Information and System Security 12(3), 1–43 (2009)
3. Atallah, M., Blanton, M., Frikken, K.: Key management for non-tree access hierarchies. In: Proceedings of 11th ACM Symposium on Access Control Models and Technologies, pp. 11–18 (2006)
4. Atallah, M., Blanton, M., Frikken, K.: Efficient techniques for realizing geo-spatial access control. In: Proceedings of the 2007 ACM Symposium on Information, Computer and Communications Security, pp. 82–92 (2007)
5. Atallah, M., Blanton, M., Frikken, K.: Incorporating temporal capabilities in existing key management schemes. In: Proceedings of the 12th European Symposium on Research in Computer Security, pp. 515–530 (2007)
6. Ateniese, G., De Santis, A., Ferrara, A., Masucci, B.: Provably-secure time-bound hierarchical key assignment schemes. Cryptology ePrint Archive, Report 2006/225 (2006), `http://eprint.iacr.org/2006/225.pdf`
7. Bertino, E., Carminati, B., Ferrari, E.: A temporal key management scheme for secure broadcasting of XML documents. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 31–40 (2002)
8. Bodlaender, H., Tel, G., Santoro, N.: Trade-offs in non-reversing diameter. Nordic Journal of Computing 1(1), 111–134 (1994)
9. Christopoulos, C., Skodras, A., Ebrahimi, T.: The JPEG2000 still image coding system: An overview. IEEE Transactions on Consumer Electronics 46(4), 1103–1127 (2000)

10. Crampton, J.: Practical constructions for the efficient cryptographic enforcement of interval-based access control policies. To appear in ACM Transactions on Information and System Security (2011), http://arxiv.org/abs/1005.4993
11. Crampton, J., Martin, K., Wild, P.: On key assignment for hierarchical access control. In: Proceedings of 19th Computer Security Foundations Workshop, pp. 98–111 (2006)
12. Davey, B., Priestley, H.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (2002)
13. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM Transactions on Database Systems 35(2) (2010)
14. De Santis, A., Ferrara, A., Masucci, B.: New constructions for provably-secure time-bound hierarchical key assignment schemes. Theoretical Computer Science 407(1-3), 213–230 (2008)
15. Denning, D.: A lattice model of secure information flow. Communications of the ACM 19(5), 236–243 (1976)
16. Dushnik, B., Miller, E.: Partially ordered sets. American Journal of Mathematics 63, 600–610 (1941)
17. Li, W.: Overview of fine granularity scalability in MPEG-4 video standard. IEEE Transactions on Circuits and Systems for Video Technology 11(3), 301–317 (2001)
18. Thorup, M.: Shortcutting planar digraphs. Combinatorics, Probability & Computing 4, 287–315 (1995)
19. Yao, A.C.: Space-time tradeoff for answering range queries (extended abstract). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 128–136 (1982)
20. Zhu, B., Feng, S., Li, S.: An efficient key scheme for layered access control of MPEG-4 FGS video. In: Proceedings of the 2004 IEEE International Conference on Multimedia and Expo., vol. 1, pp. 443–446 (2004)

Socially Constructed Trust for Distributed Authorization

Steve Barker[1] and Valerio Genovese[2,3,⋆]

[1] King's College London, UK
[2] University of Torino, Italy
[3] University of Luxembourg, Luxembourg

Abstract. We describe an approach for distributed access control that is based on the idea of using a community-constructed repository of expressions of propositional attitudes. We call this repository an *oracle*. Members of a community may consult the oracle and use the expressions of belief and disbelief in propositions that are expressed by community members about requesters for access to resources. Our conceptual model and access control policies are described in terms of a computational logic and we describe an implementation of the approach that we advocate.

1 Introduction

An important aspect that makes distributed access control different to centralized scenarios, is that a reference monitor may have insufficient knowledge of requesters to decide whether to grant or deny access to a guarded resource. Hence, the reference monitor may need to "complete" its knowledge by relying on the testimony supplied to it by one or more trusted third parties.

Logic has had a prominent role to play in the specification, reasoning and enforcement of access control policies in which testimonial knowledge is distributed among several peers (also called principals). In particular, logic programming has been used for specifying policies and for implementing primitives to aggregate principals' knowledge.[1] However, such primitives present serious challenges when dealing with real-world, highly-distributed, large-scale and open access control scenarios like e-trading systems, social networks or microdata publishing platforms (e.g., Twitter). In such cases, the reference monitor may need to remotely access the testimonial knowledge of *large communities* of principals and this knowledge may be *temporarily missing* (e.g., some remote source is not available) or even *conflicting* (e.g., two principals in the same community provide contradictory information). Unfortunately, existing access control primitives to aggregate distributed knowledge do not scale to large communities of principals and provide limited support for dealing with incomplete or conflicting information. In this paper we address the following research question: *How to define an access control framework in which testimonial knowledge results from the aggregating of information issued by communities of principals?*

[⋆] Valerio Genovese is supported by the National Research Fund, Luxembourg.

[1] Examples of such primitives are delegation structures [18], boolean principals [1], dynamic or static thresholds [23,17].

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 262–277, 2011.

This main question reduces to a number of subquestions: How to aggregate testimony and reason with individual and aggregated testimony? How to define and implement operators to query remote knowledge bases? How to deal with conflicting information that stems from the aggregation of principals' testimonial warrant? How to define a specification language that admits nonmonotonic access control policy formulation? What operational methods can be used to evaluate access requests in a correct and computationally viable manner?

In answering these questions we describe an architecture that is based on the idea of using logical databases called "oracles" that speak-for a community on matters relating to requesters of access to resources. We present an access control framework based on Answer Set Programming (ASP) which extends the meta-model of access control introduced in [4] in several ways. We introduce primitives to accommodate the community-based view that we propose. In particular, we take testimonial knowledge as being expressed in the form of *propositional attitudes reports* of type "principal A believes ψ" or "principal A disbelieves ψ". Such testimonial knowledge is collectively stored by oracles. Finally, we extend the ASP system DLV [16] to include scalable primitives for querying several different remote knowledge bases during policy evaluation.

The remainder of the paper is organized as follows. In Section 2, we describe some details on which our policy specification language is based. In Section 3, we describe the architecture of our proposed system and the types of rules we allow for policy specification. In Section 4, we describe examples of the use of access control policy specifications by "oracles" and, in Section 5, we describe examples of the use of specifications of access control policies by reference monitors (called "acceptors") that make use of (or choose to accept) the policy information that is maintained by oracles. In Section 6, we describe some practical issues of relevance to our approach. In Section 7, we describe related work and, in Section 8, we draw conclusions and suggest further work.

2 A Community Security Language

In this section, we introduce the main syntactic and semantic concepts of our $\mathcal{C}$ommunity $\mathcal{S}$ecurity $\mathcal{L}$anguage ($\mathcal{CSL}$). The main sorts of constants for $\mathcal{CSL}$ are: a countable set $\mathcal{C}$ of categories, where $c_0, c_1, \ldots$ are (strings) that are used to denote arbitrary category identifiers; a countable set $\mathcal{P}$ of principals, where $p_0, p_1, \ldots$ are used to identify users, organizations, processes, ...; a countable set $\Sigma \subseteq \mathcal{P}$ of *sources* of testimonial knowledge, where s_0, s_1, $\ldots$ are (strings) used to denote arbitrary sources of testimony; a countable set $\mathcal{A}$ of named atomic *actions*, where a_0, $a_1, \ldots$ are (strings) used to denote arbitrary action identifiers; and a countable set $\mathcal{R}$ of *resource identifiers*, where r_0, r_1, $\ldots$ denote arbitrary resources; $r(t_1, \ldots, t_n)$ is an arbitrary n-place relation that represents an "information resource" where t_i $(1 \leq i \leq n)$ is a term (a term is a function, a constant or a variable).

Informally, a category is any of several fundamental and distinct classes or groups to which entities may be assigned (cf. [4]). Roles, security classifications, security clearances, status levels etc., which are used in various access control models (see, for example, [6,8,22]), are particular instances of what we call categories. The categories of interest, for what we propose, are application-specific and are determined by usage (individual, community or universal) rather than being defined by necessary and sufficient conditions. In financial applications, for example, there may be general categories like "creditworthy" or "bad debtor" that are part of the shared ontology of a community that is interested in testimonial knowledge about e-traders, for instance.

The part of $\mathcal{CSL}$ that we use to specify access control policies is centered on three theory-specific predicates, pca, $arca$ and par, which have following meanings (cf. [4]): $pca(p, c)$ iff the principal $p \in \mathcal{P}$ is assigned to the category $c \in \mathcal{C}$; $arca(a, r, c)$ iff the privilege of performing action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$ is assigned to the category $c \in \mathcal{C}$; $par(p, a, r)$ iff the principal $p \in \mathcal{P}$ is authorized to perform the action $a \in \mathcal{A}$ on resource $r \in \mathcal{R}$. The par relation is defined in terms of $arca$ and pca by the following rule: $\forall p, a, r, c(arca(a, r, c) \wedge pca(p, c) \rightarrow par(p, a, r,))$ that reads as follows: "If a principal p is assigned to a category c to which the privilege to perform action a on resource r has been assigned, then p is authorized to perform the a action on r".

In our community-based approach to testimonial knowledge, multiple testifiers may make assertions of their *propositional attitudes* [21] to a community-based repository (a database) of assertions, which is maintained by an oracle and constitutes a store of propositional attitudes as a set of triples (s_i, α, ψ) such that: s_i $(1 \leq i \leq n)$ is a source of assertions in a community of sources $\Sigma = \{s_1, \ldots, s_n\}$ of testimonial knowledge. ψ is a proposition; α is a propositional attitude that a source s_i has in relation to ψ (e.g., s_i "believes" ψ, s_i "disbelieves" ψ).

The triples (s_i, α, ψ) are used to represent a particular type of *that*-clause expressed in terms of a particular predicate, the pca predicate. In $\mathcal{CSL}$, we represent propositional attitudes by using a 3-place $assertion$ predicate. That is, $assertion(s_i, \alpha, \psi)$ is included, in a repository maintained by an oracle, to express that the source s_i has the propositional attitude α in relation to ψ, e.g., s_i "believes that" ψ is represented as $assertion(s_i, believes, \psi)$. For example, K_{Alice} may believe that K_{Bob} is a "bad debtor".

We restrict attention to the propositional attitudes "believes" and "disbelieves" in this paper. We interpret s_i believes ψ to be source s_i holding ψ to be "true"; by s_i disbelieves ψ we intend disbelieves to mean that s_i rejects the belief that ψ. In the case of believes and disbelieves, the propositional attitudes are subjective; they are used to express the opinions of a testimonial source.

Believes and disbelieves are propositional attitudes that a source may have in relation to the atomic proposition ψ or its negation, $\neg\psi$. In this paper, we assume the following (asymmetric) semantics for believes/disbelieves. If a source s_i asserts that it disbelieves ψ $(\neg\psi)$ then that does not commit s_i to asserting that it believes $\neg\psi$ (ψ). However, a source s_i that asserts that it believes ψ $(\neg\psi)$ implicitly asserts that it disbelieves $\neg\psi$ (ψ). In our approach, ψ is a proposition on a principal-category assignment, i.e., $pca(p, c)$. A source s believing that $\neg pca(p, c)$ believes that ψ should be prohibited from being assigned to the category c and so s must disbelieve that ψ can be assigned to c (otherwise, s would not be rational). Similarly, a source s that believes that $pca(p, c)$ must disbelieve $\neg pca(p, c)$. A source that does not assert what its propositional attitude is in relation to $pca(p, c)$ is said to *suspend* its judgement on the assignment of the principal p to category c (suspension is appropriate in the case where, for instance, s has no information about a principal-category assignment or s has conflicting evidence on what the assignment should be). Suspension of judgement by source s on the attitude α in relation to proposition ψ just means, in our scheme, that there is no $assertion(s, \alpha, \psi)$ fact in an oracle's repository.

Another way of understanding our intended semantics is in terms of normative *ought* assertions. The testifier s_i disbelieving $pca(p, c)$ means that s_i asserts that the belief that the principal p is assigned to the category c ought to be rejected by the community. That is, disbelieving is the way in which testifiers express their dissenting to the beliefs of other testifiers in a community. The example that follows illustrates the different epistemic positions, expressed in terms of ought, that we allow.

Example 1. Consider the following scenario, in which there are four contributors of community testimony ($K_\alpha, K_\beta, K_\gamma$ and K_δ) on a principal K_{Alice} and one category $preferred$:

> *The contributor K_α believes that K_{Alice} ought to be assigned to the category $preferred$ because K_{Alice} satisfies K_α's criterion for that assignment. In contrast, it is K_β's position that the community ought to disbelieve the assertion that K_{Alice} has $preferred$ status (i.e., K_β recommends rejecting the assertion that K_{Alice} should be held to be categorized as $preferred$). The testifier K_γ has had several bad experiences with K_{Alice} and therefore K_γ asserts that it believes that K_{Alice} is definitely not to be assigned to the $preferred$ category. Finally, K_δ's position is that K_{Alice} ought not to be categorized as definitely not $preferred$, but K_δ does not want to assert that K_{Alice} is definitely $preferred$.*

For this policy, the following assertions are included in an oracle's repository of testimony:

$$assertion(K_\alpha, believes, pca(K_{Alice}, preferred)).$$
$$assertion(K_\beta, disbelieves, pca(K_{Alice}, preferred)).$$
$$assertion(K_\gamma, believes, \neg pca(K_{Alice}, preferred)).$$
$$assertion(K_\delta, disbelieves, \neg pca(K_{Alice}, preferred)).$$

Notice that it is up to a community to decide what propositional attitudes it wishes to admit and how those attitudes are to be understood in the community. We also note that propositional attitudes may be tensed, e.g., "believed", indexed by time, and epistemically guarded (e.g., weakly believed or strongly disbelieved), but we do not consider such possibilities in this paper because of limitations on space.

3 Distributed Architecture for Policy Specification

In Figure 1, the architecture of our abstract access control framework is illustrated in overview. There are four main types of entities: *resources, acceptors, oracles* and *contributors*. As we have said, assertions of propositional attitudes on pca definitions are contributed by community members to oracles and are then used by acceptors, of an oracle's assertions, to define authorizations in terms of par. Acceptors act as reference monitors and define the policy to access a guarded resource. Oracles serve a community of contributors (i.e., principals) in terms of making assertions that are of value to acceptors in evaluating access requests. Contributors or other external principals may

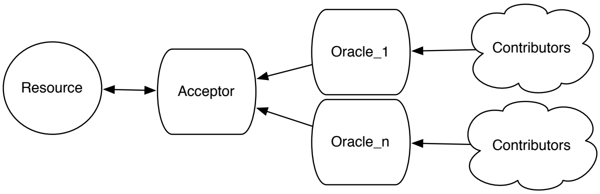


Fig. 1. Abstract Model

request access to resources that are protected by acceptors As our concern is to capture a community-based view of testimony, we need to be able to talk about all members of a community having a particular propositional attitude α on proposition ψ, or some source having attitude α on ψ, or the majority attitude on ψ being α. For that, we add "counting operators" to $\mathcal{CSL}$. These operators range over the pca predicate and have a semantics that may be informally understood as follows: $\Box^{\alpha}(pca(p, c))$ is "true" iff every source of testimonial knowledge in a community Σ has the attitude α in relation to the assignment of a principal p to category c; $\Diamond^{\alpha}(pca(p, c))$ is "true" iff some source of testimonial knowledge in Σ has the attitude α in relation to $pca(p, c)$; $M^{\alpha}(pca(p, c))$ is "true" iff the majority of sources of testimonial warrant in Σ have the attitude α on $pca(p, c)$. As we restrict attention to the propositional attitudes "believes" and "disbelieves" in this paper, assertions about aggregations of pca will henceforth be expressed in the form $\bigcirc^{B^+}(pca(p, c))$ or $\bigcirc^{B^-}(pca(p, c))$ where $\bigcirc \in \{\Box, \Diamond, M\}$ and B^+ is short for "believes that" and B^- is short for "disbelieves that". In addition, we introduce into $\mathcal{CSL}$ a particular connective @ for external query evaluation over remote policy bases such that φ @ ω expresses that: "At remote source ω, φ is true", where φ can be either a literal or an aggregate function.

For the specification of the access control policies that we will introduce, we use standard ASP-DLV syntax with aggregate functions (in the sense of [10]). An *aggregate function* has the form $f(S)$ where S is a set, and f is a function name from the set of function names $\{\#count, \#sum, \#max, \#min, \#times\}$.

Definition 1 (Access Control Policy). *An access control policy is a set of rules of the form $h \leftarrow b_1, \ldots, b_n$ where, h is a literal L or a counting operator applied to an instance of $\bigcirc pca(\_, \_)$ with $\bigcirc \in \{\Box^{\alpha}, \Diamond^{\alpha}, M^{\alpha}\}$; $\alpha \in \{B^+, B^-\}$ and;*

$$b_i := (not)L \mid (\neg)\bigcirc(\neg)pca(\_, \_) \mid assertion(\_, \_, (\neg)pca(\_, \_)) \mid L \,@\, \omega \mid \\ L_g \prec_1 f(S) \prec_2 R_g \mid L_g \prec_1 f(S) \,@\, \omega \prec_2 R_g$$

We note that $L_g \prec_1 f(S) \prec_2 R_g$ is an *aggregate atom* where $f(S)$ is an aggregate function, $\prec_1, \prec_2 \in \{=, <, \leq, >, \geq\}$; L_g and R_g (called *left guard*, and *right guard*, respectively) are terms; and $\neg$ is strong negation [3]. We use aggregates and comparison operators on the numbers of sources of testimonial knowledge to allow acceptors to specify flexibly and in high-level terms the degrees of testimonial support that are required by them in deciding access requests (e.g., whether all, some, or a majority of testifiers have the attitude α with respect to $pca(p, c)$). As is conventional, variables in policy rules will appear in the upper case; constants are in lower case. We also restrict attention to policies with non-recursive aggregates [12] that are defined as a finite set of safe stratified clauses. This means that our access control policies have a unique answer set. We assume that all community members express their access control policies in a language that has a unique answer set as its intended meaning.

A policy rule of the form $\bot \leftarrow b_1 \ldots b_n$ (where $\bot$ is *falsum* and denotes an arbitrary contradiction) is called a *constraint*; a constraint is to be understood (informally) as asserting that it is impossible for $b_1 \ldots b_n$ to be "true" (or provable) simultaneously.

The pca predicate occurring, within the scope of a counting operator, may be negated by using the strong negation operator $\neg$ (but not the weak negation operator not [3]). On this, we may have: $\Box^\alpha(\neg pca(p,c))$ is "true" for a community Σ iff every source of testimonial knowledge in Σ adopts the attitude α on principal p not being assigned to category c; $\Diamond^\alpha(\neg pca(p,c))$ is "true" for a community Σ iff some source of testimonial knowledge in Σ adopts the attitude α on principal p not being assigned to category c; $M^\alpha(\neg pca(p,c))$ is "true" for a community Σ iff the majority of sources of testimonial knowledge in Σ have the attitude α about principal p not being assigned to category c; Moreover, strong negation may be applied to the counting operators that we admit: $\neg\Box^\alpha(pca(p,c))$ is "true" for a community Σ iff it is not the case that every source of testimonial knowledge in Σ asserts that it has the attitude α on $pca(p,c)$; $\neg\Diamond^\alpha(pca(p,c))$ is "true" for a community Σ iff no source of testimonial knowledge in Σ asserts that it has the attitude α about $pca(p,c)$; $\neg M^\alpha(pca(p,c))$ is "true" for a community Σ iff it is not the case that the majority of sources of testimonial knowledge in Σ assert that they have the attitude α about $pca(p,c)$.

The meanings above for $\neg\Box^\alpha$, $\neg\Diamond^\alpha$ and $\neg M^\alpha$ extend to $\neg pca(p,c)$. For example, $\neg\Box^\alpha(\neg pca(p,c))$ is "true" for a community Σ iff not every source of testimonial knowledge in Σ has the attitude α on the proposition $\neg pca(p,c)$ (and *mutatis mutandis* for the other cases).

It should be noted that, for any literal L, $not\ L$ and $not\ \neg L$ are the only uses of the nonmonotonic negation-as-failure (NAF) not operator that we allow in $\mathcal{CSL}$, iterated modalities are not permitted in the form of $\mathcal{CSL}$ used in this paper, and uses of double $\neg$-negation are not permitted at all in $\mathcal{CSL}$.

By using counting functions, we are able to define the semantics of $\Box^\alpha$, $\Diamond^\alpha$, various forms of the M^α operator, additional operators for policy specification and a range of comparison conditions on the assertions made by contributing testifiers. We also allow arithmetic and comparison operators in the set, $\{<, \leq, >, \geq, +, -, \div, \times\}$ in $\mathcal{CSL}$. These operators, which have their usual interpretation on numbers, may also be used to express access control requirements, e.g., if the number of testifiers that believe that $pca(K_\alpha, c')$ holds exceeds the number that believe $\neg pca(K_\alpha, c')$ by at least 5 then source s' accepts that K_α is assigned to the category c'. Moreover, several connections between operators can be identified, e.g., $\Diamond^\alpha pca(P,C) \leftarrow \Box^\alpha pca(P,C)$ holds for a non-empty community of testifiers and additional operators may be added to more expressive forms of $\mathcal{CSL}$. For example, $W^\alpha\ pca(p,c)$ may be introduced to define a "minority" operator, possibly expressed in terms of M^α: $W^\alpha pca(p,c) \leftarrow \neg M^\alpha pca(p,c)$.

As we have said, a community decides what operators, propositional attitudes, and literals it wishes to talk in terms of when expressing access control requirements and therefore what general rules (axioms) apply to the community of users. The following set of rules apply to the interpretation of community security that we have described (where the notation $(\neg)pca(p,c)$ is to be understood as allowing either $\neg pca(p,c)$ or $pca(p,c)$ literals to be expressed, but not both):

(1) $\bot \leftarrow L \wedge \neg L$
(2) $\bot \leftarrow \neg \bigcirc (\neg)pca(P,C) \wedge \bigcirc(\neg)pca(P,C)$
(3) $\neg\Diamond^{\alpha}\neg pca(P,C) \leftarrow \Box^{\alpha} pca(P,C)$
(4) $\neg\Box^{\alpha}\neg pca(P,C) \leftarrow \Diamond^{\alpha} pca(P,C)$
(5) $\bot \leftarrow assertion(S, believes, pca(P,C)), assertion(S, disbelieves, pca(P,C))$
(6) $assertion(S, disbelieves, \neg pca(P,C)) \leftarrow assertion(S, believes, pca(P,C))$
(7) $assertion(S, disbelieves, pca(P,C)) \leftarrow assertion(S, believes, \neg pca(P,C))$
(8) $\neg\Diamond^{B^-}((\neg)pca(P,C)) \leftarrow \Box^{B^+}((\neg)pca(P,C))$
(9) $\Box^{B^-}(pca(P,C)) \leftarrow \Box^{B^+}(\neg pca(P,C))$
(10) $\neg\Diamond^{B^+}(pca(P,C)) \leftarrow \Box^{B^-}(pca(P,C))$

Constraint (1) is standard in ASP languages and forces truth assignments to literals to be consistent. Rule (2) is similar to (1) but it applies to any of the operators that we admit (i.e., $\bigcirc \in \{\Box^{\alpha}, \Diamond^{\alpha}, M^{\alpha}\}$). Rules (3) and (4) model the duality of the $\Box$ and $\Diamond$ operators. Constraint (5) says that no source in a community of testifiers can have the propositional attitude of both believing and disbelieving $pca(P,C)$. Rules (6) and (7) capture the semantics of the consequences of believing principal-category assignments, which we described above. Rules (8), (9) and (10) describe the duality of $\Box$ and $\Diamond$ for the counting operators.

4 Oracle Policy Specification

In this section, we illustrate, by examples, the use of oracle policy assertions and the different representations of pca that oracles may specify. The example that follows illustrates the former.

Example 2. Consider the following fragment of policy information $\mathcal{I}_1$ that needs to be represented by an oracle, denoted by ω:

> *Every contributor source of testimonial knowledge, to ω, believes that the principal K_α is assigned to the category $c_1 \in \mathcal{C}$, there is at least one contributor source of testimonial knowledge on assertions of principal-category assignments that believes that the principal K_β is assigned to $c_2 \in \mathcal{C}$, the majority of contributor sources of testimonial knowledge on assertions of principal-category assignments believe that the principal K_γ is assigned to $c_3 \in \mathcal{C}$, every contributor source of testimonial knowledge believes that the principal K_δ is not assigned to $c_4 \in \mathcal{C}$, and no contributor source says that the principal K_ϵ is assigned to the category $c_5 \in \mathcal{C}$.*

To represent the information in $\mathcal{I}_1$, the oracle ω may include the following assertions on aggregated testimony:

$$\Box^{B^+}(pca(K_\alpha, c_1)). \quad \Diamond^{B^+}(pca(K_\beta, c_2)). \quad M^{B^+}(\,pca(K_\gamma, c_3)).$$
$$\Box^{B^+}(\neg pca(K_\delta, c_4)). \quad \neg\Diamond^{B^+}(pca(K_\epsilon, c_5)).$$

In Example 2, the oracle ω expresses its community-based testimony in aggregated form. It should be clear that $\mathcal{CSL}$ is intended to be a high-level *specification* policy language for community-based testimony. This is one reason why we choose to define $\Box$, $\Diamond$ and M in terms of counting functions rather than using *count* explicitly in policy specifications. An atom like $\Box^{B^+}(pca(K_\alpha, c_1))$ is a simplification of a requirement to "count the number of community members that believe that K_α ought to be assigned to the category c_1 and if this count is greater than the total number of contributors then the atom is true". The need to have higher-level operators becomes more acute when a range of majority operators is admitted in $\mathcal{CSL}$.

The next example, illustrates what is possible in terms of policy representation by combining individual assertions on testimony with aggregate testimony from remote sources.

Example 3. Consider the following policy information $\mathcal{I}_2$ maintained by an oracle ω:

> *The consequent $M^{B^+}(pca(P, c_5))$ holds if the majority of sources of testimonial knowledge for the oracle ω_1 believe that P is assigned to the category $c_6 \in \mathcal{C}$, the majority of sources of testimonial knowledge for the oracle ω_2 disbelieve that P is assigned to $c_7 \in \mathcal{C}$, and it fails to be the case that ω's specific contributor source s' believes $\neg pca(P, c_5)$.*

To represent the information in $\mathcal{I}_2$, ω may use the following access control policy rule:

$$M^{B^+}(pca(P, c_5)) \leftarrow M^{B^+}(pca(P, c_6)) \mathbin{@} \omega_1, M^{B^-}(pca(P, c_7)) \mathbin{@} \omega_2,$$
$$not(assertion(s', B^+, \neg pca(P, c_5))).$$

A feature of our approach is that different definitions of the $\Box$, $\Diamond$ and M operators can be naturally accommodated. For example, the careful reader will have noted that, in what we have said thus far, implicitly an oracle maintains a unary relation $source$ (say), the extension of which is the domain of identifiers of sources from which the oracle gathers its testimonial knowledge. In this case, "every source" has the attitude α in relation to $pca(p, c)$ implies quantification over the elements in the extension of $source$. Similarly, $\Diamond^\alpha pca(p, c)$ corresponds to existential quantification over the elements in the extension of $source$. However, as intimated, these are not the only possibilities. Instead of $source/1$, for example, an oracle may maintain a binary form, $source(s, c)$, with the following semantics: *s is a source of testimonial knowledge on the category c.* In this case, quantification is possible with respect to sources that make assertions about *particular categories*. The difference between $source(s)$ and $source(s, c)$ is needed if it is the case that not all sources that contribute to the oracle's "knowledge" necessary make assertions about the category c. In the case where $source(s, c)$ is relevant, the semantics of $\Box$ changes to the following:

> *$\Box^\alpha_c(pca(p, c))$ is "true" according to an oracle ω iff every source that declares to ω that it is a source of testimonial knowledge on c has the attitude α in relation to $pca(p, c)$.*

The next example, illustrates the potential use of this additional operator.

Example 4. An oracle ω may express the following access control policy information using $\mathcal{CSL}$:

$$assertion(\omega, B^+, pca(P, C)) \leftarrow source(S, C), not(assertion(S, B^+, \neg pca(P, C)))$$
$$assertion(\omega, B^+, \neg pca(P, C)) \leftarrow source(S, C), not(assertion(S, B^+, pca(P, C)))$$

As it is defined, $\neg pca(P, C)$ holds according to ω if some source S of testimonial knowledge on the assignment of a principal P to the category C does not believe that the principal P is assigned to C or if some source of testimonial knowledge on C does not believe that $\neg pca(P, C)$ holds.[2]

The M^{α} operator may also be defined in multiple ways, e.g., the oracle ω may specify,

$$\begin{aligned} M^{B^{+}}(pca(P, c_0)) \leftarrow (\#count\{S : assertion(S, B^{+}, pca(P, c_1)\} \mathbin{@} \omega_1) > \\ (\#count\{S : assertion(S, B^{+}, pca(P, c_2))\} \mathbin{@} \omega_2) \end{aligned}$$

to express that the majority of the community contributors of testimonial knowledge believe $pca(P, c_0)$ if the number of contributors to the oracle ω_1 that believe $pca(P, c_1)$ is greater than the number of contributors to the oracle ω_2 that believe $pca(P, c_2)$. Perhaps, for example, ω assigns a principal P to the category c_0 (the $potential\_customer$ category, say) if the number of testifiers that contribute opinions to the oracle ω_1 believe that P is assigned to the category c_1 (the $preferred\_customer$ category, say) and their collective opinion outweighs the number of sources of testimony, that contribute to ω_2, that believe that P is assigned to the category c_2 (a $blacklisted$ category, say).

5 Acceptor Policy Specification

Having described oracle-based access control policy specification in the previous section, we now describe local policy formulation by acceptor agents that make use of an oracle's assertions. In $\mathcal{CSL}$, an acceptor agent defines the par relation (i.e., authorizations) thus: $par(P, A, R) \leftarrow pca(P, C), arca(A, R, C)$.

That is, the principal P has the access privilege A (or P can perform the action A) on the resource R if P is assigned to a category C to which the permission (A, R) is assigned. For example, a principal categorized as "sales manager" is authorized to read the "monthly sales file" msf if the permission $(read, msf)$ is assigned to members of the "sales manager" category. The categories that an acceptor chooses to use to define authorizations that apply locally are defined by it in terms of the categories used to classify principals within a community of sources of testimonial knowledge.

Example 5. Consider the following local access control policy requirements of an acceptor of assertions on principal-category assignments, which are defined with respect to a remotely located oracle ω:

> *For a principal P to be assigned to the acceptor's local category c_0, all contributors to the oracle ω must believe that $pca(P, c_1)$ and at least one member of the community must believe that P is assigned to the category c_2 (according to ω). Moreover, the specific contributor s_0 must assert (via ω) that P is assigned to the category c_3 and the specific contributor s_1 must assert (via ω) that she disbelieves that P should not be assigned to the category c_4.*

[2] Note that the repository of assertions that is maintained by an oracle may contain assertions made by any source in a community including an oracle.

To represent these access control requirements, the following policy specification may be defined by an acceptor:

$$pca(P, c_0) \leftarrow \Box^{B+}(pca(P, c_1)) \mathbin{@} \omega, \Diamond^{B+}(pca(P, c_2)) \mathbin{@} \omega.$$
$$assertion(s_0, B^+, pca(P, c_3)) \mathbin{@} \omega, assertion(s_1, B^-, \neg pca(P, c_4,)) \mathbin{@} \omega.$$

Although we take each $arca$ definition to be local to the acceptor site that controls access to the resource, a community-based approach to $arca$ specifications is possible. It is also possible for acceptors to specify $\neg arca$ definitions, to express that a permission is not to be assigned to a category or the permission is denied. When $\neg arca$ definitions are used, variants of par are possible (e.g., to represent "open" or "closed" policies with different conflict resolution strategies). For example,

$$par(P, A, R) \leftarrow pca(P, C), not\ \neg arca(A, R, C).$$

may be used to express that a principal P has the A access privilege on resource R if P is assigned to a category C and the (A, R) permission is not denied to C.

Moreover, par definitions may be specialized depending on the particular assertions stored by oracles. The example that follows next illustrates this.

Example 6. Consider the following acceptor site's specification of two forms of authorizations:

$$par_1(P, A, r_1) \leftarrow M^{B+}(pca(P, C)) \mathbin{@} \omega, arca(A, r_1, C).$$
$$par_2(P, A, r_2) \leftarrow M^{B+}(pca(P, c')) \mathbin{@} \omega, \neg\Diamond^{B^-}(pca(P, c'))\mathbin{@} \omega, arca(A, r_2, c').$$

The definition of par_1 may be read as follows: "any principal P is permitted to perform any action A on the resource r_1 if the majority of contributors to oracle ω believe that $pca(P, C)$ holds and $arca(A, r_1, C)$ holds locally". The definition of par_2 says: "Any principal P is permitted to perform any action A on the resource r_2 if the majority of contributors to ω believe that $pca(P, c')$ (i.e., the principal is assigned to category c'), none of ω's contributors disbelieve $pca(P, c')$ at ω, and $arca(A, r_2, c')$ holds locally.

The sceptical reader may argue that our approach violates the privacy of principals whose category assignments are shared in an oracle. Nevertheless, we envisage principals being able to opt-out of this type of information sharing if they wish. The sceptic may argue that our use of the nonmonotonic not operator is "unsafe" because authorizations may hold by default, in the absence of knowledge. However, negation-as-failure is used iff it is appropriate for specific policies and for specific definitions of authorizations, e.g., where access is allowed in the absence of a disbelief holding. The sceptic may suggest that what we propose bears no relation to "real" access control models or policies. However, policies defined in terms of RBAC, MAC, DAC, Attribute-based access control, status-based access control etc. can all be represented within our category-based framework (cf. [4]). That is, what we describe is an approach that extends access control models to the community-based level that we are introducing. The sceptic may argue that $\mathcal{CSL}$ is merely an abstract specification language for expressing distributed access control policies and that $\mathcal{CSL}$ has an equivalent representation in existing policy specification languages. However, as we will show next, policies that are specified in $\mathcal{CSL}$ translate into (various) practical languages for implementation and in the related work section we explain why $\mathcal{CSL}$ differs to existing proposals of policy specification languages.

6 Practical Issues

In this section, we present `secommunity`,[3] an implementation of our framework in the DLV system and some performance measures for it. We then consider proving properties of policies in our scheme. In what follows, DLV code fragments are presented using `monospace` font. Henceforth, we view acceptors, oracles and contributors as DLV knowledge bases (KBs). Counting operators can be implemented by using DLV counting functions. For instance, $\Box^{B^+}$, $\Diamond^{B^+}$ and M^{B^+} can be defined as follows[4]:

$$\begin{array}{l}
\mathtt{box(believes, pca(P, C)) :\text{-}assertion(\_, believes, pca(P, C)),} \\
\mathtt{\#count\{S : assertion(S, believes, pca(P, C)), source(S)\} = R,} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{\#count\{Y : source(Y)\} = R.} \\
\mathtt{diamond(pca(P, C)) :\text{-}assertion(S, believes, pca(P, C)), source(S).} \\
\mathtt{majority(pca(P, C)) :\text{-}assertion(\_, believes, pca(P, C)),} \\
\mathtt{\#count\{S : assertion(S, believes, pca(P, C)), source(S)\} = R1,} \\
\qquad\qquad\qquad\qquad \mathtt{\#count\{S : source(S)\} = R2, R3 = R2/2, R1 > R3.}
\end{array}$$

To represent the distributed aspect of our approach, we defined in DLV-Complex[5] two external predicates (also called built-ins):

- $\mathtt{\#at(IP, G)}$ which reads as: "At the remote DLV source indexed by $\mathtt{IP}$, literal $\mathtt{G}$ holds."
- $\mathtt{\#f\_at(IP, \{P : Conj\}, PX)}$, which has several readings depending on the aggregate function $\mathtt{f}$. For instance, If $\mathtt{f} = \mathtt{count}$, then $\mathtt{PX}$ is unified with the value $\mathtt{V}$, the sum of the elements in $\{\mathtt{P : Conj}\}$ at the remote DLV source indexed by $\mathtt{IP}$, i.e., $\mathtt{\#count\{P : Conj\} = V}$.[6]

For each external predicate (e.g., $\mathtt{\#at}$, $\mathtt{\#count\_at}$) we associated a C++ program that queries the remote DLV knowledge base indexed by $\mathtt{IP}$. For instance, the rule

$$\mathtt{L} \text{ :- } \mathtt{\#count\_at(``IP", ``S : assertion(S, believes, pca(p, c))", PX), PX > 3}$$

reads as follows: "If the remote KB located at address IP contains at least three sources that believe that $\mathtt{p}$ is assigned to category $\mathtt{c}$, then literal $\mathtt{L}$ holds true". Due to space constraints, we do not discuss the implementation details.[7] In order to assess the overhead caused by the evaluation of external predicates, we studied the relationship between execution time and the number of external predicates necessary to evaluate a given rule. In particular, we consider the evaluation of rules of the type

$$\begin{array}{ll}
(T_n) & \mathtt{L} \text{ :- } \mathtt{\#count(IP, S : assertion(S, believes, pca(p_1, c_1)), PX), \ldots,} \\
 & \quad \mathtt{\#count(IP, S : assertion(S, believes, pca(p_n, c_n)), PX), PX > m}
\end{array}$$

Informally, the literal $\mathtt{L}$ holds by rule (T_n) if there are at least $\mathtt{m}$ sources at a remote KB (located at $\mathtt{IP}$) that believe $\mathtt{pca(p_1, c_1)}$, $\mathtt{pca(p_2, c_2)}$, ..., $\mathtt{pca(p_n, c_n)}$. In Figure 2,

[3] A preliminary version of `secommunity` with a simplified language has been presented as a short paper in [5].

[4] It may be that counting operators depend on other remote knowledge bases as in Example 3.

[5] `https://www.mat.unical.it/dlv-complex`

[6] Similarly for $\mathtt{f} \in \{\mathtt{sum}, \mathtt{times}, \mathtt{min}, \mathtt{max}\}$.

[7] A working implementation with source code, documentation and examples is available at `http://www.di.unito.it/~genovese/tools/secommunity/index.html`

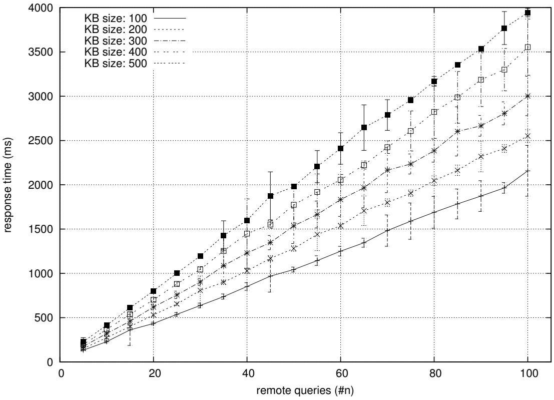


Fig. 2. Scalability of External Predicates

we report the main results of our experiments.[8] For every instance of rule $T_{i \cdot 5}$ (with $1 \leq i \leq 20$), we plot five points representing the average and the standard deviation of the amount of time (in milliseconds) needed to evaluate 30 runs of $T_{i \cdot 5}$ on five different KBs of increasing numbers of belief assertions. In the first KB, there are 100 sources that believe $\texttt{pca}(\texttt{p}_1, \texttt{c}_1), \ldots, \texttt{pca}(\texttt{p}_\texttt{n}, \texttt{c}_\texttt{n})$, in the second the sources supporting pca are 200, and so on up to 500. The results reported in Figure 2 offer some evidence to suggest that our approach is scalable for realistic applications. It is important to note that the time to compute in DLV-Complex T_n grows *linearly* in n and that the time to compute a given rule T_i grows linearly in the size of the KB. We note too that it takes about four seconds to query 100 times a remote KB of 500 assertions with the external predicate #count.

An important practical aspect of our approach is that $\mathcal{CSL}$ admits the possibility of proving properties of policies. To support this assertion, we briefly describe two (of several) types of policy-proof analysis in our framework: *authorization* and *rule-based policy analysis*.

Definition 2 (Access Control Problems). *Consider a policy represented by a set of $\mathcal{CSL}$ rules Π and a set of facts D then,*

- *The* authorization problem *consists in determining whether a principal p_0 is authorized to access resource r_0, which necessitates computing the answer set of $\Pi \cup D$ and checking whether $par(p_0, a_0, r_0)$ is contained in it.*
- *The* rule-based policy analysis *problem consists in taking as input the answer set of a policy and checking whether it is closed under a policy property that is expressible as a rule in $\mathcal{CSL}$.*

Theorem 1 (Complexity of Access Control Problems). *Given a policy $\Pi \ \cup \ D$, the problems reported in Definition 2 can be solved in time* polynomial *in the size of D.*

[8] We conduct our tests on the WAN of University of Torino, the client is a Mac Book Pro with a Intel Core 2 Duo, 4 GB of RAM running Mac OS X 10.6.5. As a server we used a Linux workstation with AMD Athlon, 2 GB of RAM running Ubuntu 10.4.

Proof. (Sketch) Both problems are data complexity *problems [2] for which it is known that the complexity of stratified ASP logic programs is polynomial. Moreover, from [12], the types of aggregate functions admitted in $\mathcal{CSL}$ do not increase the complexity of non-disjunctive ASP programs.*

Suppose next that an acceptor's policy $\Pi \cup D$ is described by a single file "`policy.txt`", which contains the policy expressed as a list of `secommunity` rules and a set of facts. The authorization problem can be solved by letting DLV compute the answer set α of $\Pi \cup D$ by running it on `policy.txt` and then parsing the DLV output looking for the required occurrence of par in α. On the rule-based policy analysis, suppose that a policy administrator wishes to check whether the answer set of `policy.txt` is closed under a given $\mathcal{CSL}$ rule $H \leftarrow L_1 \wedge \ldots \wedge L_n$. Then, it is sufficient to run DLV on `policy.txt` extended with constraint $\mathtt{:-L_1, \ldots, L_n, not\, H}$[9] and to check whether the resulting answer set is empty (i.e., the policy does not satisfy the property) or not (i.e., the policy satisfies the property). Suppose next that a policy administrator wanted to check on `policy.txt` that for any principal p, if p is authorized on action a_1 over resource r_1 then, p is authorized also on action a_2 over resource r_2. For that, it is sufficient to run DLV on `policy.txt` by adding $\mathtt{:-par(P, a_1, r_1), not\, par(P, a_2, r_2)}$ as a constraint. Additionally, suppose that a check of a static separation of duty property of the following type is required: "no principal can be granted for both action a_1 and action a_2 over the same resource r_1". For that, it is sufficient to run DLV on `policy.txt` by adding the following constraint $\mathtt{:-par(P, a_1, r_1), par(P, a_2, r_1)}$.

As a final remark, we emphasize that policy properties like safety and availability [7] follow from the soundness and completeness results of the DLV framework w.r.t. stable model semantics.

7 Related Work

We have described an extension to the work from [4] that allows for multiple distributed access control policies to be represented in a common framework and that allows for different propositional attitudes to be expressed by agents in relation to propositions of relevance to access control. The work described in [4] is concerned neither with distributed access control nor with propositional attitudes.

On the literature on logic programming for access control, our work is related to Jajodia et al.'s specification of the flexible authorization framework in logic programming terms [14] and to Barker and Stuckey's CLP-based approach [7] for access control policy specification. However, both of these approaches assume a centralized access control system and neither of them can express the range of policies that can be expressed in terms of the meta-model that we described in [4]. Neither approach allows for explicit negation to be used in policy specification, a community-based view of testimonial knowledge for authorization, the use of propositional attitudes for expressing policy information, or the use of a range of counting operators for policy specification.

The RT [18] family of trust management languages is also related to our proposal. However, RT is Datalog-based and the various RT languages do not admit the possibility of sources issuing negative assertions and they do not admit a default form of

[9] Notice that $L_1, \ldots, L_2$ may also be remote querying operators or aggregate functions.

negation. RT is based on a monotonic semantics. RT also does not take account of the idea of a community-based source of knowledge (role names could refer to oracles but a doxastic element would need to be added to RT to capture what we propose) or different strengths of knowledge expressed in terms of community-based assertions. SD3 [15] and Binder [11] are also monotonic, Datalog-based languages for expressing policies for distributed access control. As such, neither of these approaches allows for nonmonotonic access controls that we have considered in this paper. The work by Wang and Zhang [23] uses ASP, as we do, for distributed access control policy specification, but their work is based on the use of ASP to implement a variant of D2LP [18]. Our focus is quite different. We also note that RT^T allows for threshold structures that require principals from a single set to agree on an entity having a role's attribute. In $\mathcal{CSL}$, policy requirements that require all contributors to an oracle must agree that a principal-category assignment holds, but also disbeliefs, degrees of disagreement between believers and disbelievers, etc. can be expressed and so too can negated forms of belief and disbelief.

There are several additional logic-based languages (e.g., ABLP logic, DKAL and SecPAL) that, like Binder, attach special importance to a **says** construct (or a variant of it). Put simply, A **says** p is used to associate a principal with an utterance p. At a superficial level, the reader might suggest that A **says** p means A "believes" p (though that this doxastic interpretation is to be adopted it is far from being a universal view and SecPAL, in particular, assumes that principals assert "facts" not beliefs) and that a says-logic enables the concepts expressed in $\mathcal{CSL}$ to be equivalently represented. However, none of ABLP logic, DKAL or SecPAL consider disbeliefs or the subtleties of the interactions between beliefs and disbeliefs. In contrast, we give two distinctions between beliefs and disbeliefs and we give axioms to make plain the formal relationships between them. We also make clear how we interpret the notion of suspension (of beliefs). None of ABLP logic, DKAL or SecPAL are concerned with the notion of trust being grounded in a community view and being represented by using assertions expressed using aggregates. Moreover, all reasoning methods for these logics are *local*, i.e., they do not offer primitives to query distributed knowledge bases. A notable exception is Soutei [20], which is an extension of Binder that accepts remote fetching of assertions. However, Soutei is monotonic, it is based on **says** modality, it does not support aggregate functions and it is not clear that it offers a client-server architecture that can be deployed in real-world scenarios (e.g., the Internet).

Our work is also related to efforts, like SPKI/SDSI [9], that are concerned with certified authorization in distributed computing contexts. On this, Howell and Kotz [13] discuss "beliefs" in the context of their formalization of a variant of SPKI/SDSI, but our reading of their notion of belief suggests that what they have in mind is quite different to our interpretation of "believes that". Howell and Kotz give no formal definition of what they take "belief" to mean in their formalization of distributed authorization; they appear to assume beliefs to be synonymous with assertions of propositions, rather than being an indicator of an epistemic attitude to a proposition. The work by Liau et al. on BIT logic [19] is related to ours in the sense that beliefs in propositions may be expressed in a logic for representing policies for trust management. However, the social aspect of testimony is not considered by Liau et al.

There have been many approaches to trust management that are based on measures of trust (or disbelief or distrust) that are (often) expressed by using real numbers in the interval $[-1, 1]$ as measures of belief/disbelief in some proposition. However, it is often far from clear how, for example, the assignment of trust measures to propositions can be precisely and uncontroversially allocated and it is not clear how changes in evidence justify changes in measures of belief/disbelief. In contrast, we argue that the three epistemic positions that we allow (belief, disbelief and suspension of judgement) provides a basis for defining a meaningful semantics for community-based, distributed access control policies.

8 Conclusions and Further Work

We argued that for some (not all) applications, socially constructed testimony may be usefully employed for distributed access control (that relying on individual, authoritative sources of testimony is not always appropriate). On this, we have described an approach for socially constructed trust.

We have taken testimonial knowledge as being expressed in the form of a testifier's propositional attitude in relation to propositions about principal-category assignments and we view trust in terms of community beliefs. For access control policy specification, we introduced $\mathcal{CSL}$ (Section 3). By adopting a weak negation operator, nonmonotonic policies can be represented in our framework. Acceptors and oracles express access control policy information by using the same specification language. Scalability issues are addressed by subdividing the task of providing testimony between contributors to a community view. For the shared conceptual framework, we extend the meta-model for access control as defined in [4]. To illustrate our approach, we gave a number of examples of oracle-based access control representation (Section 4) and acceptor-based authorization policies (Section 5). We described an ASP-based implementation of our approach (Section 6) and we presented some performance measures for this implementation of our approach that offer evidence of its scalability (Section 6). We argued (in Section 7) that our approach offers something different to existing approaches that either do not consider the community aspect of testimonial knowledge or that do recognize the importance of community constructed knowledge but express aggregated assertions in terms of measures of "reputation".

In future work, we propose to investigate community membership issues (e.g., how to address the effects of changes to the community). Thus far, we have only considered a "democratic" form of community in which every contributor's assertions have the same weight. Accommodating variable measures of authority in terms of the assertions that contributors make to the community view is another issue that requires further investigation. We also intend to investigate issues relating to temporally constrained propositional attitude reports and we propose to include additional types of propositional attitudes (e.g., "knows that" p by proof of p) in extended forms of our framework.

Acknowledgements. The authors would like to thank Daniele Rispoli for the implementation of the external predicates of `secommunity`.

References

1. Abadi, M.: Access control in a core calculus of dependency. Electr. Notes Theor. Comput. Sci. 172, 5–31 (2007)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
3. Baral, C., Gelfond, M.: Logic programming and knowledge representation. J. Log. Program. 19/20, 73–148 (1994)
4. Barker, S.: The next 700 access control models or a unifying meta-model? In: Procs. of SACMAT, pp. 187–196 (2009)
5. Barker, S., Genovese, V.: Secommunity: A framework for distributed access control. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 297–303. Springer, Heidelberg (2011)
6. Barker, S., Sergot, M.J., Wijesekera, D.: Status-based access control. ACM Trans. Inf. Syst. Secur. 12(1) (2008)
7. Barker, S., Stuckey, P.: Flexible access control policy specification with constraint logic programming. ACM Trans. Inf. Syst. Secur. 6(4), 501–546 (2003)
8. Bell, D.E., LaPadula, L.J.: Secure computer system: Unified exposition and multics interpretation. MITRE-2997 (1976)
9. Clarke, D.E., Elien, J.-E., Ellison, C.M., Fredette, M., Morcos, A., Rivest, R.L.: Certificate chain discovery in SPKI/SDSI. J. Computer Security 9(4), 285–322 (2001)
10. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV. In: Procs. of IJCAI, pp. 847–852 (2003)
11. DeTreville, J.: Binder, a logic-based security language. In: Proc. IEEE Symposium on Security and Privacy, pp. 105–113 (2002)
12. Faber, W., Leone, N.: On the complexity of answer set programming with aggregates. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 97–109. Springer, Heidelberg (2007)
13. Howell, J., Kotz, D.: A formal semantics for SPKI. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 140–158. Springer, Heidelberg (2000)
14. Jajodia, S., Samarati, P., Sapino, M., Subrahmaninan, V.: Flexible support for multiple access control policies. ACM TODS 26(2), 214–260 (2001)
15. Jim, T.: SD3: A trust management system with certified evaluation. In: IEEE Symp. Security and Privacy, pp. 106–115 (2001)
16. Leone, N., Faber, W.: The DLV project: A tour from theory and research to applications and market. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 53–68. Springer, Heidelberg (2008)
17. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Trans. Inf. Syst. Secur. 6(1), 128–171 (2003)
18. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: Procs. of IEEE Symposium on Security and Privacy, pp. 114–130 (2002)
19. Liau, C.-J.: Belief, information acquisition, and trust in multi-agent systems–a modal logic formulation. Artif. Intell. 149(1), 31–60 (2003)
20. Pimlott, A., Kiselyov, O.: Soutei, a logic-based trust-management system. In: Hagiya, M. (ed.) FLOPS 2006. LNCS, vol. 3945, pp. 130–145. Springer, Heidelberg (2006)
21. Russell, B.: On denoting. Mind 149(1), 479–493 (1905)
22. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
23. Wang, S., Zhang, Y.: Handling distributed authorization with delegation through answer set programming. Int. J. Inf. Sec. 6(1), 27–46 (2007)

Fully Secure Multi-authority Ciphertext-Policy Attribute-Based Encryption without Random Oracles

Zhen Liu[1,2,*], Zhenfu Cao[1,*], Qiong Huang[3], and Duncan S. Wong[2], and Tsz Hon Yuen[4]

[1] Shanghai Jiao Tong University, Shanghai, China
{liuzhen@,zfcao@cs.}sjtu.edu.cn
[2] City University of Hong Kong, Hong Kong S.A.R., China
{zhenliu7@student.,duncan@}cityu.edu.hk
[3] South China Agricultural University, Guangzhou, China
csqhuang@alumni.cityu.edu.hk
[4] The University of Hong Kong, Hong Kong S.A.R., China
thyuen@cs.hku.hk

Abstract. Recently Lewko and Waters proposed the first fully secure multi-authority ciphertext-policy attribute-based encryption (CP-ABE) system in the random oracle model, and leave the construction of a fully secure multi-authority CP-ABE in the standard model as an open problem. Also, there is no CP-ABE system which can completely prevent individual authorities from decrypting ciphertexts. In this paper, we propose a new multi-authority CP-ABE system which addresses these two problems positively. In this new system, there are multiple Central Authorities (CAs) and Attribute Authorities (AAs), the CAs issue identity-related keys to users and are not involved in any attribute related operations, AAs issue attribute-related keys to users and each AA manages a different domain of attributes. The AAs operate independently from each other and do not need to know the existence of other AAs. Messages can be encrypted under any monotone access structure over the entire attribute universe. The system is adaptively secure in the standard model with adaptive authority corruption, and can support large attribute universe.

Keywords: Attribute based encryption, ciphertext-policy, multi-authority.

1 Introduction

In traditional cryptosystems, messages are encrypted for receivers who are identified by the keys they are holding. The same holds for Identity-Based Encryption (IBE) [20,3] where user public keys are binary strings, such as email

[*] The authors are supported by the National Nature Science Foundation of China No. 61033014, No. 60970110 and No. 60972034, and by the National 973 Program No. 2007CB311201.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 278–297, 2011.

addresses, which uniquely identify the users. In some applications that require access control, messages are required to be encrypted for multiple receivers who are identified by their roles, rather than their actual identities. For example, Alice may want to encrypt some message to all PhD students and alumni in the Department of Computer Science of University X, that is, she wants to do the encryption with an access policy such as ("UNIV.X.COMPUTER SCIENCE" **AND** ("UNIV.X.PhD STUDENT" **OR** "UNIV.X.ALUMNI")), so that only those receivers whose attributes satisfy this policy can perform the decryption successfully. As a counterexample, suppose Bob is a Computer Science undergraduate student, he cannot decrypt even if he colludes with another student, say Tom, who is a PhD student but in the Mathematics department.

Attribute-Based Encryption (ABE) [19] provides a solution to the application above. In ABE (Ciphertext-Policy ABE or CP-ABE as an example), an access policy defined over a set of attributes is associated with each encrypted message, and each user in the system has a private key obtained from an authority (e.g., the UNIV.X Registry) corresponding to the user's attributes (or credentials). If a user's attributes satisfy the access policy of a ciphertext, the user can decrypt the ciphertext. In most of the ABE systems [10,18,2,7,9,21,11,17], attributes are managed by a single authority. In some applications however, this may not desirable. For example, Alice encrypts a message with access policy ("UNIV.X.COMPUTER SCIENCE" **AND** "UNIV.X.ALUMNI" **AND** "GOOGLE.ENGINEER") so that only receivers who are the computer science alumni of University X and currently working as an engineer for Google can decrypt. The authority UNIV.X Registry may only manage attributes for the students, staff and alumni of University X, while Google Registry may be the authority handling its employees' attributes. A single-authority ABE may not be appropriate in this scenario.

Multi-authority ABE systems are proposed to address this issue. For the systems in [5,14,6,15,16], they are selectively secure where the adversary has to commit to the access policy before seeing the public parameters. Recently in [13], Lewko and Waters proposed a new one. Although their system may become inefficient for large attribute universe, it is the first fully secure multi-authority ABE system and is proven secure in the random oracle model. Multi-authority ABE also helps alleviate the extent of trust on authorities. In a single-authority ABE system [19,10,18,2,7,9,21,11,17], the authority can decrypt all ciphertexts. In some multi-authority ABE systems [5,15,16], there is still a central authority which can decrypt all ciphertexts. In [14,6], the multi-authority ABE systems do not have a central authority. They are Key-Policy ABE (KP-ABE), and the techniques do not seem to apply to CP-ABE. In the multi-authority CP-ABE [13], no single authority can decrypt all ciphertexts and each authority can only decrypt ciphertexts that the associated access policy can be satisfied by the authority's own domain of attributes. For example, although neither UNIV.X Registry nor Google Registry alone can decrypt a ciphertext with access policy ("UNIV.X.ALUMNI" **AND** "GOOGLE.ENGINEER"), UNIV.X Registry can decrypt a ciphertext with access policy ("UNIV.X.COMPUTER SCIENCE" **AND** "UNIV.X.ALUMNI").

Table 1. A Comparison between existing work and this work

	Multi-Authority	Adaptively Secure	Standard Model	Prevent Decryption by Individual Authorities	Support Large Attribute Universe	KP/CP
[19,10,18]	×	×	√	×	√	KP
[2]	×	×	×	×	√	CP
[7,9,21]	×	×	√	×	√	CP
[11,17]	×	√	√	×	√	KP+CP
[5]	√	×	√	×	√	KP
[14]	√	×	√	√	√	KP
[6]	√	×	√	√	√	KP
[15,16]	√	×	√	×	√	CP
[13]	√	√	×	Partially	×	CP
this work	√	√	√	√	√	CP

1.1 Our Results

We propose a new multi-authority CP-ABE system which has multiple Central Authorities (CAs) and Attribute Authorities (AAs). The CAs issue identity-related keys to users but do not involve in any attribute-related operations. AAs issue attribute-related keys to users. Each AA manages a different attribute domain and operates independently from other AAs. A party may join the system to be an AA by simply registering itself to the CAs, and then publishing its attribute-related public parameters. In the proposed system, no authority can independently decrypt any ciphertext. We show that the system is adaptively secure in the standard model which captures adaptive authority corruption. Its access policy can be any monotone access structure and the system supports large attribute universe. The efficiency of the system is also comparable to the corresponding single-authority CP-ABE system. Table 1 shows a comparison in properties and security levels between current ABE systems and this new system.

1.2 System Architecture

Fig. 1 shows the architecture of the multi-authority CP-ABE system. The system has D Central Authorities, $CA_1, \ldots, CA_D$, and K Attribute Authorities, $AA_1, \ldots, AA_K$. Each AA manages a different domain of attributes (e.g., AA_1 manages U_1, and so on). When a user joins the system, each CA issues an identity-related key to the user. Then the user obtains an attribute-related key corresponding to the attributes that the user entitled from an AA (e.g., UNIV.X Registry). In practice, one may imagine that there could have multiple CAs run by different organizations while all of them are governed under some ordinance made by the government, then universities and companies can join the system as AAs. Each AA manages its own attribute domain and the AAs operate independently from each other. The trust on the CAs by the users in the system can also be alleviated as it is unlikely to have all the CAs collude if some appropriate governmental policies and business measures are put into place to govern the practice of the CAs.

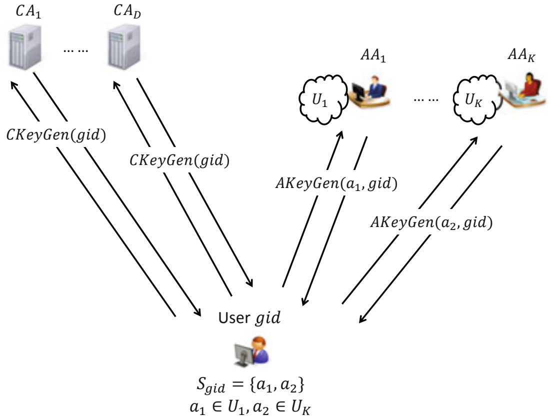


Fig. 1. Architecture of Multi-Authority CP-ABE

1.3 Related Work

Attribute-Based Encryption (ABE) was introduced by Sahai and Waters [19] and classified into Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE) by Goyal et al. [10]. In KP-ABE, attributes are associated with ciphertexts and formulas (or policies) defined over attributes are associated with users' secret keys. In CP-ABE, attributes are associated with users' secret keys and policies are associated with ciphertexts. The single-authority ABE systems proposed in [19,10,18] are KP-ABE, those proposed in [2,7,9,21] are CP-ABE, and [11,17] cover both KP-ABE and CP-ABE.

Chase [5] proposed the first multi-authority ABE system where there are one CA (Central Authority) and multiple AAs (Attribute Authorities). The CA issues identity-related keys to users and the AAs manage attributes and issue attribute-related keys. A user's keys from different AAs are linked together by the user's *global identifier*. The expressiveness of the system is limited and only "AND" policy between the AAs is supported. Also, the CA can decrypt all ciphertexts. Lin et al. [14] remove the CA using a threshold technique where the set of authorities is fixed ahead of time and all authorities must interact during the system setup. The system cannot defend against collusion attack by m or more users where m is a system parameter chosen at setup. Chase and Chow [6] also remove the central authority using a distributed PRF (pseudo random function) technique. However, the expressiveness is as limited as the original Chase's system, and their technique does not seem to apply to CP-ABE. While [5,14,6] focus on KP-ABE, Müller, Katzenbeisser, and Eckert [15,16] proposed the first multi-authority CP-ABE system where there are one CA and multiple AAs. The AAs operate independently from each other and therefore is flexible and practical. However, the CA in the system can still decrypt all ciphertexts.

In [13], Lewko and Waters proposed a new multi-authority CP-ABE system. Different from all previous multi-authority ABE systems, which are all selectively secure, this new system is adaptively secure. The system is expressive, supporting any monotone access structures. There is no central authority and each authority in the system operates independently from other authorities. The system is proven secure in the random oracle model, and does not efficiently support large attribute universe. In addition, each authority can still independently decrypt ciphertexts, if the associated access policies can be satisfied by the attributes managed by the authority.

Paper Organization. In the next section, we define the multi-authority CP-ABE and formalize its security model. Some background such as number-theoretic assumptions and access structures are reviewed in Sec. 3. Our scheme is described in Sec. 4, and some extensions are proposed in Sec. 5. In Sec. 6, the scheme is compared with some existing schemes and the paper is concluded in Sec. 7.

2 Definition and Security Model

There are three sets of entities in a Multi-Authority Ciphertext-Policy Attribute-Based Encryption (MA-CP-ABE) system: (1) Central Authorities (CAs), (2) Attribute Authorities (AAs) and (3) users. Let $CA_1, \ldots, CA_D$ be central authorities and $\mathbb{D} = \{1, \ldots, D\}$ the index set of the CAs, that is, using $d \in \mathbb{D}$ to denote the index of central authority CA_d. Let $AA_1, \ldots AA_K$ be attribute authorities and $\mathbb{K} = \{1, \ldots, K\}$ the index set of the AAs. Each user has a global identifier denoted as *gid*. The CAs are responsible for issuing keys to users according to their global identifiers. The AAs are responsible for issuing keys corresponding to attributes, and each AA manages a different attribute domain (e.g., AA_i manages attributes for a university registry, AA_j manages attributes for a company registry, etc.). Let U_k be the attribute domain managed by AA_k where $U_i \cap U_j = \emptyset$ for all $i \neq j \in \mathbb{K}$, and $U = \bigcup_{k=1}^{K} U_k$ be the attribute universe.

2.1 Definition

An MA-CP-ABE system consists of the following seven algorithms:

$\mathsf{GlobalSetup}(\lambda) \rightarrow (\mathsf{GPK})$. The algorithm takes as input the security parameter λ and outputs the global public parameter GPK of the system.

$\mathsf{CASetup}(\mathsf{GPK}, d) \rightarrow (\mathsf{CPK}_d, \mathsf{CAPK}_d, \mathsf{CMSK}_d)$. Each CA_d runs the algorithm with GPK and its index d as input, and produces master secret key CMSK_d and public parameters $(\mathsf{CPK}_d, \mathsf{CAPK}_d)$. CAPK_d will be used by AAs only.

$\mathsf{AASetup}(\mathsf{GPK}, k, U_k) \rightarrow (\mathsf{APK}_k, \mathsf{ACPK}_k, \mathsf{AMSK}_k)$. Each AA_k runs the algorithm with GPK, its index k and its attribute domain U_k as input, and produces master secret key AMSK_k and public parameters $(\mathsf{APK}_k, \mathsf{ACPK}_k)$. ACPK_k will be used by CAs only.

$\mathsf{Encrypt}(M, \mathbb{A}, \mathsf{GPK}, \{\mathsf{CPK}_d | d \in \mathbb{D}\}, \{\mathsf{APK}_k\}) \rightarrow CT$. The algorithm takes as input a message M, an access policy $\mathbb{A}$ defined over the attribute universe U, the global public parameter GPK, CAs' public parameters $\{\mathsf{CPK}_d | d \in \mathbb{D}\}$, and the related AAs' public parameters $\{\mathsf{APK}_k\}$. It outputs a ciphertext CT which contains the access policy $\mathbb{A}$.

$\mathsf{CKeyGen}(gid, \mathsf{GPK}, \{\mathsf{ACPK}_k | k \in \mathbb{K}\}, \mathsf{CMSK}_d) \rightarrow (\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d})$. When a user with global identifier gid visits CA_d for obtaining a key, CA_d runs the algorithm, which takes as input gid, GPK, $\{\mathsf{ACPK}_k | k \in \mathbb{K}\}$, and CA_d's master secret key CMSK_d. It outputs a ***user-central-key*** $(\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d})$, where $\mathsf{ucpk}_{gid,d}$ is called ***user-central-public-key***.

$\mathsf{AKeyGen}(att, \{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \mathsf{GPK}, \{\mathsf{CAPK}_d | d \in \mathbb{D}\}, \mathsf{AMSK}_k) \rightarrow \mathsf{uask}_{att,gid}$ or $\perp$. When a user requests a secret key for attribute att from AA_k, AA_k runs the algorithm, which takes as input att, $\{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}$, GPK, $\{\mathsf{CAPK}_d | d \in \mathbb{D}\}$ and AMSK_k. If all $\mathsf{ucpk}_{gid,d}$s are valid, the algorithm outputs a ***user-attribute-key*** $\mathsf{uask}_{att,gid}$, otherwise it outputs $\perp$. For a user gid with attribute set S_{gid}, the user's ***decryption-key*** is defined as

$$\mathsf{DK}_{gid} = (\{\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \{\mathsf{uask}_{att,gid} | att \in S_{gid}\}).$$

$\mathsf{Decrypt}(CT, \mathsf{GPK}, \{\mathsf{APK}_k\}, \mathsf{DK}_{gid}) \rightarrow M$ or $\perp$. The algorithm takes as input a ciphertext CT associated with access policy $\mathbb{A}$, GPK, the related attribute authorities' public parameters $\{\mathsf{APK}_k\}$, and a decryption-key DK_{gid} with attribute set S_{gid}. If S_{gid} satisfies the access policy $\mathbb{A}$, the algorithm outputs the message M, otherwise it outputs $\perp$ indicating the failure of decryption.

2.2 Security Model

The security of MA-CP-ABE is defined by the following game run between a challenger $\mathcal{B}$ and an adversary $\mathcal{A}$. $\mathcal{A}$ can corrupt CAs and AAs by specifying $\mathbb{K}_c \subset \mathbb{K}$ and $\mathbb{D}_c \subset \mathbb{D}$ after seeing the public parameters[1], where $\mathbb{D} \setminus \mathbb{D}_c \neq \emptyset$ and $\mathbb{K} \setminus \mathbb{K}_c \neq \emptyset$. Without loss of generality, we assume that $\mathcal{A}$ corrupts all CAs but one, i.e., $|\mathbb{D} \setminus \mathbb{D}_c| = 1$.

Setup

- $\mathsf{GlobalSetup}$, $\mathsf{CASetup}(\mathsf{GPK}, d)$ $(d = 1, \ldots, D)$ and $\mathsf{AASetup}(\mathsf{GPK}, k, U_k)$ $(k = 1, \ldots, K)$ are run by the challenger $\mathcal{B}$. GPK, $\{\mathsf{CPK}_d, \mathsf{CAPK}_d | d \in \mathbb{D}\}$ and $\{\mathsf{APK}_k, \mathsf{ACPK}_k | k \in \mathbb{K}\}$ are given to the adversary $\mathcal{A}$.
- $\mathcal{A}$ specifies an index $d^* \in \mathbb{D}$ as the only uncorrupted CA and specifies a set $\mathbb{K}_c \subset \mathbb{K}$ of AAs to be corrupted where $\mathbb{K} \setminus \mathbb{K}_c \neq \emptyset$. Let $\mathbb{D}_c = \mathbb{D} \setminus \{d^*\}$. $\{\mathsf{CMSK}_d | d \in \mathbb{D}_c\}$ and $\{\mathsf{AMSK}_k | k \in \mathbb{K}_c\}$ are given to $\mathcal{A}$.

Key Query Phase 1. User-central-key and user-attribute-key can be obtained by querying the following oracles:

$\mathsf{CKQ}(gid, d)$ where $d = d^*$: $\mathcal{A}$ queries with a pair (gid, d), where gid is a global identifier and $d = d^*$, and obtains the corresponding user-central-key $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$.

[1] This is stronger than the static corruption model used in [5,6,13], where the adversary has to specify the authorities to corrupt before seeing the public parameters. But on the other aspect, it is weaker than the model in [13], where the corrupted authorities are set by the adversary.

$\mathsf{AKQ}(att, \{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, k)$ where $k \in \mathbb{K} \setminus \mathbb{K}_c$: $\mathcal{A}$ queries with $(att, \{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, k)$, where $k \in \mathbb{K} \setminus \mathbb{K}_c$ is the index of an uncorrupted AA, $\{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}$ are gid's user-central-public-keys, and att is an attribute in U_k. The oracle returns a user-attribute-key $\mathsf{uask}_{att,gid}$ or $\perp$ if $\{\mathsf{ucpk}_{gid,d}\}$ are invalid.

Challenge Phase. $\mathcal{A}$ submits two equal-length messages M_0, M_1, and an access policy $\mathbb{A}$. $\mathcal{B}$ flips a random coin $\beta \in \{0, 1\}$ and sends to $\mathcal{A}$ an encryption of M_β under $\mathbb{A}$.

Key Query Phase 2. $\mathcal{A}$ further queries as in **Key Query Phase 1**.

Guess. $\mathcal{A}$ submits a guess β' for β.

For a gid, the related attribute set is defined as

$$S_{gid} = \{att \mid \mathsf{AKQ}(att, \{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, k) \text{ is made by } \mathcal{A}\}.$$

$\mathcal{A}$ wins the game if $\beta' = \beta$ under the **restriction** that there is no S_{gid} such that $S_{gid} \cup (\bigcup_{k_c \in \mathbb{K}_c} U_{k_c})$ can satisfy the challenge access policy $\mathbb{A}$. The advantage of $\mathcal{A}$ is defined as $|\Pr[\beta = \beta'] - 1/2|$.

Definition 1. *An MA-CP-ABE system is secure if for all polynomial-time adversary $\mathcal{A}$ in the game above, the advantage of $\mathcal{A}$ is negligible.*

Remarks: We assume that a user with global identifier gid requests for the central key from each CA_d only once, i.e., for each gid there is only one set of user-central-keys, $\{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}$. This is not a restriction, but can help simplify the system description. Using obscure notations such as $\mathsf{ucpk}_{gid,d,t}$ and $S_{gid,d,t}$ where t is a time stamp can remove this assumption. In the security model above, $\mathcal{A}$ has the master secret keys $\{\mathsf{CMSK}_d | d \in \mathbb{D}_c\}$, so the user only needs to query $\mathsf{CKQ}(gid, d^*)$ for getting $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$, and the user can generate $\{(\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d}) | d \in \mathbb{D}_c\}$ if they are needed for querying AKQ.

3 Background

3.1 Access Policy

Definition 2 (Access Structure [1]). *Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ is monotone if $\forall$ B,C : if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.*

In ABE, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ contains the authorized sets of attributes. As of previous work in ABE, we focus on monotone access structures in this paper. It is shown in [1] that any monotone access structure can be realized by a linear secret sharing scheme. Here we use the definition from [1,21].

Definition 3 (Linear Secret-Sharing Schemes (LSSS) [21]). *A secret sharing scheme Π over a set of parties $\mathbb{P}$ is called linear (over $\mathbb{Z}_p$) if*

1. *The shares for each party form a vector over $\mathbb{Z}_p$.*
2. *There exists a matrix A called the share-generating matrix for Π. The matrix A has l rows and n columns. For $i = 1, \ldots, l$, the i^{th} row of A is labeled by a party $\rho(i)$(ρ is a function from $\{1, \ldots, l\}$ to $\mathbb{P}$). When we consider the column vector $\boldsymbol{v} = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then $A\boldsymbol{v}$ is the vector of l shares of the secret s according to Π. The share $(A\boldsymbol{v})_i$ belongs to party $\rho(i)$.*

It is shown in [1] that every linear secret-sharing scheme according to the above definition also enjoys the linear reconstruction property, defined as follows: Suppose that Π is an LSSS for access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be an authorized set, and let $I \subset \{1, 2, \ldots, l\}$ be defined as $I = \{i : \rho(i) \in S\}$. There exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $\sum_{i \in I} \omega_i \lambda_i = s$. Furthermore, these constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix A. For any unauthorized set, no such constants exist. In this paper, we use LSSS matrix (A, ρ) to express an access policy associated to a ciphertext.

3.2 Number-Theoretic Assumptions

Our MA-CP-ABE system works on composite order bilinear groups [4]. Let $\mathcal{G}$ be the group generator, which takes a security parameter λ and outputs $(p_1, p_2, p_3, G, G_T, e)$ where p_1, p_2, p_3 are distinct primes, G and G_T are cyclic groups of order $N = p_1 p_2 p_3$, and $e : G \times G \rightarrow G_T$ is a map such that: (1) (Bilinear) $\forall g, h \in G, a, b \in \mathbb{Z}_N, e(g^a, h^b) = e(g, h)^{ab}$, (2) (Non-Degenerate) $\exists g \in G$ such that $e(g, g)$ has order N in G_T. Assume that group operations in G and G_T as well as the bilinear map e are computable in polynomial time with respect to λ. Let G_{p_1}, G_{p_2} and G_{p_3} be the subgroups of order p_1, p_2 and p_3 in G, respectively. Note that for any $h_i \in G_{p_i}$ and $h_j \in G_{p_j}$ where $i \neq j$, $e(h_i, h_j) = 1$.

For an element $T \in G$, T can (uniquely) be written as the product of an element of G_{p_1}, an element of G_{p_2}, and an element of G_{p_3}, and they are referred to as the "G_{p_1} part of T", "G_{p_2} part of T" and "G_{p_3} part of T", respectively. In the assumptions below, let $G_{p_1 p_2}$ and $G_{p_1 p_3}$ be the subgroups of order $p_1 p_2$ and $p_1 p_3$ in G, respectively. Similarly, an element in $G_{p_1 p_2}$ can be written as the product of an element of G_{p_1} and an element of G_{p_2}, and an element in $G_{p_1 p_3}$ can be written as the product of an element of G_{p_1} and an element of G_{p_3}.

Assumption 1 (Subgroup decision problem for 3 primes). *[12] Given a group generator $\mathcal{G}$, define the following distribution: $\mathbb{G} = (N = p_1 p_2 p_3, G, G_T, e) \xleftarrow{R} \mathcal{G}$, $g \xleftarrow{R} G_{p_1}$, $X_3 \xleftarrow{R} G_{p_3}$, $D = (\mathbb{G}, g, X_3)$, $T_1 \xleftarrow{R} G_{p_1 p_2}$, $T_2 \xleftarrow{R} G_{p_1}$. The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 1 is:*

$$Adv1_{\mathcal{G},\mathcal{A}}(\lambda) := |\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]|.$$

Definition 4. *$\mathcal{G}$ satisfies Assumption 1 if $Adv1_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any polynomial time algorithm $\mathcal{A}$.*

Assumption 2. *[12] Given $\mathcal{G}$, define the following distribution: $\mathbb{G} = (N = p_1p_2p_3, G, G_T, e) \xleftarrow{R} \mathcal{G}$, $g, X_1 \xleftarrow{R} G_{p_1}$, $X_2, Y_2 \xleftarrow{R} G_{p_2}$, $X_3, Y_3 \xleftarrow{R} G_{p_3}$, $D = (\mathbb{G}, g, X_1X_2, X_3, Y_2Y_3), T_1 \xleftarrow{R} G$, $T_2 \xleftarrow{R} G_{p_1p_3}$. The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 2 is:*

$$Adv2_{\mathcal{G},\mathcal{A}}(\lambda) := |\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]|.$$

Definition 5. *$\mathcal{G}$ satisfies Assumption 2 if $Adv2_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any polynomial time algorithm $\mathcal{A}$.*

Assumption 3. *[12] Given $\mathcal{G}$, define the following distribution: $\mathbb{G} = (N = p_1p_2p_3, G, G_T, e) \xleftarrow{R} \mathcal{G}$, $\alpha, s \xleftarrow{R} \mathbb{Z}_N$, $g \xleftarrow{R} G_{p_1}$, $X_2, Y_2, Z_2 \xleftarrow{R} G_{p_2}$, $X_3 \xleftarrow{R} G_{p_3}$, $D = (\mathbb{G}, g, g^\alpha X_2, X_3, g^s Y_2, Z_2)$, $T_1 = e(g, g)^{\alpha s}$, $T_2 \xleftarrow{R} G_T$. The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 3 is:*

$$Adv3_{\mathcal{G},\mathcal{A}}(\lambda) := |\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]|.$$

Definition 6. *$\mathcal{G}$ satisfies Assumption 3 if $Adv3_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any polynomial time algorithm $\mathcal{A}$.*

4 Our Multi-authority CP-ABE

4.1 Outline

Before describing our construction of MA-CP-ABE, we briefly review the CP-ABE scheme of [11] below, and then outline the ideas behind our construction.

$\mathsf{Setup}(\lambda, U) \rightarrow (\mathsf{PK}, \mathsf{MSK})$. Let G be a bilinear group of order $N = p_1p_2p_3$, and G_{p_i} be the subgroup of order p_i in G. Randomly choose $\alpha, a \in \mathbb{Z}_N$ and $g \in G_{p_1}$. For each attribute $att \in U$, randomly choose $s_{att} \in \mathbb{Z}_N$. Let X_3 be a generator of G_{p_3}. The public parameters are $\mathsf{PK} = (N, g, g^a, e(g, g)^\alpha, T_{att} = g^{s_{att}}\ \forall att \in U)$, and the master secret key is $\mathsf{MSK} = (\alpha, X_3)$.

$\mathsf{KeyGen}(\mathsf{MSK}, S, \mathsf{PK}) \rightarrow \mathsf{SK}$. The algorithm randomly chooses $r \in \mathbb{Z}_N$, $R_0, R_0' \in G_{p_3}$, and for each $att \in S$ it randomly picks $R_{att} \in G_{p_3}$. The secret key is: $K = g^\alpha g^{ar} R_0$, $L = g^r R_0'$, $K_{att} = T_{att}^r R_{att}\ \forall att \in S$.

$\mathsf{Encrypt}((A, \rho), \mathsf{PK}, M) \rightarrow CT$. A is an $l \times n$ matrix and ρ maps each row A_x of A to an attribute $\rho(x)$. The algorithm chooses a random vector $\boldsymbol{v} = (s, v_2, \ldots, v_n) \in \mathbb{Z}_N^n$, and for each row A_x of A, it randomly picks $r_x \in \mathbb{Z}_N$. The ciphertext is: $\langle C = M \cdot e(g, g)^{\alpha s}$, $C' = g^s$, $C_x = g^{aA_x \cdot \boldsymbol{v}} T_{\rho(x)}^{-r_x}$, $C_x' = g^{r_x}\ \ \forall x \in \{1, 2, \ldots, l\}\rangle$ along with (A, ρ).

$\mathsf{Decrypt}(CT, \mathsf{PK}, \mathsf{SK}) \rightarrow M$. The algorithm computes constants $\omega_x \in \mathbb{Z}_N$ such that $\sum_{\rho(x) \in S} \omega_x A_x = (1, 0, \ldots, 0)$, then

$$e(C', K) / \prod_{\rho(x) \in S} \left(e(C_x, L) \cdot e(C_x', K_{\rho(x)})\right)^{\omega_x} = e(g, g)^{\alpha s}.$$

Then M can be recovered as $C/e(g, g)^{\alpha s}$.

Now we outline the ideas used in our construction of MA-CP-ABE. We start with building a one-CA-multi-AA system.

One Central Authority and Multiple Attribute Authorities. In the KeyGen algorithm of the underlying CP-ABE system, K and L are uncorrelated to any attributes, and $\forall att \in S$

$$\begin{aligned} K_{att} &= T_{att}^{r} R_{att} \\ &= (g^{s_{att}})^{r} R_{att} = (g^{r})^{s_{att}} R_{att} = (L R_0'^{-1})^{s_{att}} R_{att} = L^{s_{att}} R_0'^{-s_{att}} R_{att} \\ &= L^{s_{att}} R'_{att} \end{aligned}$$

i.e., K_{att} can be computed from L and s_{att} without knowing the value of r. We can get a one-CA-multi-AA system as shown in Fig. 2, where different attribute authorities manage different domains of attributes and a central authority holds the master secret key and generates K and L for users.

While the L is submitted to AAs by the users, the malicious users may launch a collusion attack by submitting the same L. e.g., $L_{Bob} = L_{Tom}$ will allow them put their attribute keys together to decrypt some ciphertexts that they are not authorized to. To defend against this attack, the AAs should verify whether the L is really issued by the CA to the corresponding user. A signature scheme, which is existentially unforgeable under adaptive chosen message attacks (UF-CMA) [8] can be introduced, that is, an adversary cannot generate a valid signature for a new message. Müller, Katzenbeisser, and Eckert [15,16] proposed a similar construction based on Waters' CP-ABE [21]. Their system is selectively secure with non-adaptive key query and the collusion attack above is not considered.

Using signature scheme in multi-authority CP-ABE system was also discussed by Lewko and Waters [13], in a way that, the AAs certify users' identities and their attributes, and a CA issues attribute keys to users according to their identity-attribute certificates. As they mentioned, the CA is demanding as all attribute keys are issued by the CA. Also, the CA will know all the attributes of each user.

Multiple Central Authorities and Multiple Attribute Authorities. In the above one-CA-multi-AA system, the CA can decrypt all ciphertexts because it holds the master secret key α. We use a (D, D) threshold policy to distribute the master secret key to D central authorities to get a multi-CA-multi-AA system as shown in Figure 3. Each central authority (CA_d) publishes a $e(g,g)^{\alpha_d}$ and holds the α_d secretly. The encryptor masks M by $\prod e(g,g)^{\alpha_d s}$. A user gid with attribute set S_{gid} will visit each CA_d to get $(K_{gid,d}, L_{gid,d})$, and then submit $\{L_{gid,d} \mid d \in \mathbb{D}\}$ to related $\{AA_k\}$ where $S_{gid} \cap U_k \neq \emptyset$. For any attribute $att \in S_{gid} \cap U_k$, AA_k will generate $K_{att,gid,d}$ from $L_{gid,d}$ for each $d \in \mathbb{D}$, and issue $\{K_{att,gid,1}, \ldots, K_{att,gid,D}\}$ to gid.

However, this is insecure when some CAs are corrupted, e.g., two malicious users, Bob and Tom, corrupt CA_1. Assume $a_1 \in S_{Tom}$, $a_2 \notin S_{Tom}$, $a_2 \in S_{Bob}$. Normally Tom should not get $K_{a_2,Tom,D}$ to reconstruct $e(g,g)^{\alpha_D s}$ for a ciphertext associated with policy (a_1 **AND** a_2). But Bob can make $L_{Bob,1} = L_{Tom,D}$ because he controls CA_1, then submits $(a_2, L_{Bob,1})$ to AA_K, and AA_K will use $L_{Bob,1}$(actually, $L_{Tom,D}$) to generate $K_{a_2,Bob,1}$ for "Bob", which is actually $K_{a_2,Tom,D}$ for Tom.

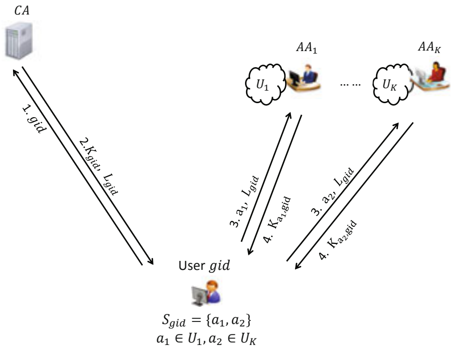


Fig. 2. One-CA-Multi-AA

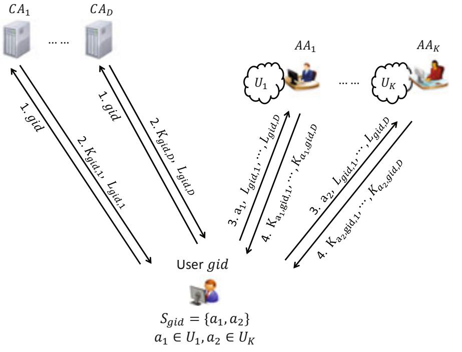

CA_1
CA_D
AA_1
AA_K
U_1
U_K
1. gid
2. $K_{gid,1}$, $L_{gid,1}$
1. gid
2. $K_{gid,D}$, $L_{gid,D}$
3. a_1, $L_{gid,1}, \cdots, L_{gid,D}$
4. $K_{a_1,gid,1}, \cdots, K_{a_1,gid,D}$
3. a_2, $L_{gid,1}, \cdots, L_{gid,D}$
4. $K_{a_2,gid,1}, \cdots, K_{a_2,gid,D}$
User gid
$S_{gid} = \{a_1, a_2\}$
$a_1 \in U_1, a_2 \in U_K$

Fig. 3. Multi-CA-Multi-AA

Our solution is to have AA_k check if CA_d has *honestly* generated $L_{gid,d}$. In our construction, when CA_d generates a $L_{gid,d} = g^r R'$, it must generate $\Gamma_{gid,d,k} = V_{k,d}^r R$ for each $k \in \mathbb{K}$ for showing the knowledge of r. This idea is also borrowed from the underlying CP-ABE scheme. In particular, given ($L = g^r R'_0$, $K_{att} = T_{att}^r R_{att}$ $\forall att \in S$) and $T_{att'}$ where $att' \notin S$, an attacker cannot construct $K_{att'} = T_{att'}^r R_{att'}$.

4.2 Construction

$\mathsf{GlobalSetup}(\lambda) \rightarrow (\mathsf{GPK})$. Let G be a bilinear group of order $N = p_1 p_2 p_3$ (3 distinct primes), and G_{p_i} be the subgroup of order p_i in G. The algorithm randomly chooses $g, h \in G_{p_1}$. Let X_3 be a generator of G_{p_3}.
The global public parameter is published as $\mathsf{GPK} = (N, g, h, X_3, \Sigma_{sign})$, where $\Sigma_{sign} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is the description of an UF-CMA secure signature scheme.

$\mathsf{CASetup}(\mathsf{GPK}, d) \rightarrow (\mathsf{CPK}_d, \mathsf{CAPK}_d, \mathsf{CMSK}_d)$. CA_d runs the KeyGen algorithm of Σ_{sign} to generate sign key pair $(\mathsf{SignKey}_d, \mathsf{VerifyKey}_d)$, and chooses a random exponent $\alpha_d \in \mathbb{Z}_N$.
CA_d publishes its public parameter $\mathsf{CPK}_d = e(g,g)^{\alpha_d}, \mathsf{CAPK}_d = \mathsf{VerifyKey}_d$.
CA_d sets its master secret key $\mathsf{CMSK}_d = (\alpha_d, \mathsf{SignKey}_d)$.

$\mathsf{AASetup}(\mathsf{GPK}, k, U_k) \rightarrow (\mathsf{APK}_k, \mathsf{ACPK}_k, \mathsf{AMSK}_k)$. For each $att \in U_k$, AA_k randomly chooses $s_{att} \in \mathbb{Z}_N$ and sets $T_{att} = g^{s_{att}}$. For each $d \in \mathbb{D}$, AA_k randomly chooses $v_{k,d} \in \mathbb{Z}_N$ and sets $V_{k,d} = g^{v_{k,d}}$.
AA_k publishes its public parameter $\mathsf{APK}_k = \{T_{att} | att \in U_k\}, \mathsf{ACPK}_k = \{V_{k,d} | d \in \mathbb{D}\}$.
AA_k sets its master secret key $\mathsf{AMSK}_k = (\{s_{att} | att \in U_k\}, \{v_{k,d} | d \in \mathbb{D}\})$.

$\mathsf{Encrypt}(M, \mathbb{A} = (A, \rho), \mathsf{GPK}, \{\mathsf{CPK}_d | d \in \mathbb{D}\}, \{\mathsf{APK}_k\}) \rightarrow CT$. M is the message to be encrypted, $\mathbb{A}$ is the access policy which is expressed by an LSSS matrix (A, ρ), where A is an $l \times n$ matrix and ρ maps each row A_x of A to an attribute $\rho(x)$. Here it is required that ρ will not map two different rows to a same attribute.

The algorithm chooses a random vector $\boldsymbol{v} = (s, v_2, \ldots, v_n) \in \mathbb{Z}_N^n$, and for each $x \in \{1, 2, \ldots l\}$, it randomly picks $r_x \in \mathbb{Z}_N$. Let $A_x \cdot \boldsymbol{v}$ be the inner product of the x^{th} row of A and the vector $\boldsymbol{v}$. The ciphertext is

$$C = M \cdot \prod_{d=1}^{D} e(g, g)^{\alpha_d \cdot s}, \ C' = g^s,$$
$$\{C_x = h^{A_x \cdot \boldsymbol{v}} T_{\rho(x)}^{-r_x}, C'_x = g^{r_x} \mid x \in \{1, 2, \ldots l\}\}$$

along with the access policy $\mathbb{A} = (A, \rho)$.

$\mathsf{CKeyGen}(gid, \mathsf{GPK}, \{V_{k,d} | k \in \mathbb{K}\}, \mathsf{CMSK}_d) \to (\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d})$. When a user submits his gid to CA_d to request the user-central-key, CA_d randomly chooses $r_{gid,d} \in \mathbb{Z}_N$ and $R_{gid,d}, R'_{gid,d} \in G_{p_3}$, then sets

$$\mathsf{ucsk}_{gid,d} = g^{\alpha_d} h^{r_{gid,d}} R_{gid,d}, \quad L_{gid,d} = g^{r_{gid,d}} R'_{gid,d}.$$

For $k = 1$ to K, CA_d randomly picks $R_{gid,d,k} \in G_{p_3}$ and computes

$$\Gamma_{gid,d,k} = V_{k,d}^{r_{gid,d}} R_{gid,d,k}.$$

CA_d computes $\sigma_{gid,d} = \mathsf{Sign}(\mathsf{SignKey}_d, gid||d||L_{gid,d}||\Gamma_{gid,d,1}|| \ldots ||\Gamma_{gid,d,K})$. Let $\mathsf{ucpk}_{gid,d} = (gid, d, L_{gid,d}, \{\Gamma_{gid,d,k} \mid k \in \mathbb{K}\}, \sigma_{gid,d})$.

$\mathsf{AKeyGen}(att, \{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \mathsf{GPK}, \{\mathsf{VerifyKey}_d | d \in \mathbb{D}\}, \mathsf{AMSK}_k) \to \mathsf{uask}_{att,gid}$ or $\perp$. When a user submits his $\{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}$ to AA_k to request the user-attribute-key for attribute $att \in U_k$,

1. For $d = 1$ to D, AA_k parses $\mathsf{ucpk}_{gid,d}$ into $(gid, d, L_{gid,d}, \{\Gamma_{gid,d,k} | k \in \mathbb{K}\}, \sigma_{gid,d})$ and checks whether

$$valid \leftarrow \mathsf{Verify}(\mathsf{VerifyKey}_d, gid||d||L_{gid,d}||\Gamma_{gid,d,1}|| \ldots ||\Gamma_{gid,d,K}, \sigma_{gid,d}) \tag{1}$$

$$e(g, \Gamma_{gid,d,k}) = e(V_{k,d}, L_{gid,d}) \neq 1. \tag{2}$$

If there is any failure, AA_k outputs $\perp$ to user to imply the submitted $\{\mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}$ are invalid.

2. For $d = 1$ to D, AA_k randomly picks $R'_{att,gid,d} \in G_{p_3}$, and sets

$$\mathsf{uask}_{att,gid,d} = (\Gamma_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d}. \tag{3}$$

Note that

$$\begin{aligned} \mathsf{uask}_{att,gid,d} &= (\Gamma_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d} \\ &= (V_{k,d}^{r_{gid,d}} R_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d} \\ &= (g^{v_{k,d} \cdot r_{gid,d}} R_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d} \\ &= T_{att}^{r_{gid,d}} (R_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d} \end{aligned}$$

As $(R_{gid,d,k})^{s_{att}/v_{k,d}} R'_{att,gid,d}$ is in G_{p_3} and $R'_{att,gid,d}$ is randomly chosen, we can write

$$\mathsf{uask}_{att,gid,d} = T_{att}^{r_{gid,d}} R_{att,gid,d}. \tag{4}$$

Without knowing the value of $r_{gid,d}$, by running (3), AA_k can compute the value as (4).

3. AA_k outputs user-attribute-key $\mathsf{uask}_{att,gid}$ to user where

$$\begin{aligned}\mathsf{uask}_{att,gid} &= \prod_{d=1}^{D} \mathsf{uask}_{att,gid,d} = \prod_{d=1}^{D} T_{att}^{r_{gid,d}} R_{att,gid,d} \\ &= T_{att}^{\sum_{d=1}^{D} r_{gid,d}} \prod_{d=1}^{D} R_{att,gid,d} \\ &= T_{att}^{\sum_{d=1}^{D} r_{gid,d}} R_{att,gid}\end{aligned} \tag{5}$$

$\mathsf{Decrypt}(CT, \mathsf{GPK}, \{\mathsf{APK}_k\}, \mathsf{DK}_{gid}) \rightarrow M$. The ciphertext CT is parsed into $\langle C, C', \{C_x, C'_x | x \in \{1, 2, \ldots, l\}\}, \mathbb{A} = (A, \rho)\rangle$, and the decryption-key DK_{gid} is parsed into $(\{\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \{\mathsf{uask}_{att,gid} | att \in S_{gid}\})$.
The algorithm computes

- $\mathsf{ucsk}_{gid} = \prod_{d=1}^{D} \mathsf{ucsk}_{gid,d} = g^{\sum_{d=1}^{D} \alpha_d} h^{\sum_{d=1}^{D} r_{gid,d}} \prod_{d=1}^{D} R_{gid,d} = g^{\alpha} h^{r_{gid}} R_{gid}$,

  with $\alpha = \sum_{d=1}^{D} \alpha_d$, $r_{gid} = \sum_{d=1}^{D} r_{gid,d}$ and $R_{gid} = \prod_{d=1}^{D} R_{gid,d}$.
- $L_{gid} = \prod_{d=1}^{D} L_{gid,d} = g^{\sum_{d=1}^{D} r_{gid,d}} \prod_{d=1}^{D} R'_{gid,d} = g^{r_{gid}} R'_{gid}$,

  with $R'_{gid} = \prod_{d=1}^{D} R'_{gid,d}$.

Note that $\forall att \in S_{gid}$, $\mathsf{uask}_{att,gid} = T_{att}^{\sum_{d=1}^{D} r_{gid,d}} R_{att,gid} = T_{att}^{r_{gid}} R_{att,gid}$.
If S_{gid} satisfies the access policy (A, ρ), the algorithm computes constants $\omega_x \in \mathbb{Z}_N$ such that $\sum_{\rho(x) \in S_{gid}} \omega_x A_x = (1, 0, \ldots, 0)$. Then it computes

$$e(C', \mathsf{ucsk}_{gid}) / \prod_{\rho(x) \in S_{gid}} \left(e(C_x, L_{gid}) \cdot e(C'_x, \mathsf{uask}_{\rho(x),gid})\right)^{\omega_x} = e(g, g)^{\alpha s}.$$

While $C = M \cdot \prod_{d=1}^{D} e(g, g)^{\alpha_d \cdot s} = M \cdot e(g, g)^{s \sum_{d=1}^{D} \alpha_d} = M \cdot e(g, g)^{s\alpha}$, M can be recovered from $C / e(g, g)^{\alpha s}$.

In the above system, it is required that an attribute appears at most once in an LSSS matrix (A, ρ). This restriction is crucial to the security proof. As in [11], we call such a system as a One-Use system, and we can use the encoding technique in [11] to extend our system to a Multi-Use system. In Appendix A, we analyze the security of the system above.

5 Extensions

5.1 Large Universe Construction

In the construction in Sec.4.2, the size of the public parameters of AA_k is linear in $|U_k|$. We can modify our scheme to get a large universe construction by using a technique similar to that in [10]. For each AA_k, let n_k denote the maximum size of the set $S_{gid} \cap U_k$ for any user gid. In addition, we let $H : \{0,1\}^* \rightarrow \mathbb{Z}_N$ be a collision-resistant hash function so that we can use arbitrary strings as attributes, and let $\psi : U \mapsto \mathbb{K}$ be a function that maps an attribute to the index of the corresponding attribute authority. The $\mathsf{AASetup}(\mathsf{GPK}, k, U_k)$ algorithm is modified to

$\mathsf{AASetup}(\mathsf{GPK}, k, n_k)$

AA_k chooses $n_k + 1$ random exponents $a_{k,0}, a_{k,1}, \ldots, a_{k,n_k} \in \mathbb{Z}_N$ and sets $F_{k,i} = g^{a_{k,i}}(i = 0, 1, \ldots, n_k)$.

For each $d \in \mathbb{D}$, AA_k randomly chooses $v_{k,d} \in \mathbb{Z}_N$ and sets $V_{k,d} = g^{v_{k,d}}$.

AA_k publishes its public parameter $\mathsf{APK}_k = \{F_{k,i} \mid i = 0, 1, \ldots, n_k\}, \mathsf{ACPK}_k = \{V_{k,d} \mid d \in \mathbb{D}\}$.

AA_k sets its master secret key $\mathsf{AMSK}_k = (\{a_{k,i} \mid i = 0, 1, \ldots, n_k\}, \{v_{k,d} \mid d \in \mathbb{D}\})$.

Let $q_k(x) = \sum_{i=0}^{n_k} a_{k,i}x^i$, $F_k(x) = g^{q_k(x)} = \prod_{i=0}^{n_k} (F_{k,i})^{x^i}$, and for any $att \in \{0,1\}^*$, $s_{att} = q_{\psi(att)}(H(att))$, $T_{att} = F_{\psi(att)}(H(att))$. Then $T_{att} = g^{s_{att}}$. Note that for any $att \in \{0,1\}^*$, the encryptor can compute T_{att} from public parameters, and the corresponding AA_k can compute s_{att} from its master secret key.

5.2 Improving Performance and Robustness

In the construction in Sec.4.2, the trust on each central authority is minimized so that the central authorities could not decrypt any ciphertext unless all central authorities are involved. However, the robustness of the system is limited. Each central authority must remain active because a user must obtain his user-central-keys from each central authority. A threshold policy will be an effective way to balance the trust on each central authority and the robustness of the system.

The Setup phase is executed by a trusted party. The trusted party chooses a random $\alpha \in \mathbb{Z}_N$ and determines a threshold policy (D, Δ) where $1 < D \leq \Delta$, then generates Δ shares $\alpha_1, \alpha_2, \ldots, \alpha_\Delta$. α_d is securely distributed to CA_d to be its master secret key. The trusted party publishes $e(g,g)^\alpha$ and (D, Δ) to global public parameters, and then discards α.

In such a system, the encryptor will mask plaintext M with $e(g,g)^{\alpha s}$. A user needs to visit any D central authorities to obtain his user-central-keys so that he can get his decryption-key.

Only when D central authorities are involved, they can decrypt a ciphetext. The system will work until more than $\Delta - D$ central authorities fail. When $D = \Delta$, it is the system proposed in Sec.4.2.

The detail of such a system will be presented in the full version.

6 Comparison

In Table 2, we compare the single-authority CP-ABE in [11], the multi-authority CP-ABE in [13] and our MA-CP-ABE system. In the table, l is the number of

Table 2. Comparison

	CP-ABE Scheme in [11]	MA-CP-ABE Scheme in [13]	Our MA-CP-ABE
Standard Model	$\surd$	$\times$	$\surd$
Multi-Authority	$\times$	$\surd$	$\surd$
Prevent Decryption by Individual Authorities	$\times$	Partially	$\surd$
Size of Ciphertext	$2l+2$	$3l+1$	$2l+2$
Size of SK	$\|S\|+2$	$\|S\|$	$\|S\|+D(K+2)$
Pairing computation of decryption	$2\|I\|+1$	$2\|I\|$	$2\|I\|+1$
Size of PK	$\|U\|+3$	$2\|U\|$	$\|U\|+3+D$
Large Universe Construction	$\surd$	$\times$	$\surd$

rows of the LSSS matrix (A,ρ), S is the attribute set of the secret key, $|I|$ is the number of rows of (A,ρ) that are used in the decryption, U is the attribute universe, D is the number of CAs, and K is the number of AAs. All the three systems are fully secure, and realize any LSSS access structure.

In [11], the authority can decrypt all ciphertexts; in [13], no authority can decrypt all ciphertexts, but each authority can independently decrypt some ciphertexts; in our MA-CP-ABE scheme, no authority can independently decrypt any ciphertext. While the user and the encryptor will not use the public parameters CAPK_d and ACPK_k, we do not count them in the size of PK. The total size of these keys is $D + D \cdot K$. The size of PK of our system shown in the table is that of the construction in Sec.4.2. For the large universe construction in Sec.5.1, the size of PK is $\sum_{k=1}^{K}(n_k+1)+3+D$ which is not related to the size of U. It is worth noticing that introducing multiple CAs is to prevent some CAs from decrypting ciphertexts. Hence D could be a small value.

7 Conclusion

In this work, we constructed a multi-authority CP-ABE scheme where different domains of attributes are managed by different attribute authorities and no authority can independently decrypt any ciphertext. The proposed system is proved fully secure in the standard model, realizes any monotone access structure, and has almost same efficiency as the underlying CP-ABE scheme. In addition, the proposed system can be extended to support large attribute universe.

References

1. Beimel, A.: Secure Schemes for Secret Sharing and Key Distribution. Ph.D. thesis, Israel Institute of Technology, Technion, Haifa, Israel (1996)
2. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society, Los Alamitos (2007)

3. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
4. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
5. Chase, M.: Multi-authority attribute based encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007)
6. Chase, M., Chow, S.S.M.: Improving privacy and security in multi-authority attribute-based encryption. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (eds.) ACM Conference on Computer and Communications Security, pp. 121–130. ACM, New York (2009)
7. Cheung, L., Newport, C.C.: Provably secure ciphertext policy abe. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 456–465. ACM, New York (2007)
8. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
9. Goyal, V., Jain, A., Pandey, O., Sahai, A.: Bounded Ciphertext Policy Attribute Based Encryption. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 579–591. Springer, Heidelberg (2008)
10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 89–98. ACM, New York (2006)
11. Lewko, A.B., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (Hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
12. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010)
13. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 568–588. Springer, Heidelberg (2011)
14. Lin, H., Cao, Z., Liang, X., Shao, J.: Secure threshold multi authority attribute based encryption without a central authority. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 426–436. Springer, Heidelberg (2008)
15. Müller, S., Katzenbeisser, S., Eckert, C.: Distributed attribute-based encryption. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 20–36. Springer, Heidelberg (2009)
16. Müller, S., Katzenbeisser, S., Eckert, C.: On multi-authority ciphetext-policy attribute-based encryption. Bulletin of the Korean Mathematical Society 46(4), 803–819 (2009)
17. Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 191–208. Springer, Heidelberg (2010)

18. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 195–203. ACM, New York (2007)
19. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
20. 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)
21. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 53–70. Springer, Heidelberg (2011)

A Security Analysis

Let Π denote the main construction, we modify Π to Π' as follows.

- In the AKeyGen algorithm, it outputs $\mathsf{uask}_{att,gid} = \{\mathsf{uask}_{att,gid,d} | d \in \mathbb{D}\}$ rather than $\mathsf{uask}_{att,gid} = \prod_{d=1}^{D} \mathsf{uask}_{att,gid,d}$. i.e., gid's decryption-key is

$$\begin{aligned}\mathsf{DK}_{gid} =& (\{\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \{\mathsf{uask}_{att,gid} \mid att \in S_{gid}\}) \\ =& (\{\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \{\{\mathsf{uask}_{att,gid,d} | d \in \mathbb{D}\} | att \in S_{gid}\}) \\ =& (\{\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d} | d \in \mathbb{D}\}, \{\{\mathsf{uask}_{att,gid,d} | att \in S_{gid}\} | d \in \mathbb{D}\}) \\ =& \{(\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d}, \{\mathsf{uask}_{att,gid,d} | att \in S_{gid}\}) | d \in \mathbb{D}\} \\ =& \{\mathsf{usk}_{gid,d} | d \in \mathbb{D}\}\end{aligned}$$

  where $\mathsf{usk}_{gid,d} = (\mathsf{ucsk}_{gid,d}, \mathsf{ucpk}_{gid,d}, \{\mathsf{uask}_{att,gid,d} | att \in S_{gid}\})$ is called gid's ***user-key*** related to d.
- In the Decrypt algorithm,
  1. For $d = 1$ to D, the algorithm uses $\mathsf{usk}_{gid,d}$ to reconstruct $e(g,g)^{\alpha_d s}$:

$$e(C', \mathsf{ucsk}_{gid,d}) / \prod_{\rho(x) \in S_{gid}} \left(e(C_x, L_{gid,d}) \cdot e(C'_x, \mathsf{uask}_{\rho(x),gid,d})\right)^{\omega_x} = e(g,g)^{\alpha_d s}. \tag{6}$$

  2. The algorithm recovers M by

$$M = C / \prod_{d=1}^{D} e(g,g)^{\alpha_d s}. \tag{7}$$

Note that the user and the attacker will get more information in Π', the security of Π' will imply the security of Π. We show the security of Π' in the following.

In the security model, CA_{d^*} is the only uncorrupted central authority and no $S_{gid} \cup (\bigcup_{k_c \in \mathbb{K}_c} U_{k_c})$ can satisfy the challenge access policy. It means that the adversary could not request keys to form a usk_{gid,d^*} to reconstruct $e(g,g)^{\alpha_{d^*} s}$. In our proof, the challenger will respond the adversary as in real attack for all key queries related to $d \neq d^*$. On the key queries related to d^*, we use the proof technique of [11] to provide the answers.

Before we give our proof, we need to define two additional structures: semi-functional ciphertexts and keys. We choose random values $z_{att} \in \mathbb{Z}_N$ associated to the attributes.

Semi-functional Ciphertext. A semi-functional ciphertext is formed as follows. Let g_2 denote a generator of G_{p_2} and c a random exponent modulo N. Besides the random vector $\boldsymbol{v} = (s, v_2, \ldots, v_n)$ and the random values $\{r_x | x \in \{1, 2, \ldots, l\}\}$, we also choose a random vector $\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{Z}_N^n$ and random values $\{\gamma_x \in \mathbb{Z}_N | x \in \{1, 2, \ldots, l\}\}$. Then:

$$C' = g^s g_2^c, \{C_x = h^{A_x \cdot \boldsymbol{v}} T_{\rho(x)}^{-r_x} g_2^{A_x \boldsymbol{u} + \gamma_x z_{\rho(x)}}, C'_x = g^{r_x} g_2^{-\gamma_x} \mid x \in \{1, 2, \ldots, l\}\}.$$

Semi-functional Key. For a gid, a semi-functional user-key usk_{gid,d^*} will take on one of two forms. Exponents $r_{gid,d^*}, \delta, b \in \mathbb{Z}_N$, $\{w_{k,d^*} \in \mathbb{Z}_N | k \in \mathbb{K}\}$, and elements $R_{gid,d^*}, R'_{gid,d^*} \in G_{p_3}, \{R_{att,gid,d^*} \in G_{p_3} | att \in S_{gid}\}, \{R_{gid,d^*,k} \in G_{p_3} | k \in \mathbb{K}\}$ are chosen randomly.

- Type 1:
  The user-central-key $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$ is formed as

$$\begin{aligned}
&\mathsf{ucsk}_{gid,d^*} = g^{\alpha_{d^*}} h^{r_{gid,d^*}} R_{gid,d^*} g_2^{\delta}, \quad L_{gid,d^*} = g^{r_{gid,d^*}} R'_{gid,d^*} g_2^{b}, \\
&\varGamma_{gid,d^*,k} = V_{k,d^*}^{r_{gid,d^*}} R_{gid,d^*,k} g_2^{b w_{k,d^*}} \, (k = 1, 2, \ldots K), \\
&\sigma_{gid,d^*} = \mathsf{Sign}(\mathsf{SignKey}_{d^*}, gid || d^* || L_{gid,d^*} || \varGamma_{gid,d^*,1} || \ldots || \varGamma_{gid,d^*,K}), \\
&\mathsf{ucpk}_{gid,d^*} = (gid, d^*, L_{gid,d^*}, \{\varGamma_{gid,d^*,k} | k \in \mathbb{K}\}, \sigma_{gid,d^*}).
\end{aligned}$$

  $\forall att \in S_{gid}$, the derived $\mathsf{uask}_{att,gid,d^*}$ is formed as

$$\mathsf{uask}_{att,gid,d^*} = T_{att}^{r_{gid,d^*}} R_{att,gid,d^*} g_2^{b z_{att}}.$$

- Type 2:
  The user-central-key $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$ is formed as

$$\begin{aligned}
&\mathsf{ucsk}_{gid,d^*} = g^{\alpha_{d^*}} h^{r_{gid,d^*}} R_{gid,d^*} g_2^{\delta}, \quad L_{gid,d^*} = g^{r_{gid,d^*}} R'_{gid,d^*}, \\
&\varGamma_{gid,d^*,k} = V_{k,d^*}^{r_{gid,d^*}} R_{gid,d^*,k} (k = 1, 2, \ldots K), \\
&\sigma_{gid,d^*} = \mathsf{Sign}(\mathsf{SignKey}_{d^*}, gid || d^* || L_{gid,d^*} || \varGamma_{gid,d^*,1} || \ldots || \varGamma_{gid,d^*,K}), \\
&\mathsf{ucpk}_{gid,d^*} = (gid, d^*, L_{gid,d^*}, \{\varGamma_{gid,d^*,k} | k \in \mathbb{K}\}, \sigma_{gid,d^*}).
\end{aligned}$$

  $\forall att \in S_{gid}$, the derived $\mathsf{uask}_{att,gid,d^*}$ is formed as

$$\mathsf{uask}_{att,gid,d^*} = T_{att}^{r_{gid,d^*}} R_{att,gid,d^*}.$$

Note that both the semi-functional user-keys of type 1 and type 2 satisfy (1) and (2), and that type 2 is a special case of type 1 with $b = 0$.

When a normal usk_{gid,d^*} and a semi-functional ciphertext, or a semi-functional usk_{gid,d^*} and a normal ciphertext, are used in computation (6), $e(g, g)^{\alpha_{d^*} s}$ is got, and this value could be used in the computation (7). When a semi-functional usk_{gid,d^*} and a semi-functional ciphertext are used in computation (6), $e(g,g)^{\alpha_{d^*}s} \cdot e(g_2,g_2)^{c\delta - bu_1}$ is got. The additional term $e(g_2,g_2)^{c\delta - bu_1}$ will hinder the computation (7). We call a semi-functional user-key of type 1 *nominally semi-functional* if $c\delta - bu_1 = 0$.

The security of Π' relies on Assumptions 1, 2, 3. We use a hybrid argument over a sequence of games. The first game $\mathbf{Game}_{\mathsf{Real}}$ is the real security game. In the final game $\mathbf{Game}_{\mathsf{Final}}$, all user-keys related d^*, $\{\mathsf{usk}_{gid,d^*}\}$, are semi-functional of type 2 and the ciphertext is a semi-functional encryption of a random message, independent of the two messages provided by $\mathcal{A}$.

$\mathbf{Game}_{\text{Real}}$. The challenge ciphertext is normal. All CKQs are answered with normal user-central-key. All AKQs are answered with user-attribute-key generated by running the normal $\mathsf{AKeyGen}$ algorithm.

$\mathbf{Game}_0$. The challenge ciphertext is semi-functional. All CKQs are answered with normal user-central-key. All AKQs are answered with user-attribute-key generated by running the normal $\mathsf{AKeyGen}$ algorithm.

Let q denote the number of CKQ made by $\mathcal{A}$. For j from 1 to q, we consider the following games:

$\mathbf{Game}_{j,1}$. In this game, the challenge ciphertext is semi-functional. The first $j-1$ CKQs are answered with semi-functional user-central-key of type 2; the j^{th} CKQ is answered with semi-functional user-central-key of type 1; and the remaining CKQs are answered with normal user-central-key. All AKQs are answered with user-attribute-key generated by running the normal $\mathsf{AKeyGen}$ algorithm.

$\mathbf{Game}_{j,2}$. In this game, the challenge ciphertext is semi-functional. The first $j-1$ CKQs are answered with semi-functional user-central-key of type 2; the j^{th} CKQ is answered with semi-functional user-central-key of type 2; and the remaining CKQs are answered with normal user-central-key. All AKQs are answered with user-attribute-key generated by running the normal $\mathsf{AKeyGen}$ algorithm.

$\mathbf{Game}_{\text{Final}}$. In this game, the challenge ciphertext is a semi-functional encryption of a random message, independent of the two messages provided by the adversary. All CKQs are answered with semi-functional user-central-key of type 2. All AKQs are answered with user-attribute-key generated by running the normal $\mathsf{AKeyGen}$ algorithm.

Note that in all the games, all AKQs are answered with user-attribute-key generated by running normal $\mathsf{AKeyGen}$ algorithm. In the proofs, we will show that the derived $\mathsf{uask}_{att,gid,d^*}$ is decided by the corresponding user-central-key $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$, i.e., if $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$ is semi-functional of type 1 (respectively, type 2), then the derived $\mathsf{uask}_{att,gid,d^*}$ is also semi-functional of type 1 (respectively, type 2). Consequently, usk_{gid,d^*} is decided by the corresponding $(\mathsf{ucsk}_{gid,d^*}, \mathsf{ucpk}_{gid,d^*})$ as well. Note that in $\mathbf{Game}_0$ all user-central-keys related to d^* are normal and in $\mathbf{Game}_{q,2}$ all user-central-keys related to d^* are semi-functional of type 2. It means that in $\mathbf{Game}_0$ all user-keys usk_{gid,d^*}

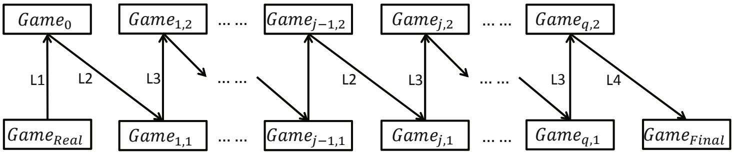


Fig. 4. Indistinguishable games. L1 denotes Lemma 1, and so on.

are normal and in $\mathbf{Game}_{q,2}$ all user-keys usk_{gid,d^*} are semi-functional of type 2. We show these games are indistinguishable in the following four lemmas (see Fig. 4), the proofs of which will appear in the full version.

Lemma 1. *Given a UF-CMA signature scheme Σ_{sign}, suppose there exists a poly-time algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{\mathsf{Real}}Adv_{\mathcal{A}} - \boldsymbol{Game}_0 Adv_{\mathcal{A}} = \epsilon$. We can construct a poly-time algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 1.*

Lemma 2. *Use $\boldsymbol{Game}_{0,2}$ to denote $\boldsymbol{Game}_0$. Given a UF-CMA signature scheme Σ_{sign}, suppose there exists a poly-time algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{j-1,2}Adv_{\mathcal{A}} - \boldsymbol{Game}_{j,1}Adv_{\mathcal{A}} = \epsilon$. We can construct a poly-time algorithm $\mathcal{B}$ with advantage negligibly close to ϵ in breaking Assumption 2.*

Lemma 3. *Given a UF-CMA signature scheme Σ_{sign}, suppose there exists a poly-time algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{j,1}Adv_{\mathcal{A}} - \boldsymbol{Game}_{j,2}Adv_{\mathcal{A}} = \epsilon$. We can construct a poly-time algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 2.*

Lemma 4. *Given a UF-CMA signature scheme Σ_{sign}, suppose there exists a poly-time algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{q,2}Adv_{\mathcal{A}} - \boldsymbol{Game}_{\mathsf{Final}}Adv_{\mathcal{A}} = \epsilon$. We can construct a poly-time algorithm $\mathcal{B}$ with advantage $\frac{\epsilon}{D}$ in breaking Assumption 3.*

Theorem 1. *If the signature scheme Σ_{sign} is UF-CMA secure and Assumptions 1, 2, and 3 hold, then our MA-CP-ABE scheme is secure.*

Proof. If Assumptions 1, 2 and 3 hold, and the signature scheme Σ_{sign} is UF-CMA secure, then we have shown by the previous lemmas that the real security game is indistinguishable from $\mathbf{Game}_{\mathsf{Final}}$, in which the value of β is information-theoretically hidden from the adversary. Hence the adversary can not attain a non-negligible advantage in breaking Π', which implies the adversary can not attain a non-negligible advantage in breaking our MA-CP-ABE scheme Π. □

How to Aggregate the CL Signature Scheme

Dominique Schröder[*]

University of Maryland, USA
schroeder@me.com
www.dominique-schroeder.de

Abstract. We present an aggregate signature scheme whose public key consists of only two group elements. It is therefore the first sequential aggregate signature scheme with short keys in the standard model. Our construction relies on the Camenisch-Lysyanskaya signature scheme (Crypto 2004) and is provably secure under the LRSW assumption. Moreover, we develop a novel aggregation technique that we call *aggregate-extension technique*. The basic idea is to extend the aggregate by a single element and to use this additional space to "store" some information that would be lost due to the compression of the signatures. We believe that this technique might be of independent interest.

1 Introduction

Aggregate signature schemes allow the combination of several signatures into a single element, the aggregate, that has roughly the same size as an ordinary signature. Here, we consider the sequential case where a signer receives an aggregate-so-far, adds its own signature to the aggregate and forwards the aggregate (containing the new signature) to the next signer. The size of the aggregate is independent of the number of signers, i.e., it has the roughly the same size as an ordinary signature scheme. Typical applications for such schemes are sensor networks where communication is prohibitively expensive [2]. Since the transmission range of each sensor is limited, the sensor forwards its data to the next sensor node towards the base station. Moreover, each sensor signs its measurement to prevent attackers from raising a false alarm. One example of such a monitoring network is the Tsunami early warning system that is already in operation in the Indian Ocean [23]. Further applications are the compression of certificate chains [10] and secure routing protocols, such as the Secure Border Gateway Protocol (S-BGP) [6].

Public-Key Size. Efficiency refers to three kinds of costs: computational, storing data, and the cost of communication. In practice, however, computational costs play a minor role due to the rapid growth of computational power over the last decades. On the other hand, the costs of transmitting data and of storing data are essential in practice. Consequently, an entire branch of research in

[*] Supported in part by a DAAD postdoctoral fellowship.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 298–314, 2011.

cryptography focuses on minimizing the size of transmitted data, such as short signatures, e.g., [11,14,13,8,22] and short group signatures, e.g., [9,14].

For aggregate signature schemes the size of the public key is an important measurement. The reason is that in most of the applications the public-keys are transported with the aggregate and bandwidth is expensive. Neven already pointed out that optimizing the transmitted data means reducing the size of all three elements, rather than only considering the size of the aggregate σ [31]. The size of the data transmitted from user to user increases thus linearly in the number of users; as it is difficult to reduce the message size (which is often fixed — consider for example the measurements of the water level of the Tsunami early warning system), and because the aggregate σ is constant, reducing the key size is the only way to reduce the size of the transmitted data. Additionally, Lu et al. mention in their paper that “large keys negates any benefit from reducing the signature size in a certificate chain, since the keys must be included in the certificate” [27]. In this paper, we present the first sequential aggregate signature scheme with short keys in the standard model.

As an example, we consider the size of an aggregate generated by 20 signers; a comparison of the size for the different schemes is given in the last column of Table 1. For the calculation of the values we assume that the size of each element on an elliptic curve has 160 bits and that the messages have 160 bits as well. In the case of RSA group elements we assume that the moduls has 1024 bits. Thus, in the case of BGLS we transmit $(20 + 20 + 1) * 160 = 6560$ bits. Note, that our construction is not as efficient as the LOSSW sequential aggregate signature scheme from a computational point of view[1], but it reduces the key-size — and thus the size of the transmitted data.

Our Approach. We show how to aggregate the Camenisch-Lysyanskaya signature scheme. The public key of their scheme has only two group elements and it does not rely on the random oracle heuristic. We briefly recall the CL signature scheme to explain why aggregation of this scheme is not straightforwardly possible. A signature $\sigma = (a, b, c)$ on a message M consists of three elements $g^r, g^{r\,y}, g^{r\,(x+Mxy)}$, where r is a random element, and the values x and y are stored in the private key. Unfortunately, common aggregation techniques fail in this case. If we try to aggregate the signature by simply multiplying different signatures together, then we end up with different signatures having a different randomness. One solution would be to rely on a global counter such that all signers share the same randomness. It is well known, however, that global counters are always difficult to realize. Instead, we pick up an idea of Lu et al. [27] and let the $(i+1)$th signer “re-use” the randomness r_i of the ith signer. That is, it treats the element $a_i = g^{r_i}$ as its own randomness and computes $b_{i+1} \leftarrow a_i^{y_{i+1}}$ and $c_{i+1} \leftarrow a_i^{x_{i+1}+M_{i+1}x_{i+1}y_{i+1}}$. It is easy to see that the tuple (a_i, b_{i+1}, c_{i+1}) forms a valid CL signature.

[1] Chatterjee, Hankerson, Knapp, and Menezes have recently compared the efficiency of the BGLS aggregate signature scheme with the LOSSW aggregate signature scheme [15]. The comparison shows that evaluating n pairings is not as expensive as one might have expected.

Table 1. Comparison of aggregate signature schemes

Scheme	ROM	KOSK	Size	SK/PK	Signing	Verification	Trans-Data 20 Users
BGLS [10]	YES	NO	1	1/1	1E	nP	6560 bits
LMRS-1 [28]	YES	NO	1	1/1	nE	$2n$E	24,704 bits
LMRS-2 [28]	YES	NO	1	1/1	nM	$4n$M	24,704 bits
Neven [31]	YES	NO	1	1/1	1E + $2\,n$M	$2\,n$M	24,704 bits
LOSSW [27]	NO	YES	2	162/162	2P + $n\ell$M	2P + $n\ell$M	~63KB
Section 3.4	NO	YES	4	**2/2**	nP + $2n$E	nP+nE	9760 bits

We use the following notation: ROM means that the security proof is given in the random oracle model and "KOSK" indicates that the security is proven in the certified-key model, where the adversary has to proof knowledge about the private keys. Sizes of the aggregate or of the keys are given as the number of objects (group elements, ring elements). Note that the size of an RSA group elements is roughly 10 times the size of an elliptic curve element. n denotes the number of participants, P a pairing computation, E a (multi)-exponentiation, M for a multiplication and ℓ for the output length of a collision resistant hash function. These values, except for the last row, are a verbatim copy of [31].

Now, multiplying the tuples (a_i, b_i, c_i) and (a_i, b_{i+1}, c_{i+1}) component wise together in order to aggregate these signatures is still not sufficient. We illustrate the problem on the following toy examples and suggest a new aggregation technique that we call *aggregate-extension technique*. Let g^a, g^b be public keys and let g^{s_a} (resp. g^{s_b}) denote the signatures under the keys g^a (resp. under the key g^b). Now, think about a verification equation that computes a non-generate, bilinear map $\mathbf{e}(g^a\,g^b, g^{s_a}g^{s_b})$ to verify both signatures. The upcoming problem results from the properties of the bilinear map: $\mathbf{e}(g^a\,g^b, g^{s_a}g^{s_b}) = \mathbf{e}(g^a, g^{s_a}g^{s_b}) \cdot \mathbf{e}(g^b, g^{s_a}g^{s_b}) = \mathbf{e}(g^a, g^{s_a}) \cdot \mathbf{e}(g^a, g^{s_b}) \cdot \mathbf{e}(g^b, g^{s_a}) \cdot \mathbf{e}(g^b, g^{s_b})$. If we consider the pairs $\mathbf{e}(g^a, g^{s_b})$ and $\mathbf{e}(g^b, g^{s_a})$, then we have two signatures that must verify under the wrong keys, which is impossible. To handle these elements, we slightly extend the aggregate by a single group element D. This element serves as a "programmable memory" which stores these (undesired) elements. In the example the "memory" element D would have the form $g^{as_b+bs_a}$ and the corresponding equation would have the form $\mathbf{e}(g^a g^b, g^{s_a}g^{s_b}) \cdot \mathbf{e}(g, g^{as_b+bs_a})^{-1}$. Finally, we apply the aggregate extension technique to the CL signature scheme.

Knowledge-of-Secret-Key. As shown in Figure 1, our construction is secure in the knowledge-of-secret-key setting (KOSK), where the adversary must prove knowledge of each private key corresponding to any maliciously generated public key. This setting can be implemented through a CA that asks the user to perform a proof-of-knowledge of the secret key when registering a public key. Alternatively, all user can generate their keys jointly [30], or the public keys come with an extractable non-interactive proof of knowledge. While the construction of Lu et al. [27] relies also on this assumption, it is clear that a solution outside the KOSK is desirable. We refer the reader to [5,3,34] for a comprehensive discussion about this setting.

Related Work. Boneh et al. [10] introduced the concept of aggregate signatures and gave a first instantiation that is provably secure in the random oracle model. At Eurocrypt 2004, Lysyanskaya et al. suggested the concept of sequential aggregate signatures, proposed a formal security model, and gave a first solution based on general assumptions [28] that is also secure in the random oracle. The first instantiation that is secure in the standard model, but in a model that is weaker than one of [28], was proposed by Lu et al. [27]. Neven suggested at Eurocrypt 2008 the notion of sequential aggregate signed data, which generalized sequential aggregate signature in the sense that these scheme compress the whole data [31]. Fischlin et al. adopt the notion of history-freeness from Eikemeyer et al. [18] to the case of sequential aggregate signatures [19]. The basic idea is to allow the aggregate-so-far only depend (explicitly) on the local message and signing key, but not on the previous messages and public keys in the sequence. The benefit of this notion is that one does not need to take care about the key management and that expensive verification queries are not necessary anymore. Eikemeyer et al. considered the notion of history-freeness only in the context of aggregate message authentication codes [25]. Multisignatures are similar in the sense that a group of signers sign the same message [24]. Many researches suggested different solutions, such as, e.g., [21,33,32]. However, the size of the multisignature proposed by some solutions grows linear in the number of signers [24] and some such schemes cannot be considered as secure [21]. Boldyreva simplified and generalized the security model of [30]. The author also gave the first solution that does not require a priori knowledge of a subgroup of signers and that is provably secure in the random oracle [5]. The first solution that is secure in the standard model has been suggested by Lu et al. [27]. Recently, Boldyreva et al. introduced the concept of ordered multisignatures [7], where the signers attest to a common message as well as to the order in which they signed.

2 Preliminaries

Notations. If x is a string then $|x|$ denotes its length, and if S is a set $|S|$ denotes its size. By $a_1\|\ldots\|a_n$ we denote a string encoding of $a_1, \ldots, a_n$ from which $a_1, \ldots, a_n$ are uniquely recoverable. If A is an algorithm then $y \leftarrow A(x_1, x_2, \ldots)$ denotes the operation of running A on inputs $x_1, x_2, \ldots$ and letting y denote the output of A. Unless otherwise indicated, we assume that all algorithms run in probabilistic polynomial-time and refer to them as being efficient.

2.1 Bilinear Groups

We denote by $\mathbb{G}$ and $\mathbb{G}_T$ two multiplicative groups of prime order p and consider $g \in \mathbb{G}$ such that: all group operations can be computed efficiently; g is a generator of $\mathbb{G}$; $\mathbf{e}$ is a bilinear pairing: $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, i.e., $\mathbf{e}$ is an efficiently computable map satisfying the following properties:

- Non-degeneracy: $\mathbf{e}\left(g, g\right) \neq 1$ and is thus a generator of $\mathbb{G}_T$;
- Bilinearity: $\forall u, v \in \mathbb{G}, \forall a, b \in \mathbb{Z}$: $\mathbf{e}\left(u^a, v^b\right) = \mathbf{e}\left(u, v\right)^{ab}$.

As a result of the bilinearity, it holds that $\mathbf{e}(g^a, g) = \mathbf{e}(g, g)^a = \mathbf{e}(g, g^a)$.

Definition 1 (Bilinear-Group Generation). *The algorithm $\mathcal{G}$ that outputs (descriptions of) $p, \mathbb{G}, \mathbb{G}_T, \mathsf{e}$ as above is called* bilinear-group generation algorithm*, and $\mathbb{G}$ is a bilinear group.*

2.2 Signature Scheme

Definition 2 (Signature Scheme). *A signature scheme $\mathsf{DS} = (\mathsf{PKg}, \mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ is a tuple of algorithms:*

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ *returns some global information I.*
Key Generation. $\mathsf{Kg}(I)$ *outputs a keypair (sk, pk).*
Signing. *The input of the signing algorithm $\mathsf{Sig}(sk, M)$ is a signing key sk and a message M; it outputs a signature σ.*
Verification. $\mathsf{Vf}(pk, M, \sigma)$ *outputs 1 iff σ is a signature on M under pk.*

The security of signature schemes is proven against existential forgery under adaptive chosen message attacks (EU-CMA) due to Goldwasser, Micali, and Rivest [20]. In this model, an adversary adaptively invokes a signing oracle and is successful if it outputs a signature on a *fresh* message.

Definition 3 (Unforgeability). *A signature scheme* DS *is* unforgeable under adaptive chosen message attacks *(EU-CMA) if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DS}}$ evaluates to 1 is negligible (as a function of λ), where*

```
***Experiment*** $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DS}}(\lambda)$
    $I \leftarrow \mathsf{PKg}(1^\lambda)$
    $(sk, pk) \leftarrow \mathsf{Kg}(I)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}(sk,\cdot)}(pk)$
    *Return* 1 *iff* $\mathsf{Vf}(pk, m^*, \sigma^*) = 1$ *and* $\mathcal{A}$ *has never queried* $\mathsf{Sig}(sk, \cdot)$ *about* $m^*$.
```

A signature scheme DS *is (t, q_S, ϵ)*-secure *if no adversary running in time at most t, invoking the signing oracle at most q_S times, outputs a valid forgery (m^*, σ^*) with probability larger than ϵ.*

2.3 The CL Signature Scheme

The signature scheme due to Camenisch and Lysyanskaya has been introduced at CRYPTO 2004 and its security relies on the interactive LRSW assumption [14]. The LRSW assumption, due to Lysyanskaya, Rivest, Sahai, and Wolf, is hard in the generic group model (as defined by Shoup [35]) [29]. Moreover, it is widely established and it is the basis for many constructions such as, e.g., [14,1,4,12,16,26].

Assumption (LRSW Assumption). Suppose that $\mathbb{G}$ is group of prime order p and that g is a generator of $\mathbb{G}$. Let $X, Y \in \mathbb{G}$ such that $X = g^x$ and $Y = g^y$

for some $x, y \in \mathbb{Z}_p$ and let $\rho := (p, \mathbb{G}, \mathbb{G}_T, g, \mathbf{e}, X, Y)$ and let $O_{X,Y}$ be an oracle that on input a value $M \in \mathbb{Z}_p$ outputs a triplet $(a, a^y, a^{(x+Mxy)})$ for a randomly chosen $a \in \mathbb{G}$. Then for all efficient algorithms $\mathcal{A}^{O_{x,y}}$, $\nu(\lambda)$ defined as follows is a negligible function (in λ):

$$\begin{aligned}&\mathrm{Prob}[x \leftarrow \mathbb{Z}_p\, ;\, y \leftarrow \mathbb{Z}_p\, ;\, X \leftarrow g^x\, ;\, Y \leftarrow g^y;\\ &\quad (Q, M, a, b, c) \leftarrow \mathcal{A}^{O_{x,y}}(\rho) : M \notin Q \,\wedge\, a \in \mathbb{G} \,\wedge\, b = a^y \,\wedge\, c = a^{x+Mxy}] = \nu(\lambda),\end{aligned}$$

where Q is the set of oracle queries. Based on this assumption, the signature scheme consists of the following algorithms:

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ executes the bilinear-group generation algorithm $\mathcal{G}$ to obtain output $(p, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, it chooses generator $g \in \mathbb{G}$ and returns $I = (p, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, g)$.

Key Generation. $\mathsf{Kg}(I)$ picks two random elements $x \leftarrow \mathbb{Z}_p$ and $y \leftarrow \mathbb{Z}_p$. It sets $X \leftarrow g^x$ and $Y \leftarrow g^y$. The private key is $sk = (I, x, y)$ and the public key is $pk = (I, X, Y)$.

Signing. The input of the algorithm $\mathsf{Sig}(sk, M)$ is a secret key $sk = (I, x, y)$ and a message $M \in \mathbb{Z}_p$. It selects a random element $r \leftarrow \mathbb{Z}_p$ and computes the signature $\sigma = (a, b, c) \leftarrow (g^r, g^{r\,y}, g^{r\,(x+Mxy)})$.

Verification. To check that $\sigma = (a, b, c)$ is a valid signature on message $M \in \mathbb{Z}_p$ under a public-key $pk = (I, X, Y)$, the algorithm $\mathsf{Vf}(pk, M, \sigma)$ verifies that

$$\mathbf{e}(a, Y) = \mathbf{e}(g, b) \quad \text{and} \quad \mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^M = \mathbf{e}(g, c) \quad \text{hold.}$$

The CL-signature scheme is provably secure in the standard model, assuming that the LRSW assumption holds.

Proposition 1 ([14]). *If the LRSW assumption is hard relative to $\mathcal{G}$, then the CL-signature scheme is unforgeable under adaptively chosen message attacks.*

3 Sequential Aggregate Signature

Boneh et al. proposed a new signature primitive called an aggregate signature [10]. Aggregate signature generalize multisignatures [24] as they combine several signatures, on distinct messages from different users, into a single signature that has *roughly the same size* as an ordinary signature. In such a scheme, the individual users generate their signature independently in advance. The signatures are then combined into a single aggregate by a third, maybe untrusted, party.

Sequential aggregate signature schemes (SAS) are different in the sense that aggregation is performed sequentially. Each individual signer gets as input an aggregate-so-far σ' and "adds" its own signature onto the aggregate. Signing and aggregation are therefore a combined process. The resulting aggregate has roughly the same size as a standard signature.

More formally, the input of the aggregate-signing algorithm is a secret key *sk*, a message M_i to be signed and an aggregate-so-far tuple $(\sigma', \mathbf{M}, \mathbf{pk})$. This tuple consists of an aggregate σ', of a sequence of messages $\mathbf{M} = (M_1, \ldots, M_{i-1})$, and of a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_{i-1})$. This algorithm outputs a new aggregate σ for message and public key sequences $\mathbf{M}||M := (M_1, \ldots, M_{i-1}, M_i)$ and $\mathbf{pk}||pk := (pk_1, \ldots, pk_{i-1}, pk_i)$, such that the aggregate has roughly the same size as an ordinary signature.

3.1 Definition

Definition 4 (Sequential Aggregate Signature). *A sequential aggregate signature scheme is a tupple of efficient algorithms* $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$*, where:*

System Parameter. $\mathsf{SeqPKg}(1^\lambda)$ *returns the system parameters* I*.*
Key Generation. $\mathsf{SeqKg}(I)$ *generates a key pair* (sk, pk)*.*
Signature Aggregation. *The input of the aggregation algorithm* SeqAgg *is a tuple* $(sk, M_i, \sigma', \mathbf{M}, \mathbf{pk})$ *consisting of a secret key* sk*, a message* $M_i \in \{0,1\}^*$*, an aggregate* σ'*, and sequences* $\mathbf{M} = (M_1, \ldots, M_{i-1})$ *of messages and* $\mathbf{pk} = (pk_1, \ldots, pk_{i-1})$ *of public keys. It computes the aggregate* σ *for the message sequence* $\mathbf{M}||M = (M_1, \ldots, M_{i-1}, M_i)$ *and the key sequence* $\mathbf{pk}||pk = (pk_1, \ldots, pk_{i-1}, pk_i)$*.*
Aggregate Verification. *The algorithm* $\mathsf{SeqAggVf}(\sigma, \mathbf{M}, \mathbf{pk})$ *takes as input an aggregate* σ*, a sequence of messages* $\mathbf{M} = (M_1, \ldots, M_\ell)$*, as well as a sequence of public keys* $\mathbf{pk} = (pk_1, \ldots, pk_\ell)$*. It returns a bit.*

The sequential aggregate signature scheme is complete *if for any finite sequence of key pairs* $(sk, pk), (sk_1, pk_1), \ldots, (sk_n, pk_n) \leftarrow \mathsf{SeqKg}(1^\lambda)$*, for any (finite) sequence* $M_1, \ldots, M_n$ *of messages with* $M_i \in \{0,1\}^*$*, for any* $\sigma \leftarrow \mathsf{SeqAgg}(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$ *with* $\mathsf{SeqAggVf}(\sigma', \mathbf{M}, \mathbf{pk}) = 1$ *or* $\sigma' = \emptyset$*, we have* $\mathsf{SeqAggVf}(\sigma, \mathbf{M}||M, \mathbf{pk}||pk) = 1$*.*

3.2 Security Model

The security of a sequential aggregate signature is defined in the chosen-key model. Thereby, the adversary gets as input a single key (referred to as the challenge key pk_c), it is allowed to choose all other keys, and has access to a sequential aggregate signing oracle $\mathsf{OSeqAgg}$ and to a key registration oracle RegKey. The input of the oracle $\mathsf{OSeqAgg}$ is a message M to be signed under the challenge key sk_c, an aggregate-so-far σ', on a set of messages $\mathbf{M}$, under public key sequence $\mathbf{pk}$; the oracle then outputs a new aggregate σ that contains also the signature σ' on the challenge message M. The second oracle RegKey takes as input a private key sk and the corresponding public key pk. The task for the adversary is to output a sequential aggregate signature such that the aggregate contains a signature on a "fresh" message M_c for the challenge key pk_c. A message is "fresh" in the usual sense, namely that it has not been sent to the aggregate signing oracle [27]. Note that the requirement that the challenge public-key pk_c has not been registered, i.e., $pk_c \notin C$, can be dropped in the KOSK because the adversary would have to compute the corresponding private key sk_c to ask such a question. Here, however, we keep this restriction to simplify the proof.

Definition 5 (Aggregate-Unforgeable). *A sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *is* aggregate-unforgeable *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{SAS}}$ *evaluates to* 1 *is negligible (as a function of* λ*), where*

```
Experiment $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{SAS}}(\lambda)$
    $I \leftarrow \mathsf{SeqPKg}(1^{\lambda})$
    $(sk_c, pk_c) \leftarrow \mathsf{SeqKg}(I)$
    $(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{RegKey}(\cdot,\cdot),\mathsf{OSeqAgg}(sk_c,\cdots)}(pk_c)$
    Let $C$ denote the list of all certified key $(sk_1, pk_1), \ldots, (sk_\ell, pk_\ell)$
        and let $M_c$ be the message that corresponds to $pk_c$.
    Return 1 iff $pk_i \neq pk_j$ for all $i \neq j$ and $\mathsf{SeqAggVf}(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) = 1$ and
        $pk_c \notin C$ and $\mathcal{A}$ has never sent $M_c$ to $\mathsf{OSeqAgg}(sk_c, \cdots)$.
```

A sequential aggregate signature scheme DS *is* (t, q_S, q_N, ϵ)-aggregate unforgeable *if no adversary running in time at most* t*, invoking the signing oracle at most* q_S *times, registering at most* q_N *keys, outputs a valid forgery* $(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*)$ *with probability larger than* ϵ.

3.3 Intuition for the Construction

We consider signers who have unique key-pairs as in our signature scheme, consisting of two private values x, y, and two public keys $X = g^x, Y = g^y$ for a shared generator g of a group $\mathbb{G}$. Thereby the signature consists of three group elements A, B, and C. We view the element $A = g^r$ as the randomness; $B = g^{ry}$ as a "commitment" to the randomness; and $C = g^{r(x+Mxy)}$ as the signature on the message M.

An aggregate in our case consists of four group elements A, B, C, D where the first three elements A, B, C play the same role as in the standard signature scheme. The element D serves as a programmable "memory". We illustrate the necessity for this element on the following example, where a second user wishes to aggregate his signature on a message M_2 onto the aggregate $\sigma_1 = (A_1, B_1, C_1)$. Let $(sk_i, pk_i) = ((x_i, y_i), (g^{x_i}, g^{y_i}))$ denote the key-pair of the i-th user (here for $i = 1, 2$) and let M_2 be the message to be signed. This second user first views $A_1 = g^r$ as the randomness to be used for his signature. It then computes $B \leftarrow B_1 \cdot A_1^{y_2} = g^{r(y_1+y_2)}$ and $C \leftarrow C_1 \cdot A_1^{x_2} A_1^{M_2 x_2 y_2} = g^{r(x_1+M_1x_1y_1+x_2+M_2x_2y_2)}$. If we now wish to verify that both signatures are valid, we must to check that:

$$\prod_i \left(\mathbf{e}(X_i, A) \cdot \mathbf{e}(X_i, B)^{M_i}\right) = \mathbf{e}(g, A)^{x_1+x_2} \cdot \mathbf{e}(g, B)^{M_1x_1+M_2x_2} = \mathbf{e}(g, C)$$

holds. Unfortunately, this is not the case, because an "error" occurs during the evaluation of the second bilinear map:

$$\begin{aligned}\mathbf{e}(g, B)^{M_1x_1+M_2x_2} &= \mathbf{e}(g, B)^{M_1x_1} \cdot \mathbf{e}(g, B)^{M_2x_2} \\ &= \mathbf{e}(g, A^{y_1+y_2})^{M_1x_1} \cdot \mathbf{e}(g, A^{y_1+y_2})^{M_2x_2} \\ &= \mathbf{e}(g, A)^{M_1y_1x_1+M_1y_2x_1} \cdot \mathbf{e}(g, A)^{M_2y_1x_2+M_2y_2x_2}.\end{aligned}$$

In fact, the evaluation results in two "noisy" elements: $M_1 y_2 x_1$ and $M_2 y_1 x_2$. We now program the element D such that it cancels all these elements out.

3.4 The Construction

An aggregate $\sigma := (A, B, C, D)$ on messages $\mathbf{M} := (M_1, \dots, M_\ell)$ under public keys $\mathbf{pk} := (pk_1, \dots, pk_\ell)$ has the form:

$$A = g^r \quad ; \quad B = \prod_i g^{r\,y_i} \quad ; \quad C = \prod_i g^{r(x_i + M_i x_i y_i)} \quad ; \quad D = \prod_{i \neq j} g^{M_i x_i y_j} \quad ;$$

where A may be seen as some "shared" randomness. The scheme $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqSign}, \mathsf{SeqVf})$ is defined as follows:

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ chooses two suitable groups $\mathbb{G}, \mathbb{G}_T$, a generator $g \in \mathbb{G}$ and returns $I = (\mathbb{G}, \mathbb{G}_T, g, \mathbf{e})$.

Key Generation. For a particular signer, algorithm $\mathsf{SeqKg}(I)$ picks $x \leftarrow \mathbb{Z}_p$ and $y \leftarrow \mathbb{Z}_p$ and sets $X \leftarrow g^x$ as well as $Y \leftarrow g^y$. It returns sk as (I, x, y) and pk as $pk = (I, X, Y)$.

Sequential Signing. Algorithm SeqAgg takes as input a secret signing key $sk = (I, x, y)$, a message $M \in \mathbb{Z}_p$, an aggregate-so-far σ', a sequence of messages $\mathbf{M} = (M_1, \dots . M_i)$, and a sequence of public keys $\mathbf{pk} = (pk_1, \dots, pk_i)$. The algorithm first checks that $|\mathbf{M}| = |\mathbf{pk}|$ and that $\mathsf{SeqVf}(\sigma', \mathbf{M}, \mathbf{pk}) = 1$. If so, it parses σ' as (A', B', C', D'), and it sets

$$w_1 \leftarrow A' \quad ; \quad w_2 \leftarrow B' \cdot (A')^y \quad ; \quad w_3 \leftarrow C' \cdot (A')^{x+Mxy} \quad ;$$

$$\text{and} \quad w_4 \leftarrow D' \cdot \left(\prod_{j=1}^{i} X_j^{yM_j} \cdot Y_j^{xM} \right) .$$

Note that the first signer uses $(1, 1, 1, 1)$ as the aggregate-so-far. The tuple (w_1, w_2, w_3, w_4) forms a valid aggregate signature (for our algorithm) on messages $\mathbf{M}||M$ under public keys $\mathbf{pk}||pk$. Now, we have to re-randomize the aggregate, or an attacker could forge [2]:

$$\tilde{r} \leftarrow \mathbb{Z}_p^* \quad ; \quad A \leftarrow w_1^{\tilde{r}} \quad ; \quad B \leftarrow w_2^{\tilde{r}} \quad ; \quad C \leftarrow w_3^{\tilde{r}} \quad ; \quad D \leftarrow w_4 .$$

It follows easily that $\sigma = (A, B, C, D)$ is also a valid aggregate on messages $\mathbf{M}||M$ under public keys $\mathbf{pk}||pk$ with randomness $g^{r\,\tilde{r}}$.

[2] In particular, consider an adversary that chooses some random keys (x, y), a message M, and a randomness r. It computes the signature as an honest signer and sends it as an aggregate-so-far to the oracle (having the secret keys (x_c, y_c)) together with a challenge message M_c. If the oracle does not re-randomize the aggregate, then the adversary receives an element $C = g^{r(x+Mxy+x_c+M_c x_c y_c)}$. As the adversary knows the randomness r, it can easily obtain the value $XY_c = g^{x_c y_c}$ and can forge a signature on an arbitrary message.

Aggregate Verification. The input of algorithm SeqVf consists of a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_\ell)$, a sequence of message $\mathbf{M} = (M_1, \ldots, M_\ell)$ and an aggregate σ. It parses the aggregate σ as (A, B, C, D). It first checks that $|\mathbf{M}| = |\mathbf{pk}|$ and that no public key appears twice in $\mathbf{pk}$. Afterwards, it validates the structure of the elements A, B, and D:

$$\mathbf{e}(A, \prod_i Y_i) = \mathbf{e}(g, B) \quad \text{and} \quad \prod_{i \neq j} \mathbf{e}(X_i, Y_j)^{M_i} = \mathbf{e}(g, D),$$

and proves that C is formed correctly:

$$\prod_i \left(\mathbf{e}(X_i, A) \cdot \mathbf{e}(X_i, B)^{M_i} \right) \cdot \mathbf{e}(A, D)^{-1} = \mathbf{e}(g, C)\,.$$

If all equations are valid, SeqVf outputs 1; otherwise it returns 0.

Completeness follows inductively.

Performance and Symmetric Verification. The verification equation of element C suggests that, for N signers, a total number of $3\,N$ pairings have to be computed. However, this is not the case. We chose to present the computation in this form for the sake of esthetics. A more computationally efficient version of the verification equation is the following:

$$\mathbf{e}(\prod_i X_i, A) \cdot \mathbf{e}(\prod_i X_i^{M_i}, B) \cdot \mathbf{e}(A, D)^{-1} = \mathbf{e}(g, C)$$

which is identical to the one above, but which requires the computation of only three pairings irrespective of the number of signers. A more efficient variant for the verification of the element D is the following

$$\prod_i \mathbf{e}(X_i^{M_i}, \prod_j Y_i) = \mathbf{e}(g, D).$$

This equation involves only n pairing computations (instead of $n^2/2$).

Our construction has the additional feature, similar to [10,27], that the verifier does not need to know the order in which the aggregate was created. This property, sometimes called symmetric verification, is useful for several applications such as, e.g., building an optimistic fair exchange out of any sequential two-party multisignature as shown by Dodis et al. [17].

Applications to Ordered Multisignature. In a multisignature scheme a group of signers sign the *same* message such that the signature has roughly the same size as an ordinary signature. Recently, Boldyreva, Gentry, O'Neill, and Yum generalized multisignatures to *ordered multisignatures* (OMS). This primitives has the additional property that the adversary cannot re-order the position of honest signers [7]. The authors also suggest a generic transformation that turns every aggregate signature scheme into an orderd multisignature scheme. The idea is to encode the position of each signer into the message, i.e., suppose the the signer is at position i and wishes to sign the common messsage M, then it runs the aggregate signing algorithm on the message $M\|i$. Applying this transformation to our scheme yields also the first OMS with short keys in the standard model.

3.5 Proof of Security

In this section we prove the security of our scheme.

Theorem 1. *The aggregate signature scheme is $(t, q_C, q_S, n, \epsilon)$-secure with respect to Definition 4, if the CL-signature scheme (t', q', ϵ') unforgeable on $\mathbb{G}$, where*

$$t' = t + O(q_C + nq_S + n) \quad \text{and} \quad q' = q_S \quad \text{and} \quad \epsilon' = \epsilon\,.$$

Proof. We use a proof by contradiction. Suppose that $\mathcal{A}$ is an adversary which forges the sequential aggregate signature scheme. We then show how to construct an algorithm $\mathcal{B}$ that breaks the unforgeability of the underlying CL-signature scheme.

Setup. The algorithm $\mathcal{B}$ gets as input a public-key $pk_c = (I, X, Y)$. It initializes the query list $C \leftarrow \emptyset$ and runs a black-box simulation of the attacker $\mathcal{A}$ on input pk_c.

Certification Queries. If algorithm $\mathcal{A}$ wishes to certify a public key $pk = (I, X, Y)$, it hands over the corresponding secret key $sk = (I, x, y)$. Algorithm $\mathcal{B}$ verifies that (x, y) is the private key corresponding to pk. If so, it adds (sk, pk) to the list of certified keys, i.e., $C \leftarrow C \cup (sk, pk)$. Otherwise, it rejects the query.

Signature Queries. Whenever the algorithm $\mathcal{A}$ invokes its aggregate signing oracle $\mathsf{SeqAgg}(sk, \cdot)$ on: a message M, an aggregate-so-far σ', a sequence of messages $\mathbf{M}'$, and a sequence of public keys $\mathbf{pk}'$, then algorithm $\mathcal{B}$ behaves as follows: it first checks that σ' is a valid aggregate on messages $\mathbf{M}'$ under public keys $\mathbf{pk}'$; afterwards, it checks that all public keys $pk \in \mathbf{pk}$ are certified, that each key appears only once, and that $|\mathbf{pk}'| = |\mathbf{M}'|$. If any of these conditions is violated, then $\mathcal{B}$ returns `invalid`. In the following let $|\mathbf{M}'| = |\mathbf{pk}'| =: q$.

Otherwise, algorithm $\mathcal{B}$ invokes its own signing oracle $\mathsf{Sig}(sk_c, \cdot)$ on M and receives the output signature σ. Note that σ is also an aggregate on message M under the public key pk_c. Next, algorithm $\mathcal{B}$ "adds" the missing q signatures onto the aggregate. More precisely, it sets: $\sigma_0 \leftarrow \sigma$, $M_0 \leftarrow M$, and $pk_0 \leftarrow pk_c$, and it executes $\sigma_i \leftarrow \mathsf{SeqAgg}(sk_i, M_i, \sigma_{i-1}, \mathbf{M}_{i-1}, \mathbf{pk}_{i-1})$, where $\mathbf{M}_{i-1} = (M_1, \ldots, M_{i-1})$ and $\mathbf{pk}_{i-1} = (pk_1, \ldots, pk_{i-1})$ for $i = 1, \ldots, q$. Observe that this is possible since all public keys in $\mathbf{pk}$ are registered, and that each private key is stored in C. Afterwards, $\mathcal{B}$ returns the aggregate $\sigma \leftarrow \sigma_i$ on messages $\mathbf{M} \leftarrow \mathbf{M}'||M$ under public keys $\mathbf{pk} \leftarrow \mathbf{pk}'||pk$.

Output. Finally $\mathcal{A}$ stops, eventually outputting a valid forgery σ' on messages $\mathbf{M}^*$ under public keys $\mathbf{pk}^*$. This forgery is valid if the following relations hold:

- $|\mathbf{pk}^*| = |\mathbf{M}^*|$;
- $\mathsf{SeqVf}(\sigma^*, \mathbf{M}^*, \mathbf{pk}^*) = 1$,
- $pk_c \in \mathbf{pk}^*$ say at index i_c,
- all keys in $\mathbf{pk}^*$ except for the challenge key pk_c are registered,
- $\mathcal{A}$ never queried its sequential signing oracle about the message M_{i_c}.

W.l.o.g. we assume that the challenge key pk_c is stored at the first position in $\mathbf{pk}^*$, i.e., $i_c = 1$ and let $|\mathbf{pk}^*| =: q$. Just as in the case of [27], this is not a restriction, because the verification equation does not take into account the order of the participants.

Next, for each $2 \leq i \leq q$ let (I, x_i, y_i) be the secret key corresponding to $pk_i = (I, X_i, Y_i)$. Algorithm $\mathcal{B}$ computes:

$$a^* \leftarrow A^* \ ; \ b^* \leftarrow B^* \cdot \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1} \quad ; \quad c^* \leftarrow C^* \cdot \left((A^*)^{\sum_{i=2}^{q} x_i + M_i x_i y_i}\right)^{-1},$$

and outputs $(M^*, \sigma^*) \leftarrow (M_1, (a^*, b^*, c^*))$.

For the analysis, first note that $\mathcal{B}$ is efficient since $\mathcal{A}$ runs in polynomial-time and since the attacker $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view. In the following, we show that $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does. Thus, we have to show firstly that algorithm $\mathcal{B}$ outputs a valid CL-signature, and secondly, that $\mathcal{B}$ has never queried the message to its signing oracle. The second condition follows easily from the assumption that $\mathcal{A}$ outputs a valid forgery. For the first property we show that we can divide all other signatures out such that only the signature for the challenge key remains. Consider the first verification equation of the aggregate signature scheme. We know that:

$$\mathbf{e}(A^*, \prod_i Y_i) = \mathbf{e}(g^r, \prod_i g^{y_i}) = \mathbf{e}(g, \prod_i g^{r y_i}) = \mathbf{e}(g, \prod_i (A^*)^{y_i}) = \mathbf{e}(g, B^*) \quad (1)$$

for some r; this implies that:

$$\begin{aligned}
\mathbf{e}(g, b^*) = \mathbf{e}\left(g, B^* \cdot \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) &= \mathbf{e}\left(g, B^*\right) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&\overset{(1)}{=} \mathbf{e}(A^*, \prod_{i=1}^{q} Y_i) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&= \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i})) \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&= \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i})) \cdot \mathbf{e}(A^*, (g^{\sum_{i=2}^{q} y_i})^{-1}) = \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i} g^{\sum_{i=2}^{q} -y_i}) \\
&= \mathbf{e}(a^*, Y_1)\,.
\end{aligned}$$

Now, the second and third verification equations of the aggregate signature scheme prove that:

$$\begin{aligned}
\prod_{i \neq j} \mathbf{e}(X_i, Y_j)^{M_i^*} &= \mathbf{e}(g, g)^{\sum_{i \neq j} M_i^* x_i y_j} = \mathbf{e}(g, g^{\sum_{i \neq j} M_i^* x_i y_j}) \\
&= \mathbf{e}(g, \prod_{i \neq j} g^{M_i^* x_i y_j}) = \mathbf{e}(g, D^*)
\end{aligned}$$

and that:

$$\prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \mathbf{e}(A^*, D^*)^{-1} = \mathbf{e}(g, C^*)\,. \tag{2}$$

This implies that:

$$\begin{aligned}
\mathbf{e}(g, c^*) &= \mathbf{e}(g, C^*) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
&\overset{(2)}{=} \prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \\
&\qquad\qquad \cdot \mathbf{e}(A^*, D^*)^{-1} \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right)\,.
\end{aligned}$$

Observe that $\mathbf{e}(g, B^*) \overset{(1)}{=} \mathbf{e}(g, \prod_i (A^*)^{y_i}) = \mathbf{e}(g, \prod_i g^{r y_i})$ and thus,

$$\prod_i \mathbf{e}(X_i, B^*)^{M_i^*} = \prod_i \mathbf{e}(g, B^*)^{M_i^* x_i} = \prod_i \mathbf{e}(g, \prod_i g^{r y_i})^{M_i^* x_i}\,. \tag{3}$$

Next, we replace the index i of the product $\prod_i g^{r y_i}$ with j. Using the results of these equations, we have the following:

$$\begin{aligned}
\mathbf{e}(g, c^*) &= \prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \\
&\qquad\qquad \cdot \mathbf{e}(A^*, D^*)^{-1} \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
&\overset{(3)}{=} \prod_i \left[\mathbf{e}(A^*, X_i)\right] \cdot \prod_i \left[\mathbf{e}(g, \prod_j g^{r y_j})^{M_i^* x_i}\right] \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \\
&\qquad\qquad \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
&= \prod_i \left[\mathbf{e}(A^*, X_i)\right] \cdot \mathbf{e}(A^*, \prod_i \prod_j g^{M_i^* x_i y_j}) \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \\
&\qquad\qquad \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right)\,.
\end{aligned}$$

In the following, we divide the products $\mathbf{e}(A^*, \prod_i \prod_j g^{M_i^* x_i y_j})$ into two parts; the first part consists of all factors for which $i = j$, while the second part contains the remaining factors:

$$\begin{aligned}
\mathbf{e}(g, c^*) = \prod_{i} \left[\mathbf{e}(A^*, X_i)\right] \cdot \mathbf{e}(A^*, \prod_{i=j} g^{M_i^* x_i y_j} \prod_{i \neq j} g^{M_i^* x_i y_j}) \cdot \\
\mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
= \prod_{i} \left[\mathbf{e}(A^*, X_i)\right] \cdot \mathbf{e}(A^*, \prod_{i=j} g^{M_i^* x_i y_j}) \cdot \\
\cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j}) \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
= \prod_{i=1}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \prod_{i=2}^{q} \mathbf{e}\left(A^*, \left(X_i \cdot g^{M_i^* x_i y_i}\right)^{-1}\right) \\
= \prod_{i=1}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)^{-1}\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)^{-1}\right)\right] \\
= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
\cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)^{-1}\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)^{-1}\right)\right] \\
= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
\cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)\right)^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)\right)^{-1}\right] \\
= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
\cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)\right)\right]^{-1} \\
= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \\
= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(g^r, g^{M_1^* x_1 y_1}) \cdot \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(g^{x_1}, g^{r y_1})^{M_1^*} \\
= \mathbf{e}(a^*, X_1) \cdot \mathbf{e}(X_1, b^*)^{M_1^*}
\end{aligned}$$

as desired. Thus, $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does.

Acknowledgments. The author thanks Heike Schröder, Cristina Onete, Markus Rückert, and the anonymous reviewers for valuable comments. This work was partially supported by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of

Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

References

1. Ateniese, G., Camenisch, J., de Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS), pp. 92–101. ACM, New York (2005)
2. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (2007)
3. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM Conference on Computer and Communications Security 2006, pp. 390–399. ACM Press, New York (2006)
4. 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)
5. Boldyreva, A.: Efficient threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2002)
6. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS 2007), pp. 276–285. ACM Press, New York (2007)
7. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: New multiparty signature schemes for network routing applications. ACM Transactions on Information and System Security (TISSEC) 12(1) (2008)
8. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. Journal of Cryptology 21(2), 149–177 (2008)
9. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
10. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
12. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: Efficient periodic n-times anonymous authentication. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS), pp. 201–210. ACM Press, New York (2006)
13. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007)

14. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
15. Chatterjee, S., Hankerson, D., Knapp, E., Menezes, A.: Comparing two pairing-based aggregate signature schemes. Cryptology ePrint Archive, Report 2009/060 (2009), `http://eprint.iacr.org/`
16. Damgård, I., Dupont, K., Pedersen, M.Ø.: Unclonable group identification. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 555–572. Springer, Heidelberg (2006)
17. Dodis, Y., Lee, P.J., Yum, D.H.: Optimistic fair exchange in a multi-user setting. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 118–133. Springer, Heidelberg (2007)
18. Eikemeier, O., Fischlin, M., Götzmann, J.F., Lehmann, A., Schröder, D., Schröder, P., Wagner, D.: History-free aggregate message authentication codes. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 309–328. Springer, Heidelberg (2010)
19. Fischlin, M., Lehmann, A., Schröder, D.: History-free sequential aggregate signatures. Cryptology ePrint Archive, Report 2011/231 (2011), `http://eprint.iacr.org/`
20. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
21. Harn, L.: Group-oriented (t, n) threshold digital signature scheme and digital multisignature. IEE Proceedings of Computers and Digital Techniques 141(5), 307–313 (1994)
22. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 21–38. Springer, Heidelberg (2008)
23. IOC: Ioc tsunami website (2009), `http://ioc3.unesco.org/`, `http://ioc3.unesco.org/indotsunami/`
24. Itakura, K., Nakamura, K.: A public key cryptosystem suitable for digital multisignatures. NEC Research & Development 71, 1–8 (1983)
25. Katz, J., Lindell, A.Y.: Aggregate message authentication codes. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 155–169. Springer, Heidelberg (2008)
26. Kiayias, A., Zhou, H.-S.: Concurrent blind signatures without random oracles. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 49–62. Springer, Heidelberg (2006)
27. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
28. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)
29. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Walker, M. (ed.) Cryptography and Coding 1999. LNCS, vol. 1746, pp. 184–199. Springer, Heidelberg (1999)
30. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: ACM Conference on Computer and Communications Security 2001, pp. 245–254. ACM Press, New York (2001)

31. Neven, G.: Efficient sequential aggregate signed data. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 52–69. Springer, Heidelberg (2008)
32. Ohta, K., Okamoto, T.: A digital multisignature scheme based on the fiat-shamir scheme. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 139–148. Springer, Heidelberg (1993)
33. Okamoto, T.: A digital multisignature schema using bijective public-key cryptosystems. ACM Trans. Comput. Syst. 6(4), 432–441 (1988)
34. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (2007)
35. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)

Key Exchange in IPsec Revisited: Formal Analysis of IKEv1 and IKEv2

Cas Cremers

ETH Zurich, Switzerland
cas.cremers@inf.ethz.ch

Abstract. The IPsec standard aims to provide application-transparent end-to-end security for the Internet Protocol. The security properties of IPsec critically depend on the underlying key exchange protocols, known as IKE (Internet Key Exchange).

We provide the most extensive formal analysis so far of the current IKE versions, IKEv1 and IKEv2. We combine recently introduced formal analysis methods for security protocols with massive parallelization, allowing the scope of our analysis to go far beyond previous formal analysis. While we do not find any significant weaknesses on the secrecy of the session keys established by IKE, we find several previously unreported weaknesses on the authentication properties of IKE.

Keywords: Security protocols, IPsec, IKE, IKEv1, IKEv2, Formal analysis, protocol interaction, multi-protocol attacks.

1 Introduction

IPsec [19] is an IETF protocol suite that provides Internet Protocol (IP) security. In particular, IPsec provides confidentiality, data integrity, access control, and data source authentication [17]. In contrast to, e. g., SSL/TLS [12], IPsec provides end-to-end security in an application-transparent way, i. e., without having to change each application separately. IPsec was developed in conjunction with IPv6, and any compliant implementation of IPv6 must support IPsec. For IPv4, IPsec is an optional extension.

The main goal of IPsec is to provide for each packet a means to guarantee origin authentication, integrity, and confidentiality. Additionally, IPsec may provide identity protection for the communicating parties. IPsec implements this by setting up for each communication a so-called *Security Association*, which essentially boils down to establishing a session key, which is used to provide the three main security properties for the subsequently transmitted messages. Perhaps surprisingly, establishing such a session key in IPsec is an involved process with a large amount of options to choose from, such as various sub-protocols, cryptographic methods, and optional fields. The protocol suite responsible for this key establishment phase is known as IKE (Internet Key Exchange). According to its specification, IKE performs "mutual authentication between two parties and establishes an IKE security association" [17].

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 315–334, 2011.

Currently, there are two versions of IKE. The original design, IKEv1 [15], was criticised for its complexity and large amount of options. Its successor, IKEv2 [17, 16], is significantly simpler and seems to provide at least the same level of security, but still offers a large amount of options.

When IKEv1 was proposed, it was analyzed by several groups, e. g., [29,30,32, 25,14]. Given that recent years have seen many new developments in methods to analyze security protocols, as well as the establishment of many strong security notions for key exchange protocols, one might expect that IKE is a prime candidate for an updated analysis. Surprisingly, only limited analysis was performed, e. g., [26] for some sub-protocols of IKEv2.

Contributions. We provide the most comprehensive formal analysis of IKE so far. This includes providing the first complete analysis of unintended interactions between pairs of protocols in IKE, and analyzing IKE with respect to formalizations of advanced cryptographic security notions [5,22]. We find a large number of previously unreported logical weaknesses. In general, these weaknesses do not lead to violations of the main security goals, but show that strong authentication properties are not always achieved. Our analysis thus provides a more precise view on the security properties provided by the IKE protocols.

We provide full details of our tests and protocol models online [10].

Overview. We give background on the IKE protocols and describe previous analyses of IKE in Section 2. We describe the setup and details of our analysis in Section 3. In Section 4 we provide an overview of the results of our analysis and describe both rediscovered and newly found weaknesses. Possible fixes and remaining issues are discussed in Section 5. Finally, we conclude in Section 6.

In the Appendix we describe the used adversary models and give the full analysis results for multi-protocol attacks.

2 Background on IKE

In this section we provide background on the IKEv1 and IKEv2 protocols and their intended security properties. Additionally we give an overview of related work, in particular previous analyses of these protocol standards.

A *Security Association* (SA) is the establishment of a secure transmission session between two network entities. This amounts to establishing a set of shared attributes, which typically include the chosen cryptographic algorithms and one or more keys for encrypting/decrypting traffic. The main purpose of IKE within IPsec is to establish a security association, more specifically the *IPsec SA*, between two authenticated IPsec peers. The IPsec SA includes traffic keys that can be used for a secure IPsec tunnel.

Notation. We denote the (plaintext) headers in each IKE message by HDR_i for some i. We write $\{m\}_{sk(X)}$ to denote the digital signature of the agent X of the message m, and $\{m\}_{pk(X)}$ for the public key encryption of m with X's public key. For a key K that is not of the form $sk(X)$ or $pk(X)$, $\{m\}_K$ denotes the symmetric encryption of m with key K. Fresh random values, usually referred to as Nonces, are denoted as N_X when created by the agent X. ID_X denotes (a representation of) the identity of agent X. The short-term Diffie-Hellman private key of agent A, also known as the ephemeral private key, is written as x_A and the corresponding ephemeral public key is written as g^{x_A}. We write $prf_K(m)$ to denote the application of the pseudorandom function prf to the message m, optionally keyed with K. CKY_A and CKY_B respectively denote the initiator's and responder's cookie, which are also included in the headers. Following the standard, we denote the proposals for negotiating security parameters by SA in the protocol descriptions.

2.1 IKE Version 1 (IKEv1)

The design of IKEv1 [15] is based on the Oakley protocol [27] and ISAKMP [24]. The protocol is essentially an authenticated key exchange protocol with additional payloads that supports multiple cryptographic algorithms and which is split into two distinct phases. In phase 1 an ISAKMP SA is established that is used in phase 2 to set up an IPsec SA.

In phase 1, the peers are authenticated and a shared secret key is established. The shared secret key is derived from the exchanged Diffie-Hellman tokens. Phase 1 can be performed in one of the following two modes: *Main Mode* (MM) and *Aggressive Mode* (AM). Main Mode provides *identity protection* (i. e., an adversary cannot determine from the messages who the participating agents are) and only exchanges security parameters with authenticated peers. The Main Mode protocol consists of six messages. In contrast, Aggressive Mode offers no identity protection and may exchange some security parameters with peers before they are authenticated, but requires only three messages.

For both MM and AM, IKEv1 supports three types of underlying key infrastructures, based on respectively symmetric cryptography, digital signatures, and public key cryptography. For public key cryptography there are two different types of protocol, one of which is supposed to replace the other but has been retained for compatibility purposes. The combination of the two modes with the four key types yields eight phase 1 protocols to establish a shared key.

Example 1 (IKEv1 Aggressive Mode with digital signatures). The IKEv1 AM protocol with digital signatures proceeds as follows:

$$\begin{aligned}
&1.\ A \rightarrow B : \mathrm{HDR}_1,\ SA, g^{x_A}, N_A, ID_A\\
&2.\ B \rightarrow A : \mathrm{HDR}_2,\ SA, g^{x_B}, N_B, ID_B,\\
&\qquad\qquad\qquad \{prf_K(g^{x_B}, g^{x_A}, CKY_B, CKY_A, ID_B)\}_{sk(B)}\\
&3.\ A \rightarrow B : \mathrm{HDR}_3,\ \{prf_K(g^{x_A}, g^{x_B}, CKY_A, CKY_B, ID_A)\}_{sk(A)}
\end{aligned}$$

where K is a shorthand for $prf_{(N_A,N_B)}(g^{x_A x_B})$.

After the completion of phase 1, the resulting shared secret is used to derive several shared secret keys. In what follows, these are denoted by SKEYID_x where x identifies the specific key.

The shared secret keys established in phase 1 are used in phase 2 to establish or update the session keys for the IPsec tunnel. This second phase is called *Quick Mode* (QM), and consists of three protocols. The first protocol provides perfect forward secrecy (PFS),i. e., revealing the long-term keys does not compromise the security of past sessions, but no identity protection. The second provides no perfect forward secrecy but is more efficient than the first, and the third provides identity protection. In IKEv1, AM and MM are always directly followed by QM.

Example 2 (IKEv1 Quick Mode without perfect forward secrecy). Let M-ID denote a message identifier and let KAB and KBA be the appropriate symmetric traffic encryption keys derived from the secret established in phase 1.

1. $A \rightarrow B : \mathrm{HDR}_1, \{\ prf_{\mathrm{SKEYID}_a}(\text{M-ID}, SA, N'_A), SA, N'_A\ \}_{KAB}$
2. $B \rightarrow A : \mathrm{HDR}_2, \{\ prf_{\mathrm{SKEYID}_a}(\text{M-ID}, N'B, SA, N'_A), SA, N'_B\ \}_{KBA}$
3. $A \rightarrow B : \mathrm{HDR}_3, \quad prf_{\mathrm{SKEYID}_a}(0, \text{M-ID}, N'_A, N'_B)$

The resulting new keying material is defined as $prf_{\mathrm{SKEYID_d}}(\text{protocol}, SPI, N'_A, N'_B)$, where protocol and SPI are taken from the ISAKMP SA proposals.

2.2 IKE Version 2 (IKEv2)

IKEv2 [17, 16] was designed to add new features, address some weaknesses in IKEv1 and, at the same time, provide a cleaner design. In IKEv2 there are only three sub-protocols in phase 1. These are based on digital signatures, MAC's, and EAP (Extensible Authentication Protocol). Similar to IKEv1, Diffie-Hellman exponents and nonces are exchanged and used to compute several shared secret keys. For example, the initiator i will use SK_{ei} (for encryption), SK_{ai} (for authentication), and SK_{pi} (for the authentication payload). Similarly, the responder r will use SK_{er}, SK_{ar}, and SK_{pr}.

The phase 1 protocols establish an IKE SA (similar to IKEv1's ISAKMP SA) and a first child SA (similar to IKEv1's IPsec SA). Hence, contrary to IKEv1, an SA that can be used for IPsec (the child SA) is directly available after the first phase. The specification of the first phase protocols contains optional fields, and allows for initiators and responders to use different authentication mechanisms.

Example 3 (IKEv2 phase 1 with digital signatures). Let SAi1 and SAr2 denote the supported cryptographic algorithms and let SAi2 and SAr2 denote IPsec SA proposals. In this example, we write $\{m\}_{SK_x}$ to denote that m is integrity protected and encrypted using SK_{ax} and SK_{ex}. TSi and TSr denote traffic selectors and are not directly relevant for our analysis. Similarly, $SPIi$ and $SPIr$ denote Security Parameter Indexes. Furthermore, we define

$$\begin{aligned}AUTHi &= \{\mathrm{SAi1}, g^{x_A}, N_A, N_B, prf_{SK_{pi}}(ID_A)\}_{sk(A)}\\ AUTHr &= \{\mathrm{SAr1}, g^{x_B}, N_B, N_A, prf_{SK_{pr}}(ID_B)\}_{sk(B)}\end{aligned}$$

Then, the protocol proceeds as follows:

1. $A \to B : \mathrm{HDR}_1, \mathrm{SAi1}, g^{x_A}, N_A$
2. $B \to A : \mathrm{HDR}_2, \mathrm{SAr1}, g^{x_B}, N_B$
3. $A \to B : \mathrm{HDR}_3, \{ID_A, ID_B, AUTHi, \mathrm{SAi2}, TSi, TSr\}_{SK}$
4. $B \to A : \mathrm{HDR}_4, \{ID_B, AUTHr, \mathrm{SAr2}, TSi, TSr\}_{SK}$

The second phase in IKEv2 is known as *Child Mode*. The purpose of Child Mode is to re-key previous (IKE or child) SA's or to establish additional child SA's. For Child Mode, there are only two protocols, one of which provides Perfect Forward Secrecy. These are similar to their IKEv1 phase 2 counterparts, but consist only of two messages. For further details we refer to [17].

2.3 Intended Security Properties

The IKE standards mention the following intended properties of the protocols:

1. To obtain authenticated keying material for use with ISAKMP and for other security associations [15, p. 1].
2. To achieve the security goals provided by Oakley [27]: perfect forward secrecy for keys, identity protection, and authentication [15, p. 2].
3. To perform mutual authentication [17, p. 5].

The standards do not contain more detailed descriptions of these properties.

2.4 Previous Analyses of IKE

IKEv1 has been analyzed before by several authors. In 1999, Meadows performed a large case-study of IKEv1 using the NRL protocol analyzer [25], a formal protocol analysis tool. Her analysis uncovered many subtle properties and weaknesses at the logical level. In 2000, Perlman and Kaufman performed a manual analysis of IKEv1 [29,30]. Their analysis covered not only the logical level, but also many different other aspects of the protocol, leading to a significant critique of the standard. Many of their suggestions were included in the design of the IKEv2 standard. Zhou analyzed further issues of the IKEv1 standard [32] and suggests amendments for the weaknesses he finds. In 2001 and 2002, Canetti and Krawczyk analyzed selected sub-protocols of IKE in the cryptographic setting [6,5], e. g., showing that a variant of IKE Main Mode with signatures satisfies a form of cryptographic key-exchange security.

In contrast, there has only been very limited analysis of IKEv2. In 2003, three sub-protocols of IKEv2 were formally analyzed in the context of the AVISPA project [1,26]. They found that an IKEv2 sub-protocol suffers from a weakness that was already pointed out on a related IKEv1 protocol by Meadows.

3 Formal Analysis of IKEv1 and IKEv2

In this section we describe the setup of our analysis. The results will be described in Section 4. Similar to the approach taken by Meadows for her analysis of IKEv1 [25], we follow the line of work by Dolev and Yao [13]. Hence we focus on the detection of logical vulnerabilities of protocols, and assume that cryptography is perfect in the sense that, for example, the adversary learns nothing from an encrypted message unless he knows the decryption key. This can be seen as a separation of concerns: we rely on the stated properties of the used cryptographic algorithms. A second assumption is that the adversary has full control over the network, and can intercept or modify all messages, or inject his own.

Our analysis covers significantly more security aspects than the earlier formal analyses of IKE. Along the lines of the security notions for key exchange first explored by Bellare and Rogaway [3], we also consider various advanced security properties, such as perfect forward secrecy, key compromise impersonation, and known-key attacks. Here we follow the formalizations by Basin and Cremers [2]. Additionally, we consider interactions between sub-protocols, which are an instance of multi-protocol attacks [18,11].

Formal analysis tool and extensions. In our analysis we used Scyther [8], a formal protocol analysis tool. In particular, we exploited two of its features: analyzing protocol properties with respect to various adversary models [2], and support for multi-protocol analysis [11].

For the analysis of IKE, we extended the Scyther tool with two features. First, we provided support for checking *agreement* as defined by Lowe [23]. Scyther's built-in notions of authentication, (weak) aliveness and synchronisation[1], were respectively too weak and too strong in the context of IKE. Second, because the scope of our analysis is currently not feasible on standard computing hardware, we developed infrastructure for performing large-scale *parallel* analysis for multi-processor computing clusters. We will return to this issue in Section 3.

Protocol models. A critical step in the formal analysis consists of providing abstract models of the protocols under investigation, such that they can be analyzed within the formal framework underlying the tool [9]. The abstract protocol models should include all security-relevant information. It is critical to find a balance here between precision (i. e., complexity) of the models and feasibility of the analysis.

For our models, we abstracted the security association proposals into arbitrary choices (but checking that the recipient and sender agree on the contents of the proposal). Similarly, we abstracted from the public key certificates and assume that they have been pre-distributed in a secure manner, which is justifiable as ‘our models do not consider concepts such as key revocation.

Within the formal framework, the commutativity property of the Diffie-Hellman exchange ($(g^a)^b = (g^b)^a$) currently cannot be modeled precisely,

[1] This can be considered similar to the notion of “matching histories”, and requires messages to have been sent exactly as received, as well as in the correct order.

Table 1. Overview of IKEv1 protocol models. The bottom four protocols are not part of the standard, but are improvements suggested in [30]

name	mode	encrypt last message	separate encryptions
ikev1-sig-m	MM		
ikev1-sig-a1	AM	N	
ikev1-sig-a2	AM	Y	
ikev1-pk-m	MM	N	Y
ikev1-pk-m2	MM	N	N
ikev1-pk-a1	AM	N	Y
ikev1-pk-a12	AM	Y	Y
ikev1-pk-a2	AM	N	N
ikev1-pk-a22	AM	Y	N
ikev1-pk2-m	MM	N	Y
ikev1-pk2-m2	MM	N	N
ikev1-pk2-a	AM	N	Y
ikev1-pk2-a2	AM	N	N
ikev1-psk-m	MM		
ikev1-psk-a	AM		
ikev1-quick	QM		
ikev1-quick-noid	QM		
ikev1-quick-nopfs	QM		
ikev1-sig-a-perlman1	AM	N	
ikev1-sig-a-perlman2	AM	Y	
ikev1-sig-m-perlman	MM		
ikev1-psk-m-perlman	MM		

Table 2. Overview of IKEv2 protocol models

name	mode	optional identity
ikev2-sig	SIG	Y
ikev2-sig2	SIG	N
ikev2-mac	MAC	Y
ikev2-mac2	MAC	N
ikev2-eap	EAP	Y
ikev2-eap2	EAP	N
ikev2-sigtomac	SM	Y
ikev2-sigtomac2	SM	N
ikev2-mactosig	SM	Y
ikev2-mactosig2	SM	N
ikev2-child	C	
ikev2-child-nopfs	C	

and we therefore underapproximate this property by giving the adversary the capability of rewriting such exponentiations at fixed subterm positions, which are derived from the protocol specification. Full details can be found in the protocol models. Our underapproximation ensures soundness of the attacks that we find.

In Table 1 we list the models of the sub-protocols that we consider for IKEv1. The first column shows the names which are meant to be self-explanatory as far as possible, including the underlying cryptographic mode in the name (hence pk and pk2 for the two public-key variants, and sig and psk for signatures and pre-shared keys, respectively). The second column shows the mode. For phase 1 we have “MM” for Main Mode and “AM” for Aggressive Mode. For phase 2 we have “QM” for Quick Mode. The third column, “encrypt last message”, marks protocol variants in which the last message is encrypted by the session key. The fourth column, “separate encryptions”, refers to an ambiguity in the specification: in one message, it is required that a nonce and an ID are encrypted. This can be interpreted in two ways: first encrypt each separately, and then concatenate the result, or alternatively, first concatenate the nonce and ID and then encrypt the result. For our analysis, we have modeled both interpretations. For Quick Mode, there are three variants. The first two are Diffie-Hellman based exchanges with and without the optional identity fields. The third variant uses plain nonces instead of Diffie-Hellman and can be used when perfect forward secrecy (PFS) is not required. Besides the protocol models described in the IKEv1 and IKEv2 standards, we also modeled four variants suggested by Perlman and Kaufman.

Similarly, Table 2 lists the sub-protocols modeled for IKEv2. The main modes here are "SIG", "MAC", and "EAP", denoting digital signatures, MACs, and Extensible Authentication Protocol (EAP), respectively. Additionally, "SM" denotes the variants in which one agent uses a different mode than the peer within the same sub-protocol. Finally, "C" denotes Child Mode, which is IKEv2's variant of the re-keying phase. The third column marks the variants in which optional identity fields are omitted in the specification.

Our full protocol models are publicly available at [10].

Security properties. As stated before, the IKE standards do not provide detailed descriptions of the intended properties. One of the contributions of our work is therefore to establish more precisely which properties are guaranteed by the protocols. In our analysis, we consider the following *basic security properties.*

Aliveness. If an agent a executes a role of the protocol, thinking he ran it with b, then b has indeed performed an action (and is therefore "alive"). This is a very weak authentication property.

Weak agreement. Aliveness, where additionally b assumes that he is communicating with a, and hence they both "agree" on the agents involved in the protocol.

Agreement (on a list of terms S). This implies weak agreement and additionally, b is indeed performing the role that a assumes. Finally, a and b agree on the values in S, e. g., they agree on the computed session key.

Secrecy (of a term t). The term t, e. g., a computed session key, will not become known to the adversary.

More formally, for the first three properties, we follow the definitions given in [23], and for the final property, we follow [2]. The supposed security properties are specified in the protocol description by means of *claim events*, such that they can be analyzed by the Scyther tool.

Following [2], we combine basic security properties with *adversary models.* We consider a total of 10 adversary models, which include the standard Dolev-Yao model, as well as abstract versions of the adversary capabilities considered in the CK model [5] and the eCK model [22], which are commonly used security models for proving the security of authenticated key exchange protocols. We informally describe these models in Appendix A.

Combining the above basic security properties with the appropriate adversary model, we analyze the protocols with respect to the following properties.

(Perfect) Forward Secrecy. We consider secrecy of a term (typically a session key) occurring in a session s whilst allowing the adversary to compromise the long-term keys of all agents after s has been completed. Thus, the compromise of the long-term keys should not lead to the compromise of session keys (or any session-specific secret) from earlier sessions.

Weak Perfect Forward Secrecy. Similar to Perfect Forward Secrecy above, but slightly weaker: if he did not actively interfere in the session s, then he may compromise the long-term keys of agents. This notion was introduced in [20] as a variant of Perfect Forward Secrecy suitable for a class of implicitly authenticated key exchange protocols, such as HMQV.

(resilience to) Key Compromise Impersonation. If the adversary is able to compromise the long-term keys of an agent a, he should not be able to impersonate as an arbitrary agent to a. This can be modeled by analyzing authentication properties in the presence of an adversary capable of compromising the actor's long-term keys.

(resilience to) known session key attacks. If the adversary learns a session key, this should not allow him to attack other sessions. Here we follow Bellare and Rogaway [3] and analyze secrecy of a particular session key in the presence of an adversary that can compromise all session keys of *non-matching sessions*. We will return to the concept of matching sessions when discussing the results.

Using the same mechanism, i. e., combining basic security properties with adversary models, allows us to analyze protocols with respect to abstract versions of cryptographic security notions, such as the previously mentioned CK and eCK models. Full details and formal specifications of the protocol execution model and the various adversary capabilities can be found in [2].

Multi-protocol attacks. We also considered possible protocol interactions in the context of a standard Dolev-Yao adversary (i. e., with static corruption). When (sub-)protocols that are executed in parallel share the same long-term keys, the interaction between the protocols can enable *multi-protocol attacks* [18,11]. As already pointed out in [25], given the many sub-protocols in IKE, unforeseen protocol interactions cannot be excluded. However, analysing all possible interactions is significantly harder than their individual analysis, and therefore not all possible interactions were considered by Meadows in [25].

In our analysis we consider all possible interactions between pairs of subprotocols in IKEv1 and IKEv2. In theory, attacks could be possible that require interactions among three or more protocols [11] but this is currently infeasible to analyze in our setup.

Analysis hardware. Our analysis would not have been feasible without using a significant amount of hardware. The enabling factor for our analysis was the high-performance cluster of ETH Zurich, called *Brutus* [31]. Brutus is a heterogeneous system with several types of compute nodes. Currently, Brutus has a total of 9912 processor cores in 1108 compute nodes. The majority of the compute nodes are four quad-core AMD Opteron 8380 CPUs with 32 GB of RAM. In our tests we used a maximum of 128 processor cores in parallel, due to default usage limits.

Our analysis mostly consisted of a large number of relatively small tasks, such as analyzing a single protocol property with respect to a particular adversary model, and hence could be easily parallelized. The set of tests for analyzing all protocols with respect to all adversary models, excluding the multi-protocol analysis, took about a day of computation on the cluster. The multi-protocol analysis for all two-protocol combinations with respect to a single adversary model took just over two days. To put these numbers into perspective, on a single desktop machine these computations would have taken at least two orders of magnitude more time, thus requiring about a year of computation time. Of course, in practice, the situation is worse: in some cases the test results revealed modeling errors, whose fixing required partial recomputation of the results. Thus, we used the Brutus cluster therefore much longer than three days, and analysis using a single machine would not have been feasible.

4 Results

Our analysis automatically rediscovers known weaknesses but also discovers many new weaknesses, as shown in Tables 3 and 4. We briefly discuss the rediscovered weaknesses, before explaining in detail the newly discovered weaknesses.

4.1 Automatically Rediscovered Weaknesses

(K1) Reflection attack on IKEv1 Main Mode with digital signatures or pre-shared keys. In [14], Ferguson and Schneier report a reflection attack on IKEv1 Main Mode when used with digital signatures or pre-shared keys. The

Table 3. Overview of attacks on properties of IKEv1 sub-protocols in the presence of a standard Dolev-Yao adversary. A dagger (†) denotes that the attacks require self-communication.

Protocol	Roles	Violated properties	Classification
ikev1-pk-m2	I	Agreement, Weakagree (†)	N1
ikev1-pk2-m2	I	Agreement, Weakagree (†)	N1
ikev1-psk-m	I	Agreement, Weakagree (†)	K1
ikev1-psk-m-perlman	I	Agreement, Weakagree (†)	K1
ikev1-quick-noid	I,R	Aliveness, Agreement, Weakagree	K2
ikev1-quick-nopfs	I,R	Aliveness, Agreement, Weakagree	K2
ikev1-sig-a-perlman1	I	Agreement, Weakagree	N2
ikev1-sig-a-perlman2	I	Agreement, Weakagree	N2
ikev1-sig-a1	I	Agreement, Weakagree	N2
ikev1-sig-a2	I	Agreement, Weakagree	N2
ikev1-sig-m	I	Agreement, Weakagree (†)	K1
ikev1-sig-m	R	Agreement, Weakagree	N3
ikev1-sig-m-perlman	I	Agreement, Weakagree	K1

Table 4. Overview of attacks on properties of IKEv2 sub-protocols with respect to a Dolev-Yao style adversary (INT)

Protocol	Roles	Violated properties	Classification
ikev2-child	I,R	Aliveness, Agreement, Weakagree	N4
ikev2-child-nopfs	I,R	Aliveness, Agreement, Weakagree	N4
ikev2-sig2	R	Agreement, Weakagree	K3
ikev2-sigtomac2	R	Agreement, Weakagree	K3

attack is a simple reflection that requires that an initiator may accept her own identity as the peer. We will refer to the case in which an agent accepts her own identity as the peer as the *self-communication* scenario. In this case, the authenticators in the protocol become equal, and the adversary simply sends all messages coming from the initiator back to her, unchanged. In this case weak agreement fails, because there is no agent performing the responder role. Secrecy of the key is still guaranteed. Such a self-communication scenario may be relevant under some circumstances, as argued, e. g., in [4]. However, it is clear that self-communication is not prohibited by the specification of IKEv1 and should therefore be considered a legitimate use. When self-communication is not possible, e. g., because identity-inequality checks are implemented for both roles, the attack pointed out by Ferguson and Schneier is no longer possible.

(K2) Reflection Attack on IKEv1 Quick Mode. Meadows reported a reflection attack on IKEv1 Quick Mode [25]. Because of the symmetry of the messages, and the possibility of leaving out some optional identities, the recipients of messages cannot determine directly whether they sent these messages themselves (possibly in other sessions). This leads to a straightforward reflection attack, and even allowing the reflection of messages to the responder role. Meadows observed that this may be prevented by other details of the implementation, but it is clear that there is no mechanism at the logical level to prevent reflections, and hence this may be an issue in some implementations.

(K3) Weak Authentication for IKEv2 with Digital Signatures. The IKEv2 digital signature mode allows identities to be omitted. As a result, an agent may successfully complete the responder role while her peer believes that he is in a partial run, talking to a different agent [26]. This constitutes a violation of agreement on the agent identities. Moedersheim et al. report this as a violation of "penultimate authentication" (We will return to this property in Section 5). We observe that this violates strong authentication for the responder. In practice this means that an IKEv2 responder Bob may accept an IPsec SA as valid for Alice, even though only weak authentication is provided: it may not be the case that Alice intended to talk to Bob. However, subsequent messages received over the IPsec tunnel are guaranteed to provide strong authentication. We will see an example of a similar weakness below (N2).

4.2 Previously Unreported Weaknesses

(N1) Reflection Attack on IKEv1 Main Mode with Public Key Encryption. Our analysis reveals that the reflection attack (K1) reported by Ferguson and Schneier, is also possible for the public key variants. Similar to their attacks, the self-communication scenario is required for the attacks to work.

(N2) Authentication Failure on IKEv1 Aggressive Mode with Digital Signatures. For IKEv1 Aggressive Mode with digital signatures, as described in Example 1, we find that (weak) agreement is not guaranteed for the initiator. The following scenario can occur:

1. A generates x_A and N_A and sends (for B): $\mathrm{HDR}_1, SA, g^{x_A}, N_A, ID_A$.
2. The adversary intercepts this message and changes ID_A to ID_C for an arbitrary agent C. The adversary sends the result to B: $\mathrm{HDR}_1, SA, g^{x_A}, N_A, ID_C$.
3. B accepts the message, and assumes that g^{x_A} and N_A were generated by C.
4. B generates x_B and N_B and computes the key $K = \mathit{prf}_{(N_A,N_B)}((g^{x_A})^{x_B})$.
5. B sends the following message for C:
   $\mathrm{HDR}_2, SA, g^{x_B}, N_B, ID_B, \{\mathit{prf}_K(g^{x_B}, g^{x_A}, CKY_B, CKY_A, ID_B)\}_{sk(B)}$.
6. The adversary intercepts this message and sends it to A, who accepts it, computing the same key K.
7. A sends (for B): $\mathrm{HDR}_3, \{\mathit{prf}_K(g^{x_A}, g^{x_B}, CKY_A, CKY_B, ID_A)\}_{sk(A)}$.
   Note that this message is not what B expects, and will be rejected by B.
8. A successfully completes her part of the protocol.

Consequently, agreement on the participants is not provided by this protocol: after A finishes her initiator role with B, B was indeed running the responder role, but B was under the assumption he was running the protocol with C. When A finishes her role, B may still be waiting for the final message, or may have aborted. Clearly, B will not accept the final message because he is expecting the message to be signed by C. However, A cannot detect whether B has accepted the final message unless additional messages are exchanged.

In practice, this means that the established ISAKMP SA provides no guarantees about the intended partner of the peer. In the context of IPsec, A will next try to establish an IPsec SA using QM, which will fail, because B did not accept the ISAKMP SA. Hence this does not result in a weakness at the IPsec level. However, the IKEv1 specification [15] explicitly allows ISAKMP services to directly use the ISAKMP SA, without establishing an IPsec SA. For such services, only a weak form of authentication is guaranteed, as in (K3).

The underlying problem is that the responder's signature only includes the name of the sender (ID_B), but not that of the (supposed) peer. The signature therefore does not provide any information about who the responder believes he is communicating with. This issue can be prevented by including the name of the intended recipient inside the signature. In the above scenario, B would have inserted ID_C in step 5, and A would not have accepted the message in step 6.

(N3) Authentication Failure on IKEv1 Main Mode with Digital Signatures That Does not Require Self-Communication. Ferguson and Schneier pointed out that the IKEv1 Main Mode with signatures is subject to a reflection attack in self-communication scenarios (K1). We find that there is another problem for the responder in this protocol, which does not involve self-communication. Critically, the third message does not include information on the initiator's intended partner. Thus, when receiving the third message, the responder cannot be sure that the message was meant for him. The attack and its consequences are similar to (N2).

(N4) Reflection Attack on IKEv2 Phase 2 Exchange. The reflection attack that was noted before for IKEv1 Quick Mode (K2) is also possible on the related IKEv2 phase. This is surprising, because it could have been easily fixed by breaking the symmetry of the messages, e. g., by including distinct constants and checking their presence when parsing incoming messages.

Multi-protocol analysis. Our analysis of the composition of all pairs of IKE subprotocols showed that there are no problematic interferences between them. This was not a priori given as the protocols do not include explicit tagging to distinguish between the messages of different sub-protocols. The only interaction was between the "mixed" IKEv2 modes, where the sender of the last message can not be sure whether his peer is expecting (or able to parse) an authentication message of the type that he sends. However, these minor interactions clearly show that the various sub-protocols, whose joint state is defined as their long-term private keys, cannot be considered to be *universally composable* in the sense of, e. g., [7]. The full results can be found in Table 6 in Appendix B.

Compromise analysis. We analyzed each protocol with respect to the adversary compromise models described in Appendix A. The results are shown in Table 5. In the top row, the abbreviated names of the adversary models are listed.

We highlight some results. The protocol variants establish perfect forward secrecy (the **S**ecrecy sub-column in the "AF" column, where AF is the adversary model in which the adversary can compromise all long-term keys after the session) except for those without Diffie-Hellman. Interestingly, we found that some protocols revealed no attacks in the CK adversary model (which models the adversary from Canetti and Krawczyk's model for secure key exchange [5]), even when queries such as "session-state reveal" are considered, which suggests that computational proofs in CK for these protocols may be feasible. None of the considered protocols is correct in the eCK adversary model (represented by combining the columns eCK-1 and eCK-2) which also stems from the field of secure key exchange [22]. This result confirms that the IKE protocols do not offer any protection against the compromise of ephemeral keys (the short-term private Diffie-Hellman exponents). Perhaps surprisingly, many protocols seem to be vulnerable to known session-key attacks in the style of Bellare-Rogaway [3]. However, closer examination of the attacks shows that this is mainly due to a technicality. In these known-key attacks, the adversary does not compute the session key from another (related) session key, but rather reveals the same session

Table 5. Analysis results for IKEv1 and IKEv2 subprotocols with respect to different adversary models, using notation from [2] (Appendix A). Subcolumns indicate the property considered: **S**ecrecy, **A**liveness, and **W**eak agreement. Property violations that were previously reported are marked with ∘, and previously unreported violations are marked with ⋆.

	EXT			INT			CA			AFC			AF			BR			CKw			CK			eCK-1			eCK-2		
	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W
Signature authenticators																														
ikev1-sig-a1			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev1-sig-a2			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev1-sig-a-perlman1			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev1-sig-a-perlman2			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev1-sig-m			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev1-sig-m-perlman			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆		⋆	⋆		∘
ikev2-sig																									⋆		⋆			
ikev2-sig2			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	⋆		∘	⋆		⋆	⋆		∘
Public key authenticators																														
ikev1-pk-a1			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆	⋆	⋆	⋆		∘
ikev1-pk-a12			∘			∘			∘			∘			∘	⋆		∘	⋆		∘	∘		∘	⋆	⋆	⋆	⋆		∘
ikev1-pk-a2																			⋆			∘			⋆	⋆	⋆			
ikev1-pk-a22																			⋆			∘			⋆	⋆	⋆			
ikev1-pk2-a																									⋆	⋆	⋆			
ikev1-pk2-a2																									⋆	⋆	⋆			
ikev1-pk-m			⋆			⋆			⋆			⋆			⋆	⋆		⋆	⋆		⋆	∘		⋆	⋆	⋆	⋆	⋆		⋆
ikev1-pk-m2			⋆			⋆			⋆			⋆			⋆	⋆		⋆	⋆		⋆	∘		⋆	⋆	⋆	⋆	⋆		⋆
ikev1-pk2-m			∘			∘			∘			∘			∘			∘			∘			∘	⋆	⋆	⋆			∘
ikev1-pk2-m2			∘			∘			∘			∘			∘			∘			∘			∘	⋆	⋆	⋆			∘
Pre-shared key authenticators																														
ikev1-psk-a							⋆	⋆	⋆										⋆	⋆	⋆							⋆	⋆	⋆
ikev1-psk-m			∘			∘	⋆	⋆	⋆			∘			∘			∘	⋆	⋆	⋆			∘			∘	⋆	⋆	⋆
ikev1-psk-m-perlman			∘			∘	⋆	⋆	⋆			∘			∘			∘	⋆	⋆	⋆			∘			∘	⋆	⋆	⋆
ikev2-mac							⋆	⋆	⋆										⋆	⋆	⋆				⋆			⋆	⋆	⋆
ikev2-mac2							⋆	⋆	⋆										⋆	⋆	⋆				⋆			⋆	⋆	⋆
Other authenticators																														
ikev2-eap																									⋆	⋆	⋆			
ikev2-eap2																									⋆	⋆	⋆			
ikev2-mactosig							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-mactosig2							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-sigtomac							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-sigtomac2			∘			∘	⋆	⋆	⋆			∘			∘	⋆		∘	⋆	⋆	⋆	⋆		∘	⋆		⋆	⋆	⋆	⋆
Phase 2 subprotocols																														
ikev1-quick							⋆	⋆	⋆										⋆	⋆	⋆							⋆	⋆	⋆
ikev1-quick-nopfs		∘	∘		∘	∘	⋆	⋆	⋆		∘	∘	⋆	∘	∘		∘	∘	⋆	⋆	⋆	⋆	∘	∘		∘	∘	⋆	∘	∘
ikev1-quick-noid		∘	∘		∘	∘	⋆	⋆	⋆		∘	∘		∘	∘		∘	∘	⋆	⋆	⋆		∘	∘		∘	∘	⋆	∘	∘
ikev2-child		⋆	⋆		⋆	⋆	⋆	⋆	⋆		⋆	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆
ikev2-child-nopfs		⋆	⋆		⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆		⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆		⋆	⋆	⋆	⋆	⋆

key at a so-called *non-matching session.* In our models, following [3], matching sessions are defined as sessions whose exchanged messages are matching. Consequently, known-key attacks can occur if the adversary can make an agent accept the session key while slightly modifying some messages parts. Even when these modifications do not have a practical impact on security, they lead to known-key attacks in the formal model. The relevance of such attacks is mostly theoretical; at worst, they suggest unknown-key share attacks.

Reproducing the results of the analysis. The Scyther tool, the input files, and the test scripts we used are publicly available from [10]. Although significant computational effort was used to obtain the full results, only modest computational effort is required to reproduce individual attacks. For example, a single standard current-generation PC (running, e. g., Linux) would suffice to reproduce any one of the attacks described here within minutes.

5 Discussion

Our analysis reveals that many IKEv1 and IKEv2 sub-protocols do not satisfy strong security properties, such as agreement. The first underlying problem is that insufficient information about the involved agents is included in the cryptographically-protected parts of the messages. This occurs even in protocols where identity protection is not an issue. The second underlying problem is a lack of distinguishing tags in the protocols, enabling reflection attacks.

Along the same lines, though our analysis did not reveal any problematic interaction among the various sub-protocols, the lack of unique identification within the various encrypted or signed payloads remains an issue. For example, the IKE version number is only included in the header of messages and not included (or checked) within any cryptographically protected message. As a result, an active adversary can arbitrarily change version numbers and reroute encrypted content from one protocol version into another. This significantly complicates the problem of updating broken protocol versions, because the presence of the old versions may still break the security of new versions, as pointed out in [11].

In cryptographic definitions of security, such as [3], the adversary can compromise session keys of any non-matching session. As we have seen, many of the IKE variants are vulnerable to these attacks. These attacks may seem artificial; clearly, if the adversary learns the relevant session key (from whichever session) then security is lost. However, there is an important issue underlying such attacks: session keys should be computed in as few contexts as possible. The threat of computing session keys in non-matching sessions is that it may give the adversary unnecessarily more options to compromise the session key, e. g., because he can only compromise the memory of particular sessions or at certain points in time. Protocols can be designed such that the least possible amount of sessions compute the same session key, e. g., only in matching sessions, thereby minimizing the window of opportunity for compromise.

The IKEv1 attack on authentication (N2) was not reported by previous formal analysis. This is remarkable because the attack is within the scope of the methods that were used by, e. g., Meadows [25]. The reason that this attack was missed is that the property that was considered by Meadows is a much weaker form of authentication: "The receiver A of the final message should not have accepted security association SA as good for communication with B without B having itself accepted SA." [25]. It is clear from this formulation that there is no requirement stated that A's acceptance of SA for communication with B means that B accepted SA *for communication with* A. Instead, Meadows observed that the protocol does not satisfy "penultimate authentication". In our model, checking for penultimate authentication corresponds to analyzing whether B's security guarantees also hold before the last message of the protocol is received and verified. We did not consider this property here, because we assume implementations successfully complete individual sub-protocols before assuming any security properties.

6 Conclusions

We have provided the most comprehensive formal analysis of IKEv1 and IKEv2 so far. Our analysis goes far beyond what was analyzed before using formal methods and we leveraged massive parallelization to obtain our results.

Our analysis did not reveal any new critical weaknesses in the IKE protocols with respect to the secrecy of the established keys. However, we discover several new weaknesses. These mainly revolve around the failure of the IKE protocols themselves to provide mutual authentication, even though the established session keys can be considered "safe" in the sense that they will be known, at most, to the intended partners. Hence, when using IKE, strong mutual authentication cannot be taken for granted and in fact requires an additional step where the session key is used, in such a way that reflection attacks (on exchanges protected by the session key) are prevented.

As may have been expected, IKEv2 shows significantly less weaknesses than IKEv1, and should therefore certainly be preferred over IKEv1.

The remaining weakness in IKEv2 phase 1, in which only weak authentication is achieved, in practice means that no strong authentication guarantees are provided before a message is received over the IPsec tunnel. This could be solved by either documenting the particular limitations of the signature-based Aggressive Mode, or by reintroducing the required agent field in the specification.

For IKEv2 phase 2, the practical implication of the reflection attacks is that after a phase 2 exchange, recent aliveness of the peer is not guaranteed. As above, this is resolved once a message is subsequently received over the tunnel. This could be mentioned in the specification, so that developers do not make unfounded assumptions (e. g., aliveness of the peer) after a phase 2 exchange.

With respect to future work, there are several properties that are currently out of scope of our methods. Two obvious candidates for future work are *identity protection* and resilience against *denial-of-service* attacks. Additionally, we note that in this analysis we focused on attack detection. Another possible goal is formal verification, i. e., showing the *absence* of flaws at the logical level. This seems out of scope for the current state-of-the-art in formal analysis. Along the same lines it would be desirable to prove more precise properties of the individual sub-protocols, e. g., as in [28].

Acknowledgements. The author is grateful to Adrian Kyburz, whose IKE models and initial analysis [21] provided the starting point for this work.

References

1. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
2. Basin, D., Cremers, C.J.F.: Modeling and Analyzing Security in the Presence of Compromising Adversaries. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 340–356. Springer, Heidelberg (2010)
3. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
4. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment, 1st edn. Springer, Heidelberg (2003)
5. Canetti, R., Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
6. Canetti, R., Krawczyk, H.: Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (2002)
7. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
8. Cremers, C.J.F.: The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
9. Cremers, C.J.F., Mauw, S.: Operational Semantics of Security Protocols. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools, International Workshop. LNCS, vol. 3466, pp. 66–89. Springer, Heidelberg (2005)
10. Cremers, C., Kyburz, A.: IKEv1 and IKEv2 protocol models for the Scyther tool (2011), `http://people.inf.ethz.ch/cremersc/scyther/ike`
11. Cremers, C.: Feasibility of multi-protocol attacks. In: Proc. of The First International Conference on Availability, Reliability and Security (ARES), pp. 287–294. IEEE Computer Society Press, Vienna (2006)
12. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (August 2008), `http://www.ietf.org/rfc/rfc5246.txt` (updated by RFCs 5746, 5878)
13. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(12), 198–208 (1983)
14. Ferguson, N., Schneier, B.: A Cryptographic Evaluation of IPsec. Tech. rep., Counterpane Internet Security, Inc. (2000)
15. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard) (November 1998), `http://www.ietf.org/rfc/rfc2409.txt` (obsoleted by RFC 4306, updated by RFC 4109)
16. Harkins, D., Kaufman, C., Kent, S., Kivinen, T., Perlman, R.: Design Rationale for IKEv2. IETF Internet Draft (expired) (February 2002), `http://www.ietf.org/proceedings/54/I-D/draft-ietf-ipsec-ikev2-rationale-00.txt`
17. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P.: RFC 5996: Internet Key Exchange Protocol Version 2 (IKEv2) (September 2010), `http://www.rfc-editor.org/info/rfc5996`

18. Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. In: Christianson, B., Lomas, M. (eds.) Proc. 5th International Workshop on Security Protocols 1997. LNCS, vol. 1361, pp. 91–104. Springer, Heidelberg (1998)
19. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard) (December 2005), http://www.ietf.org/rfc/rfc4301.txt
20. Krawczyk, H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
21. Kyburz, A.: An automated formal analysis of the security of the Internet Key Exchange (IKE) protocol in the presence of compromising adversaries. Master's thesis, ETH Zurich (November 2010)
22. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger Security of Authenticated Key Exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
23. Lowe, G.: A Hierarchy of Authentication Specifications. In: Proc. 10th IEEE Computer Security Foundations Workshop (CSFW), pp. 31–43. IEEE Computer Society Press, Los Alamitos (1997)
24. Maughan, D., Schertler, M., Schneider, M., Turner, J.: Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408 (Proposed Standard) (November 1998), http://www.ietf.org/rfc/rfc2408.txt (obsoleted by RFC 4306)
25. Meadows, C.: Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 216–231 (1999)
26. Moedersheim, S., Drielsma, P.H., et al.: AVISPA Project Deliverable D6.2: Specification of the Problems in the High-Level Specification Language (2003), http://www.avispa-project.org/
27. Orman, H.: The Oakley Key Determination Protocol. Tech. rep., University of Arizona, Tucson, AZ, USA (1997); also described in RFC 2412
28. Paterson, K.G., Watson, G.J.: Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 345–361. Springer, Heidelberg (2010)
29. Perlman, R., Kaufman, C.: Key exchange in IPSec: analysis of IKE. IEEE Internet Computing 4(6), 50–56 (2000)
30. Perlman, R.J., Kaufman, C.: Analysis of the IPSec Key Exchange standard. In: 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001), Cambridge, MA, USA, June 20-22, pp. 150–156. IEEE Computer Society, Los Alamitos (2001)
31. Swiss National Computing Centre: Brutus cluster, http://www.cscs.ch/index.php
32. Zhou, J.: Further analysis of the Internet Key Exchange protocol. Computer Communications 23(17), 1606–1612 (2000)

A Adversary Models

In this section we briefly describe the adversary models considered in this analysis as listed in Table 5. Their formal definitions can be found in [2].

EXT. In this model, the adversary has full control over the network but is an outsider (EXTernal): he does not have an identity within the system and he does not know any long-term private keys.

INT. The INT model (INTernal) models the standard Dolev-Yao model, as used, e. g., by Lowe for his analysis of the Needham-Schroeder protocol. The adversary has full control over the network, but he additionally has access to some long-term private keys. This models either a malicious insider or an adversary that has compromised some agents. For example, this allows the adversary to perform Lowe's man-in-the-middle attack on the Needham-Schroeder protocol.

CA. An agent A can authenticate another agent B if B knows some secrets that are not known to the adversary, such as B's long-term private keys. However, it is not necessary that A also knows some secrets. Some protocols allow for authentication even when the long-term keys of the authenticating agent (i. e., the verifier) are known to the adversary. The CA model (Compromised Actor) gives the adversary full network control but also the ability to learn the long-term keys of the authenticating agent. This adversary model is used to model Key Compromise Impersonation (KCI) attacks.

AF and AFC. The AF and AFC adversary models correspond to an adversary that is capable of learning all long-term private keys of the agents after (AFter) a session, and are used to analyze Perfect Forward Secrecy and weak Perfect Forward Secrecy. Compared to the AF model, the additional restriction of the AFC model (AFter Correct) is that the adversary can only learn the keys after sessions in which he did not actively interfere, i. e., in which he was passive, which is used to model weak Perfect Forward Secrecy.

BR. The BR adversary model (Bellare-Rogaway) corresponds to an INT adversary that can additionally compromise session keys of other (non-matching) sessions, thereby modeling known-key attacks.

CK and CKw. The CK (Cannetti-Krawczyk) and CKw (Weak CK) models correspond to a BR adversary with two additional powers. First, the adversary can also compromise long-term keys after the session (as in AF, for CK, and as in AFC, for CKw). Second, the adversary can reveal the local state (e. g., the random numbers) generated in other sessions.

eCK-1 and eCK-2. The eCK-1 and eCK-2 models together model the eCK adversary model (Extended CK). The eCK model is similar to the combination of the CKw and CA adversary. The main difference is that instead of revealing the local state of other (i. e., non-matching) sessions, the adversary can reveal the randomness generated in every session for which he did not compromise the owner's long-term private key.

B Multi-protocol Analysis Results

In Table 6 we give an overview of the protocol interactions. In the left column the protocol and property considered is indicated. The markers in the top row indicate the protocols that were considered to be running in parallel. An open dot in the table denotes that a violation was found that requires self-communication.

A closed dot indicates a violation that does not require self-communication. The interferences found seem to be of a harmless nature but could trivially have been avoided by putting appropriate unique constants in each message, i. e., by tagging.

Table 6. Multi-protocol interactions leading to property violations. These mainly involve wrong assumptions on the protocol variant that the partner is running.

No	Prot	Claim	ikev1-pk-a1	ikev1-pk-a12	ikev1-pk-a2	ikev1-pk-a22	ikev1-sig-a1	ikev1-sig-m	ikev1-sig-m-perlman	ikev2-mac	ikev2-mac2	ikev2-mactosig	ikev2-mactosig2	ikev2-sig	ikev2-sig2	ikev2-sigtomac
1	ikev1-pk-a1	I Agreement		●												
2	ikev1-pk-a1	I Weakagree		●												
3	ikev1-pk-a12	I Agreement	●													
4	ikev1-pk-a12	I Weakagree	●													
5	ikev1-pk-a2	I Agreement				●										
6	ikev1-pk-a2	I Weakagree				●										
7	ikev1-pk-a22	I Agreement			●											
8	ikev1-pk-a22	I Weakagree			●											
9	ikev1-sig-a-perlman1	R Agreement					○									
10	ikev1-sig-a-perlman1	R Weakagree					○									
11	ikev1-sig-m-perlman	R Agreement						●								
12	ikev1-sig-m-perlman	R Weakagree						●								
13	ikev2-mac	R Agreement										●				
14	ikev2-mac	R Weakagree										●				
15	ikev2-mac2	R Agreement											●			
16	ikev2-mac2	R Weakagree											●			
17	ikev2-mactosig	R Agreement								●						
18	ikev2-mactosig	R Weakagree								●						
19	ikev2-mactosig2	R Agreement									●					
20	ikev2-mactosig2	R Weakagree									●					
21	ikev2-sig	I Agreement														
22	ikev2-sig	I Weakagree														
23	ikev2-sig	R Agreement														●
24	ikev2-sig	R Weakagree														●
25	ikev2-sig2	I Agreement														
26	ikev2-sigtomac	R Agreement												●		
27	ikev2-sigtomac	R Weakagree												●		

Adapting Helios for Provable Ballot Privacy

David Bernhard[1], Véronique Cortier[2], Olivier Pereira[3],
Ben Smyth[2], and Bogdan Warinschi[1]

[1] University of Bristol, England
[2] LORIA - CNRS, France
[3] Université Catholique de Louvain, Belgium

Abstract. Recent results show that the current implementation of Helios, a practical e-voting protocol, does not ensure independence of the cast votes, and demonstrate the impact of this lack of independence on vote privacy. Some simple fixes seem to be available and security of the revised scheme has been studied with respect to symbolic models.

In this paper we study the security of Helios using computational models. Our first contribution is a model for the property known as ballot privacy that generalizes and extends several existing ones.

Using this model, we investigate an abstract voting scheme (of which the revised Helios is an instantiation) built from an arbitrary encryption scheme with certain functional properties. We prove, generically, that whenever this encryption scheme falls in the class of *voting-friendly* schemes that we define, the resulting voting scheme provably satisfies ballot privacy.

We explain how our general result yields cryptographic security guarantees for the revised version of Helios (albeit from non-standard assumptions).

Furthermore, we show (by giving two distinct constructions) that it is possible to construct voting-friendly encryption, and therefore voting schemes, using only standard cryptographic tools. We detail an instantiation based on ElGamal encryption and Fiat-Shamir non-interactive zero-knowledge proofs that closely resembles Helios and which provably satisfies ballot privacy.

1 Introduction

Electronic voting protocols have the potential to offer efficient and sound tallying with the added convenience of remote voting. It is therefore not surprising that their use has started to gain ground in practice: USA, Norway and Estonia are examples of countries where e-voting protocols have been, at the very least, trialled in elections on a national scale.

Due to the sensitive nature of elections, security of e-voting protocols is crucial and has been investigated extensively. Among the security properties that have been identified for e-voting, perhaps the most desirable one is that users' votes should remain confidential. Three levels of confidentiality have been identified. These are (in increasing strength) the following.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 335–354, 2011.

- Ballot privacy: A voter's vote is not revealed to anyone.
- Receipt–freeness: A voter cannot obtain information which can prove to a coercer how she voted.
- Coercion resistance: Even a voter who collaborates with a coercer cannot obtain information that proves how she voted.

Other important properties that are desirable include ballot independence [12] (the ballots cast do not depend on each other) and end-to-end verifiability [23,28,38] (it is possible to verify that the election process has been followed honestly).

This paper is motivated by recent developments regarding the security of the Helios voting scheme [45]. Starting from version 2.0 [35], Helios has been using a variant of a classical protocol by Cramer et al. [14] incorporating tweaks proposed by Benaloh [29], and has been used in real-world elections, for example by the International Association for Cryptographic Research (IACR) to elect its 2010 board [36], by Princeton University to elect the undergraduate student government [46] and to elect the president of the Université Catholique de Louvain [35]. Helios aims to achieve only ballot privacy and explicitly discards the stronger confidentiality notions (which it does not satisfy) in favor of efficiency. It turns out that the current implementation of Helios does *not* enforce ballot independence (contrary to the original protocol of Cramer et al. [14]) and, as a result, Cortier and Smyth [37,42] have exhibited several attacks against the ballot privacy property of Helios. (The property is called "ballot secrecy" in Cortier and Smyth's papers.) The attacks range from simple ballot copying to subtle reuse of parts of existing ballots, however they can all be detected (and prevented) by public algorithms. A revised scheme has been proved secure in a symbolic model but its security in the stronger, computational sense has not been assessed.

Contributions. We start by providing a *computational* security model for ballot privacy (Section 2). In a sense, our model generalizes and strengthens the model of [24,26] where an attacker tries to distinguish when two ballots are swapped. Here, we ask that the adversary cannot detect whether the ballots cast are ballots for votes that the adversary has chosen or not. In doing so, the adversary is allowed to control arbitrarily many players and see the result of the election. Our model uses cryptographic games and thus avoids imposing the more onerous constraints that other definitional styles (in particular simulability) require from protocols.

Next we turn our attention to the revised version of Helios. Our analysis follows a somewhat indirect route: instead of directly analysing the scheme as it has been implemented, we analyze an abstract version of Helios that follows the same basic architecture, but where the concrete primitives are replaced with more abstract versions. Of course, the version we analyze implements the suggested weeding of ballots. We present this abstract scheme as a generic construction of a voting scheme starting from encryption scheme with specific functional and security properties (Section 3).

Focusing on this more abstract version brings important benefits. Firstly, we pin-down more clearly the requirements that the underlying primitives should satisfy. Specifically, we identify a class of *voting-friendly* encryption schemes which when plugged in our construction yield voting schemes with provable ballot privacy. Roughly speaking, such encryption schemes are IND-CCA2 secure and have what we call a homomorphic embedding (parts of the ciphertexts can be seen as ciphertexts of a homomorphic encryption scheme). Secondly, our analysis applies to all voting schemes obtained as instantiations of our generic construction. Although we analyze and propose constructions which for efficiency reasons resort to random oracles, our generic approach also invites other (non-random oracle based) instantiations.

Next, we show how to construct voting-friendly encryption schemes using standard cryptographic tools (Section 4). We discuss two distinct designs. The first construction starts from an arbitrary (IND-CPA) homomorphic encryption scheme and attaches to its ciphertexts a zero-knowledge proof of knowledge of the plaintext. We refer to this construction as the Enc+PoK construction. Despite its intuitive appeal, we currently do not know how to prove that the above design leads to an IND-CCA2 secure encryption scheme (a proprety demanded by voting-friendliness). We therefore cannot conclude the security of our generic scheme when implemented with an arbitrary Enc+PoK scheme. Nevertheless, an investigation into this construction is important since the instantiation where Enc is the ElGamal scheme and PoK is obtained using the Fiat-Shamir paradigm applied to a Schnorr-like protocol corresponds precisely to the encryption scheme currently used in Helios. The security of this specific construction has been analyzed in prior work. Tsiounis and Yung [17] and Schnorr and Jakobsson [19] demonstrate that the scheme is IND-CCA2 secure, but their proofs rely on highly non-standard assumptions. Nevertheless, in conjunction with the security of our main construction, one can conclude that the current implementation of Helios satisfies ballot privacy based on either the assumption in [17] or those of [19].

We then take a closer look at the Enc+PoK construction and revisit a technical reason that prevents an IND-CCA2 security proof, first studied by Shoup and Gennaro [16]. Very roughly, the problem is that the knowledge extractor associated to the proof of knowledge may fail if used multiple times since its associated security guarantees are only for constant (or logarithmically many) uses. With this in mind, we note that a security proof is possible if the proof of knowledge has a so called *straight line* extractor [22]. This type of extractor can be reused polynomially many times. In this case, the Enc+PoK construction leads to a voting-friendly encryption scheme, whenever Enc is an arbitrary IND-CPA homomorphic encryption scheme.

The second design uses the well-known Naor-Yung transformation [7]. We show that if the starting scheme is an arbitrary (IND-CPA) homomorphic encryption scheme then the result of applying the NY transform is a voting-friendly encryption scheme. Applied generically, the transform may lead to non-efficient schemes (one of its components is a simulation-sound zero-knowledge proof of membership [18]). We present a related construction (where the proof of membership is replaced by a proof of knowledge) which can be efficiently instantiated in the random oracle model. In the final section of the paper (Section 5) we propose adopting an instantiation of Helios where the encryption-friendly scheme is implemented as above. The computational overhead for this scheme is reasonable (and can be further improved through specific optimization) and the scheme comes with the formal guarantees offered by the results of this paper.

Related work. Chevallier-Mames *et al.* [27] present an unconditional definition of ballot privacy but Helios cannot be expected to satisfy this definition due to its reliance on computational assumptions. Chevallier-Mames additionally show that their definition of unconditional ballot privacy is incompatible with universal verifiability; however, ballot privacy and universal verifiability have been shown to coexist under weaker assumptions, for example as witnessed by Juels, Catalano & Jakobsson [23]. Computational definitions of ballot privacy have been considered by Benaloh *et al.* [2,4,5]. These definitions however do not come with a general characterization of the properties that an encryption scheme should satisfy in order to ensure that they are satisfied (the corresponding security notions did not exist at that time either). Wikström [34] considered the general problem of secure submission of inputs with applications to mixnet-based voting protocols. His definitions and constructions are the most closely related to ours, and will be discussed below. Other definitions for voting systems have been proposed in terms of UC realization of ideal voting functionalities, starting with Groth [21], which capture privacy as part of the functionality behavior.

In addition, receipt-freeness has been considered by Benaloh & Tuinstra [11] and Moran & Naor [25] and coercion resistance has been studied by Juels, Catalano & Jakobsson [23], Küsters, Truderung & Vogt [40] and Unruh & Müller-Quade [39]. These definitions can be used to show ballot privacy because it is believed to be a weaker condition [11,26]; however, they are too strong for protocols which only provide ballot privacy and in particular, they cannot be used to analyse ballot privacy in Helios. Ballot privacy has also been formalized in the symbolic model (for example, [26,33]) but the symbolic model suffers a serious weakness: In general, a correct security proof does not imply the security of the protocol. Cortier & Smyth [37,42] present an attack against ballot privacy in Helios and propose a variant of Helios which aims to prevent the attack by weeding ballots. Their solution has been shown to satisfy ballot privacy in the symbolic model but Cortier & Smyth acknowledge that a thorough cryptographic analysis of the solution is necessary.

2 Ballot Privacy

Notation. Throughout this paper, we use the following notation. Assignment and input/output of algorithms are both denoted by a left-facing arrow $\leftarrow$. Picking a value x uniformly at random from a set S is denoted by $x \overset{R}{\leftarrow} S$. The expression $C \overset{+}{\leftarrow} c$ appends c to the list C, () on its own is an empty list. We use "C" style returns in algorithms, i.e. "Return $a = b$" to mean return 1 if $a = b$, otherwise 0.

A function f is called *negligible* if for any polynomial P, there exists η_0 such that $\forall \eta \geq \eta_0$, $f(\eta) \leq \frac{1}{P(\eta)}$.

2.1 Voting Schemes

In this section we fix a general syntax for the class of voting schemes that we treat in this paper. In particular, our syntax encompasses several variations of the Helios protocol.

We consider schemes for votes in a non-empty set $\mathbb{V}$, and we assume $\perp$ to be a special symbol not in $\mathbb{V}$ that indicates that the voter has abstained. The result of an election is then an arbitrary function ρ that takes a list of votes as input and returns the election result. Elections are stateful, so the algorithms that we define next use such a state. Since often, and in particular in the case of Helios, this state is a bulletin board, in the definition below we write BB for this state (and even refer to it as a bulletin board).

Definition 1 (Voting scheme). *Algorithms* $(\mathsf{Setup}, \mathsf{Vote}, \mathsf{ProcessBallot}, \mathsf{Tally})$ *define a voting scheme as follows.*

- Setup*: The setup algorithm takes a security parameter* 1^λ *as input and returns secret information* x*, public information* y*, and initializes the state* BB*. We write* $(x, y, BB) \leftarrow \mathsf{Setup}(1^\lambda)$ *for this process. We assume the public information is available to all subsequent algorithms.*
- Vote*: The voting algorithm takes a vote* $v \in \mathbb{V}$ *as input and produces as output a ballot* b *(that encodes the vote). We write* $b \leftarrow \mathsf{Vote}(v)$ *for this process.*
- $\mathsf{ProcessBallot}$*: The ballot processing algorithm takes a candidate ballot* b *and a bulletin board* BB*, checks the ballot for correctness (e.g. that it is well formed, it is not a duplicate, etc.) and returns a result (accept/reject) and the new state of the bulletin board. We write* $(a, BB) \leftarrow \mathsf{ProcessBallot}(BB, b)$ *for this process. Here* a *is either* accept *or* reject*.*
- Tally*: The tallying algorithm takes the secret information* x *and the bulletin board* BB *and produces the election result.*

For correctness of the scheme, we demand two conditions: 1) ballot tallying corresponds to evaluating the function ρ on the underlying votes; and 2) correctly constructed votes will be accepted by the ballot processing algorithm. Both conditions should hold with overwhelming probability and can be captured by the experiment described in Figure 1. In this experiment, an adversary repeatedly submits votes $v_1, v_2, \ldots \in \mathbb{V}$ and each vote is used to construct a ballot which is then processed. The game outputs 1 (the adversary wins) if the $\mathsf{ProcessBallot}$ algorithm rejects some ballot or the result of the election does not correspond to the votes cast. The voting scheme is correct if the algorithm outputs 1 with at most negligible probability.

2.2 Security Model

Informally, ballot privacy is satisfied if an adversary in control of arbitrarily many voters cannot learn anything about the votes of the remaining, honest

```
$\mathbf{Exp}_{\Pi}^{corr}(A)$
    $(x, y, BB) \leftarrow \mathsf{Setup}$
    $V = ()$
    repeat
        $(a, v) \leftarrow A$
        $b \leftarrow \mathsf{Vote}(v)$
        $(r, BB) \leftarrow \mathsf{ProcessBallot}(BB, b)$
        $V \xleftarrow{+} v$
    until $a = \mathsf{stop}$ or $r = \mathsf{reject}$
    if $r =$“$\mathsf{reject}$” or $\mathsf{Tally}(x, BB) \neq \rho(V)$ then return 1 else return 0
```

Fig. 1. Experiment for defining the correctness of a voting scheme

voters beyond what can be inferred from the election result. The adversary can read the (public) bulletin board and the communication channels between the honest parties and the bulletin board (in other words, we assume them to be authentic but not secret). Ballot privacy requires that the adversary is unable to distinguish between real ballots and fake ballots, where ballots are replaced by ballots for some fixed vote ε chosen by the adversary.

Formally, we consider an adversary that can issue two types of queries, vote and ballot, to an oracle $\mathcal{O}$. The oracle maintains two bulletin boards initialized via the setup algorithm: BB is visible to the adversary and BB' always contains ballots for the real votes. A vote query causes a ballot for the given vote to be placed on the hidden BB'. In the real world, the same ballot is placed on BB; in the fake one a ballot for ε is placed on BB instead. A ballot query always causes the submitted ballot to be processed on both boards. This process is defined formally in Figure 2. The experiment on the right of Figure 2 is used to define ballot privacy. The selection of β corresponds to the real world ($\beta = 0$) or the fake world ($\beta = 1$). Throughout the experiment the adversary has access to BB, but tallying is done using BB'.

Definition 2 (Ballot Privacy). *We define the advantage of adversary A in defeating ballot privacy for voting scheme Π by:*

$$\mathbf{Adv}_{\Pi}^{BS}(A) = \Pr[\mathbf{Exp}_{\Pi}^{BS}(A) = 1] - \frac{1}{2}$$

and say that Π ensures ballot privacy if for any efficient adversary its advantage is negligible.

We make a few remarks regarding the security model that we propose. Firstly, we use cryptographic games rather than a simulation based definition. The former offer well-accepted levels of security, are more flexible, and allow for more efficient implementations. Second, we model directly the more relaxed notion of vote privacy and not stronger notions like receipt-freeness or coercion resistance [26]. While stronger notions are certainly desirable, they are more

```
$\mathsf{vote}(v)$
    $b' \leftarrow \mathsf{Vote}(v)$
    if $\beta = 0$ then $b \leftarrow b'$
            else $b \leftarrow \mathsf{Vote}(\varepsilon)$
    $(r, BB) \leftarrow \mathsf{ProcessBallot}(b, BB)$
    $(r', BB') \leftarrow \mathsf{ProcessBallot}(b', BB')$
    return $(r, BB, b)$

$\mathsf{ballot}(b)$
    $(r, BB) \leftarrow \mathsf{ProcessBallot}(b, BB)$
    if $r = \mathsf{accept}$ then
        $(r', BB') \leftarrow \mathsf{ProcessBallot}(b, BB')$
    return $(r, BB)$
```

```
$\mathbf{Exp}_{\Pi}^{BS}(A)$
    $(x, y, BB) \leftarrow \mathsf{Setup}(1^{\lambda})$
    $BB' \leftarrow BB$
    $(\varepsilon, st) \leftarrow A(y)$
    $\beta \leftarrow \{0, 1\}$
    $st \leftarrow A^{\mathcal{O}}(st)$
    $\mathsf{result} \leftarrow \mathsf{Tally}(x, BB')$
    $\hat{\beta} \leftarrow A(st, \mathsf{result})$
    return $\beta = \hat{\beta}$
```

Fig. 2. The algorithms on the left explain how the oracle processes adversary's queries. The experiment on the right is used to define ballot privacy.

difficult to achieve leading to rather inefficient protocols. Indeed, Helios deliberately trades these stronger notions for efficiency. Finally, we emphasize that our computational definition does not mirror existing security definitions in more abstract models, e.g. [24]. It turns out that the direct extension of that definition to computational models seems strictly weaker than the definition that we provide. We comment more on this point later in the paper.

3 A Generic Construction of Voting Schemes with Ballot Privacy

In this section we present a generic construction of a voting scheme starting from any encryption scheme with certain properties. We first fix this class of encryption schemes (which we call voting-friendly), then give our construction and prove its security.

3.1 Voting-Friendly Encryption

In a nutshell, a voting-friendly encryption scheme is a "(threshold) checkable provable IND-CCA2 secure public key encryption scheme with key derivation and a homomorphic embedding". These rather convoluted looking requirements are in fact not too onerous. We explain informally each of the requirements in turn and give formal definitions. For simplicity, the presentation in this section is for the non-threshold case, that is decryption is carried out using a single key by a single party, as opposed to implementing decryption via an interactive process where several parties share the keys.

Non-Interactive Zero Knowledge Proof Systems. Here we recall some basic notions regarding non-interactive zero-knowledge proof systems [6]. Given language L_R defined by NP relation R we write $(w, x) \in R$ if w is the witness that $x \in L_R$.

A proof system for L_R is given by a pair of algorithms $(\mathsf{Prover}, \mathsf{Verifier})$ called prover and verifier, respectively. We distinguish between proof systems in the common reference string model (in this situation, an additional algorithm Setup produces a common reference string accessible to both the prover and the verifier) and the random oracle model (where the setup is not required, but all algorithms in the system have access to a random oracle). In a standard execution of the proof system, the prover and the verifier both have an element $x \in L_R$ as input and in addition, the prover has as input a witness w that $x \in L_R$ (i.e. $R(w, x) = 1$). The prover sends a single message π to the verifier who outputs the decision to accept/reject. We call π a proof for the statement $x \in L_R$. Typical requirements for such proof systems are that they should be sound (if the input x is not in L_R then the verifier rejects π with overwhelming probability) and complete (if x is in the language then the verifier accepts π with probability 1). We write $\pi \leftarrow \mathsf{Prover}$ for the process of producing proof π when the statement x and the witness w are clear from the context. A non-interactive proof system is zero-knowledge if there exists a simulator Sim that is able to produce transcripts indistinguishable from those of a normal execution of the protocol. The simulator may use a trapdoor in the common reference string model, or can program the random oracle in the random oracle model. We occasionally write $(\mathsf{Prover}, \mathsf{Verifier}) : R$ to indicate that the proof system is for the language L_R.

We assume the reader is familiar with public key encryption and its associated security notions. We write $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for the key generation, encryption, and decryption algorithms of a public key encryption scheme.

Homomorphic encryption. We also briefly recall the notion of *homomorphic* encryption. An encryption scheme is homomorphic if the plaintext space is a group and there exists an algorithm Add that takes two ciphertexts for messages m_0 and m_1 and produces a ciphertext for $m_0 \circ m_1$ (where $\circ$ is the group operation on plaintexts).

Embeddable Encryption. A crucial property for the encryption schemes that are the focus of this section is that they have a *homomorphic embedding.* Informally, this property means that it is possible to identify part(s) of the ciphertexts as forming a ciphertext for some other encryption scheme, and this second encryption scheme is homomorphic. The ElGamal+PoK construction sketched in the previous section is an example of an encryption scheme with an homomorphic embedding. Indeed the e component of a ciphertext (e, π) is a ciphertext for an homomorphic encryption scheme (ElGamal). The next definition makes this discussion more precise.

Definition 3 (Homomorphic Embedding). *We say that the homomorphic encryption scheme $\Pi = (\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EAdd})$ is embedded in encryption scheme $\Pi' = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, or alternatively that encryption scheme Π' has Π as a* homomorphic embedding *if there are algorithms* $\mathsf{ExtractKey}, \mathsf{Extract}$ *such that for all m, pk, sk, c*

$$\mathsf{EGen}() = \mathsf{ExtractKey}(\mathsf{Gen}())$$

$$\mathsf{EEnc}(m, \mathsf{ExtractKey}(pk)) = \mathsf{Extract}(\mathsf{Enc}(m, pk))$$
$$\mathsf{Dec}(c, sk) = \mathsf{EDec}(\mathsf{Extract}(c), sk)$$

Essentially, the ExtractKey algorithm maps keys (or key pairs) for the "larger" scheme to keys for the embedded one, and the Extract algorithm extracts the ciphertext for the embedded scheme out of ciphertext for the larger one, while performing validity verifications at the same time.

The Extract algorithm must, by definition, produce a ciphertext that decrypts to the same value as the input that it is given; in particular it must produce a "ciphertext" that decrypts to $\perp$ if and only if its input does. However, the Extract algorithm does not take any secret keys as input. This implies that anyone can check whether a ciphertext is valid (in the sense that it decrypts to something other than $\perp$) without knowing the secret key. This property forms the basis for combining homomorphic and IND-CCA2 secure encryption in our construction.

We note that an IND-CCA2 secure cryptosystem with homomorphic embedding is actually very close to a submission secure augmented (SSA) cryptosystem as defined by Wikström [34]. Some important differences appear, though. The most important one is that SSA cryptosystems do not require public verifiability of the ciphertexts: it might be necessary to publish a private key augmentation to be able to perform ciphertext validity checks. While this feature enables efficient solutions that are secure in the standard model, it is however often not desirable in practice: it is quite useful to be able to dismiss invalid votes as soon as they are submitted (and to resolve potential conflicts at that time) rather than needing to wait for some partial key to be revealed. Besides, in order to mitigate this inconvenience, SSA cryptosystems allow multiple independent augmentations, which enables updating an augmentation and revealing the previous one in order to be able to check the validity of previously submitted ciphertexts. Our requirement of immediate public verifiability property makes this feature unnecessary for our purpose.

We also note that in concurrent work, Persiano [44] and Smart [41] define similar embedding concepts.

S2P Key Derivation. This property simply requires that if a key pair is produced by the key generation algorithm of an encryption scheme then it is possible to compute the public key from the secret key. This property will allow us to use proofs of knowledge of the secret key corresponding to the public key.

Definition 4 (S2P Key Derivation). *An encryption scheme has the* S2P key derivation *property if there is an algorithm* DeriveKey *such that* $(x, y) \leftarrow \mathsf{Gen}$ *implies* $y = \mathsf{DeriveKey}(x)$.

Provable Encryption. In our generic construction voters need to certify that various encryptions in the ballots that they produce satisfy some desirable properties (e.g. that a ciphertext encrypts 0 or 1, and not something else), and such certification can be done via zero-knowledge proofs of knowledge. Since all of the statements that we are interested in are NP statements, the existence of appropriate proof systems follows from general results [9]. Here, we make more precise the statements for which we demand the existence of such proof systems and introduce some useful notation for the proof systems associated to the various languages that we define.

In particular, it should be possible to prove knowledge of the secret key corresponding to the public key, knowledge of the plaintext underlying a ciphertext, as well as proving that a certain plaintext has been obtained by decrypting with the key associated to the public key. To avoid complex nomenclature, we call a scheme for which this is possible a scheme with provable encryption.

Definition 5 (Provable Encryption). *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* provable *if it has the S2P key derivation property and the following non-interactive zero-knowledge proof systems exist:*

1. $(\mathsf{ProveGen}, \mathsf{VerifyGen})$: $R_1(x, y) := y \overset{?}{=} \mathsf{DeriveKey}(x)$
2. $(\mathsf{ProveEnc}, \mathsf{VerifyEnc})$: $R_2((m, r), c) := c \overset{?}{=} \mathsf{Enc}(m; r)$
3. $(\mathsf{ProveDec}, \mathsf{VerifyDec})$: $R_3(x, (c, y, d)) := y \overset{?}{=} \mathsf{DeriveKey}(x) \wedge d \overset{?}{=} \mathsf{Dec}(x, c)$

The above definition is for standard encryption schemes. For the case when the encryption scheme that we need is embedded, we demand in addition the existence of proof systems for the following two properties. The first requires that one can prove a statement that involves plaintexts underlying several ciphertexts, and secondly, one should be able to prove that the keys for the embedded schemes in use have been correctly obtained from the keys of the embedding one. This latter condition is a simple adaptation of provability as defined above.

Definition 6 (Provable Embedding). *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for message space M with embedded scheme* $(\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec})$ *has* embedded provability *for $M' \subseteq M^N$ (for some $N \in \mathbb{N}$) if the following zero-knowledge proof-systems exist:*

1. $(\mathsf{ProveGen}, \mathsf{VerifyGen})$: $R_4(x, y) := y \overset{?}{=} \mathsf{DeriveKey}(x)$
2. $(\mathsf{ProveEnc}, \mathsf{VerifyEnc})$: $R_5((m_1, m_2, \ldots, m_N, r_1, r_2, \ldots, r_N), (c_1, c_2, \ldots, c_N)) :=$

$$\bigwedge_{i=1}^{N} c_i \overset{?}{=} \mathsf{Enc}(m_i; r_i) \wedge (m_1, \ldots, m_N) \in M'$$

3. $(\mathsf{ProveEDec}, \mathsf{VerifyEDec})$: $R_6(x, (y, d, c)) :=$

$$y \overset{?}{=} \mathsf{DeriveKey}(x) \wedge (x', y') \leftarrow \mathsf{ExtractKey}(x, y) \wedge d \overset{?}{=} \mathsf{EDec}(x', c)$$

In the last relation, the second conjunct is not a boolean condition, but simply indicates that the keypair (x', y') is derived from (x, y) using the $\mathsf{ExtractKey}$ algorithm.

The following definition states all the properties that we require from an encryption scheme in order to be able to implement our generic voting scheme.

Definition 7 (Voting-Friendly Encryption). *A voting-friendly encryption scheme for vote space $\mathbb{V}$ is a public-key scheme for message space M with $\mathbb{V} \subseteq M^N$ such that it is IND-CCA2 secure and has S2P key derivation, an embedded homomorphic scheme and embedded provability for $\mathbb{V}$.*

Note that voting-friendly encryption requires security guarantees of both the encryption scheme and the contained proof systems.

3.2 Our Generic Construction

In this section we describe a voting scheme based on an arbitrary voting-friendly encryption scheme. The design idea is similar to that of Helios.

The scheme handles elections with multiple candidates. In an election with three candidates a vote is a triple (a, b, c) such that $a, b, c \in \{0, 1\}$ and $a+b+c = 1$. A ballot is then simply formed by individually encrypting each component of the list with an IND-CCA scheme that has an homomorphic embedding, and proving in zero-knowledge that the individual plaintexts in a ballot satisfy the desired relation. To prevent an adversary from casting a vote somehow related to that of an honest voter, we ensure that each ballot cast does not contain any ciphertexts that are duplicates of ones in the ballots already on the bulletin board. This condition is checked while processing ballots.

More formally, denote the set of ciphertexts contained in a ballot b by $\mathsf{Cipher}(b)$ and the set of all ciphertexts on the bulletin board BB by $\mathsf{Cipher}(BB)$, that is $\mathsf{Cipher}(BB) = \bigcup_{b' \in BB} \mathsf{Cipher}(b')$. When submitting a ballot b, we check that $\mathsf{Cipher}(b) \cap \mathsf{Cipher}(BB) = \emptyset$.

Definition 8 (Abstract Voting Scheme). *Let Π be a voting-friendly encryption scheme. The abstract voting scheme $V(\Pi)$ is the construction consisting of algorithms 1–4.*

In our construction, $\mathcal{V}$ is the set of voters, $\mathcal{Z}$ is a party representing "the public" (elements sent to $\mathcal{Z}$ are published) which also functions as a trusted party for generating the initial setup parameters and $\mathcal{T}$ is the trustee of the election (that receives the decryption keys).

If M is the message space of the voting-friendly encryption scheme we consider the space of votes to be $\mathbb{V} \subseteq M^N$ for some $N \in \mathbb{N}$.

We consider result functions of the form $\rho : \mathbb{V}^* \to M^*$ where $\mathbb{V}^* := \cup_{i \in \mathbb{N}_0} \mathbb{V}^i$ (this allows us to tally an arbitrary number of votes) and each component of the range of ρ can be described by a sum of the form $\rho_k = \sum_{i \in \mathbb{N}} a_{i,k} \cdot v_i$ for constants $a_{i,k} \in \mathbb{N}$. This covers the class of result functions that can be computed homomorphically, including normal and weighted sums of votes but also the special case of revealing all the votes and allows us to exploit the homomorphism in the tallying operation: The same operation can be performed on homomorphic ciphertexts using the EAdd algorithm, for which we write $\oplus$ i.e. $a \oplus b := \mathsf{EAdd}(a, b)$. Furthermore, we can define scalar multiplication $\otimes$ on the ciphertexts i.e. $2 \otimes a := \mathsf{EAdd}(a, a)$.

We also provide a public verification algorithm as Algorithm 5 although we do not define this property formally.

Algorithm 1. Setup(1^λ)

```
$\mathcal{Z}$ :
    $params \leftarrow$ Setup$(1^\lambda)$. These parameters are implicitly available to all further algorithms.
    $BB \leftarrow ()$
$\mathcal{T}$ :
    $(x, y) \leftarrow$ Gen$(1^\lambda)$
    $\pi^{Gen} \leftarrow$ ProveGen$(x, y)$
    $\mathcal{Z} \leftarrow (y, \pi^{Gen})$
$\mathcal{Z}$ :
    VerifyGen$(y, \pi^{Gen}) \overset{?}{=} 1$ or abort with failure.
```

Algorithm 2. Vote$((v_1, v_2, \ldots, v_N))$

```
$\forall j \in \{1, 2, \ldots, N\}$
    $c_j \leftarrow$ Enc$(y, v_j)$
    $\pi_j^b \leftarrow$ ProveEnc$(y, v_j, c_j)$
    $b_j \leftarrow (c_j, \pi_j^b)$
output $b$
```

Algorithm 3. ProcessBallot(b, BB)

```
    if VerifyEnc$(b) = 0$ then return (“reject”, $BB$) end if
    for all $c \in$ Cipher$(b)$ do
        if Extract$(c) = \perp$ then return (“reject”, $BB$) end if
        if Cipher$(b) \cap$ Cipher$(BB) \neq \emptyset$ then return (“reject”, $BB$) end if
    end for
    $BB \overset{+}{\leftarrow} b$
    return (“accept”, $BB$)
```

Algorithm 4. Tally(BB)

```
    for all $c_j \in BB$ $(j \in \mathcal{V})$ do  $e'_j \leftarrow$ Extract$(c_j)$  end for
    for all k do
        $e''_k \leftarrow \bigoplus_{j \in \mathcal{V}} (a_{j,k} \otimes e'_j)$ {I.e. use EAdd to compute ciphertexts for the results.}
        $r_k \leftarrow$ EDec$(x, e''_k)$
        $\pi_k^{Dec} \leftarrow$ ProveEDec$(x, e''_k, r_k)$
    end for
    $\mathcal{Z} \leftarrow (r_k, \pi_k^{Dec})_k$
```

Algorithm 5. Verification

$\mathcal{Z}$ performs the following, aborting if any of the checks (denoted by $\stackrel{?}{=}$) fail. The ordering on $\mathcal{V}$ is a slight abuse of notation; it represents the order the ballots were received in. If successful, the result of the election is r.

```
$\mathsf{VerifyGen}(y, \pi^{Gen}) \stackrel{?}{=} 1$
for all $j \in \mathcal{V}$ do
    $(c_j, \pi_j^b) \leftarrow b_j$
    $\mathsf{VerifyEnc}(b_j) \stackrel{?}{=} 1$
    $(c_j \notin (c_{j'})_{j' \in \mathcal{V}, j' < j}) \stackrel{?}{=} 1$
    $e'_j \leftarrow \mathsf{Extract}(c_j)$
    $e'_j \stackrel{?}{\neq} \perp$
end for
$e' \leftarrow \mathsf{EAdd}(\rho, (e'_j)_{j \in \mathcal{V}})$
$\mathsf{VerifyEDec}(r, \pi^{Dec}, e') \stackrel{?}{=} 1$
```

We only prove ballot privacy of our construction formally; correctness follows from the correctness of the voting-friendly encryption scheme. The following theorem states that ballot privacy relies entirely on the security of the underlying voting-friendly scheme.

Theorem 1. *Let Π be a voting-friendly encryption scheme. Then $V(\Pi)$ has ballot privacy.*

To prove the theorem we proceed in two steps. First, we strip the voting scheme of the unnecessary details that concern verifiability, resulting in a scheme that we call "mini-voting". We prove that ballot privacy for this latter scheme only relies on the IND-CCA2 security of the encryption scheme employed (which highlights IND-CCA2 security as the crucial property needed from the underlying building block). We then explain how to adapt the proof to show the security of $V(\Pi)$.

The full proof can be found in the full version of this paper.

4 Constructions for Voting–Friendly Schemes

In the previous section we gave a generic construction of a voting scheme with ballot privacy starting from an arbitrary voting-friendly encryption scheme. In this section we show that such schemes can be easily constructed using standard cryptographic tools in both the standard and the random oracle models. We discuss three different possibilities.

Encrypt + PoK. This construction does not lead immediately to a voting-friendly scheme but its security is highly relevant to that of Helios, and the design idea forms the basis of a construction that we discuss later.

Under this paradigm, one attempts to construct an IND-CCA2 scheme starting from an IND-CPA scheme and adding to the ciphertext a non-interactive proof of knowledge of the underlying plaintext. Intuitively, this ensures that an adversary cannot make use of a decryption oracle (since he must know the underlying plaintext of any ciphertext) hence the security of the scheme only relies on IND-CPA security. Unfortunately, this intuition fails to lend itself to a rigorous proof, and currently the question whether Enc+PoK yields an IND-CCA2 scheme is widely open. A detailed treatment of the problem first appeared in [16].

Yet, the question is important for the security of Helios: the current implementation is essentially an instantiation of our generic construction with an Enc+PoK encryption scheme. More precisely the encryption scheme Enc is ElGamal, and the proof of knowledge is obtained by applying the Fiat-Shamir transform to a Schnorr proof. Per the above discussion, no general results imply that the resulting ElGamal+PoK scheme is IND-CCA2 secure (a requirement for voting-friendliness) and our generic result does not apply. However, if one is prepared to accept less standard assumptions, two existing results come in handy. The security of the particular construction that employs ElGamal encryption and Fiat-Shamir zero-knowledge proofs of knowledge has been investigated by Tsiounis & Yung [17] and Schnorr & Jakobsson [19]. Both works support the conjecture that the construction is IND-CCA2 but neither result is fully satisfactory. Tsiounis & Yung make a knowledge assumption that essentially sidesteps a crucial part in the security proof, whereas the proof of Schnorr & Jakobsson assumes both generic groups [13] and random oracles [10]. Nevertheless, since using either assumption we can show that ElGamal+PoK construction is a voting-friendly scheme, we conclude that Helios satisfies ballot privacy under the same assumptions. Unfortunately, the security of the construction under standard assumptions is a long-standing open question. This observation motivates the search for alternative constructions of voting-friendly schemes.

Straight-line Extractors. To motivate the construction that we discuss now, it is instructive to explain why a proof that Enc+PoK is IND-CCA2 fails. In such a proof, when reducing the security of the scheme to that of the underlying primitive, a challenger would need to answer the decryption queries of the adversary. Since the underlying encryption scheme is only IND-CPA secure, the only possibility is to use the proof of knowledge to extract the plaintext underlying the queried ciphertexts. Unfortunately here the proof gets stuck. Current definitions and constructions for proofs of knowledge only consider single statements and the knowledge extractor works for polylogarithmically many proofs but it may break down (run in exponential time [19]) for polynomially many. Since the IND-CCA2 adversary is polynomially bounded answering all of its decryption queries may thus not be feasible.

A construction that gets around this problem employs a zero-knowledge proof of knowledge with a *straight-line* extractor. Such extractors do not need to rewind the prover and in this case the Enc+PoK construction yields an IND-CCA2 encryption scheme. This notion of extraction and a variation of the Fiat-Shamir transform that turns a sigma-protocol into a non-interactive proof of knowledge with a straight-line extractor in the random oracle model has recently been proposed by Fischlin [22]. As above, starting with a homomorphic encryption scheme would yield a voting friendly encryption scheme. Unfortunately the construction in that paper is not sufficiently efficient to yield a practical encryption scheme.

The Naor-Yung Transformation. This transformation starts from any IND-CPA secure encryption scheme. An encryption of message m is simply two distinct encryptions c_1 and c_2 of m under the original scheme, together with a simulation-sound zero-knowledge proof π that c_1 and c_2 encrypt the same message with an extra property that we call unique applicability. Formally, we have the following definition.

Definition 9 (Naor-Yung Transformation). *Let* $E = (\mathsf{EGen}, \mathsf{EEnc}, \mathsf{EDec})$ *be a public-key encryption system. Let* $P = (\mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *be a non-interactive zero-knowledge proof scheme for proving (in Camenisch's notation [15])*

$$\mathbf{PoK}\{(m, r_1, r_2) : \; c_1 = \mathsf{Enc}(y_1, m; r_1) \wedge c_2 = \mathsf{Enc}(y_2, m; r_2)\}$$

with uniquely applicable proofs. Assume the input to Prove *is given in the form* $(m, y_1, y_2, r_1, r_2, c_1, c_2)$.

The Naor-Yung transformation [7] $NY(E, P)$ *of the encryption system is the public-key cryptosystem defined in Algorithm 6.*

Algorithm 6. Naor-Yung Transformation

```
Gen
  $(x_1, y_1) \leftarrow \mathsf{EGen}$
  $(x_2, y_2) \leftarrow \mathsf{EGen}$
  return $((x_1, x_2), (y_1, y_2))$
$\mathsf{Enc}((y_1, y_2), m; (r_1, r_2))$
  $c_1 \leftarrow \mathsf{Enc}(y_1, m; r_1)$
  $c_2 \leftarrow \mathsf{Enc}(y_2, m; r_2)$
  $\pi \leftarrow \mathsf{Prove}(m, y_1, y_2, r_1, r_2, c_1, c_2)$
  return $(c_1, c_2, \pi)$
$\mathsf{Dec}(c_1, c_2, \pi)$
  if $\mathsf{Verify}(c_1, c_2, \pi) = 1$ then return $\mathsf{EDec}(x_1, c_2)$ else return $\perp$ end if
```

Sahai [18] showed that the above transformation yields an IND-CCA2 encryption scheme if the starting scheme is IND-CPA and the proof system is simulation-sound and has uniquely applicable proofs (essentially each proof can only be used to prove one statement).

Theorem 2 (Sahai[18]). *If the zero-knowledge proof system* P *has uniquely applicable proofs then the Naor-Yung transformation* $NY(E, P)$ *of an IND-CPA secure scheme* E *gives IND-CCA2 security.*

It turns out that if the starting encryption scheme is homomorphic, then the resulting construction is a voting-friendly encryption scheme. Indeed, the resulting scheme has a homomorphic embeding (given either the first or the second component of the ciphertext) and it is checkable (the checking algorithm only needs to verify the validity of π). As explained earlier, the required proof-systems for provability of the embeding exist, from general results. One can therefore obtain voting schemes with provable ballot privacy in the standard model starting from any homomorphic encryption scheme that is IND-CPA secure in the standard model.

In general, the above construction may not be very efficient (the simulation-sound zero-knowledge proof and associated required proof-systems may be rather heavy). In the random oracle model one can implement the above idea efficiently by replacing the simulation-sound zero-knowledge proof (of membership) with a zero-knowledge proof of knowledge of the message that underlies the two ciphertexts. Interestingly, one may regard the NY transform as providing the underlying encryption scheme with a straight-line extractor (so our previous results already apply).

The following theorem is a variation of the basic Naor-Yung transform applied to our setting.

Theorem 3. *If E is an IND-CPA secure homomorphic encryption scheme with S2P key derivation and P is a zero-knowledge proof of knowledge system with uniquely applicable proofs, then $NY(E, P)$ is a voting friendly encryption scheme.*

5 Application to the Helios Protocol

We propose an enhanced version of Helios 3.0 which is an instantiation of our generic voting scheme with a voting-friendly encryption scheme obtained from ElGamal encryption [1] via the NY transform [7]. The required proof of knowledge is obtained via the Fiat-Shamir transform [3] applied to generalized Schnorr proofs. In this scheme duplicate ballots would be rejected as defined in the ProcessBallot procedure (Algorithm 3). We can further improve the efficiency by reusing some components as described by [20].

Thanks to Theorems 1 and 3, we deduce that the enhanced version of Helios 3.0 (provably) preserves ballot privacy. The modification of Helios we propose does not change the architecture nor the trust assumption of Helios and can be easily implemented. The computational overhead is reasonable (both the length of the messages and the time of computation would at most double and some optimizations can be foreseen). In exchange, we get the formal guarantee that Helios does preserve ballot privacy, a very crucial property in the context of electronic voting. For concreteness, we prove the details of the construction, as well as a proof of security in the full version of this paper.

We emphasize that our results go beyond proving ballot privacy of a particular e-voting protocol. We have identified IND-CCA2 as a sufficient condition for constructing voting schemes satisfying our notion of ballot privacy and have given an abstract construction of a Helios-type voting scheme from IND-CPA secure homomorphic threshold encryption and non-interactive zero-knowledge proofs of knowledge. Our construction is independent of any hardness assumptions or security models (in particular, the random oracle model). We have formalized the concept of embeddable encryption and showed how to construct IND-CCA2 secure encryption with homomorphic embedding, despite the known impossibility of homomorphic IND-CCA2 secure encryption.

As further work, we plan to extend the definitions and proofs for threshold encryption scheme in order to have a fully complete proof for Helios. We are confident that our proof techniques will apply in a straightforward way. We also wish to investigate the possibility of defining ballot privacy in a more general way, e.g. allowing the current voting algorithm to be replaced by a protocol. Indeed, it could the case that casting a vote or tallying the vote require more than one step.

Acknowledgements. We are very grateful to Ben Adida for helpful discussions on how to enhance ballot privacy in Helios.

This work was partially supported by the European Commission through the ICT Programme under Contract ICT- 2007-216676 ECRYPT II, by the Interuniversity Attraction Pole P6/26 BCRYPT, and by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 258865 (ProSecure project). Olivier Pereira is a Research Associate of the Belgian Funds for Scientific Research (F.R.S.-FNRS).

References

1. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, 469–472 (1985)
2. Cohen (Benaloh), J., Fischer, M.: A Robust and Verifiable Cryptographically Secure Election Scheme. In: Proceedings of the 26th Symposium on Foundations of Computer Science, pp. 372–382 (1985)
3. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
4. Cohen (Benaloh), J., Yung, M.: Distributing the Power of a Government to Enhance the Privacy of Voters. In: Proceedings of the 5th Symposium on Principles of Distributed Computing, pp. 52–62 (1986)
5. Cohen (Benaloh), J.: Verifiable Secret-Ballot Elections. Yale University Department of Computer Science Technical Report number 561 (1987)
6. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: 20th STOC, pp. 103–112 (1988)
7. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC 1990), pp. 42–437 (1990)
8. Schnorr, C.-P.: Efficient signature generation for smart cards. Journal of cryptology 4, 161–174 (1991)

9. Damgård, I.B.: Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 341–355. Springer, Heidelberg (1993)
10. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS 1993), pp. 62–73 (1993)
11. Benaloh, J., Tuinstra, D.: Receipt-Free Secret-Ballot Elections. In: Proceedings of the 26th ACM Symposium on Theory of Computing, pp. 544–553 (1994)
12. Gennaro, R.: Achieving independence efficiently and securely. In: Proceedings of the 14th Principles of Distributed Computing Symposium (PODC 1995), pp. 130–136 (1995)
13. Shoup, V.: Lower Bounds for Discrete Logarithms and Related Problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
14. Cramer, R., Gennaro, R., Schoenmakers, B.: A Secure and Optimally Efficient Multi-authority Election Scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
15. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
16. Shoup, V., Gennaro, R.: Securing Threshold Cryptosystems against Chosen Ciphertext Attack. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 1–16. Springer, Heidelberg (1998)
17. 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)
18. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of th 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), pp. 543–553 (1999)
19. Schnorr, C.-P., Jakobsson, M.: Security of Signed ElGamal Encryption. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 73–89. Springer, Heidelberg (2000)
20. Bellare, M., Boldyreva, A., Staddon, J.: Multi-recipient encryption schemes: Security notions and randomness re-use. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567. Springer, Heidelberg (2002), http://cseweb.ucsd.edu/~mihir/papers/bbs.html
21. Groth, J.: Evaluating Security of Voting Schemes in the Universal Composability Framework. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 46–60. Springer, Heidelberg (2004)
22. Fischlin, M.: Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg (2005)
23. Juels, A., Catalano, D., Jakobsson, M.: Coercion-Resistant Electronic Elections. In: Proceedings of the 4th Workshop on Privacy in the Electronic Society (WPES 2005), pp. 61–70 (2005)
24. Kremer, S., Ryan, M.D.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)

25. Moran, T., Naor, M.: Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer, Heidelberg (2006)
26. Delaune, S., Kremer, S., Ryan, M.D.: Coercion-Resistance and Receipt-Freeness in Electronic Voting. In: 19th Computer Security Foundations Workshop (CSFW 2006), pp. 28–42 (2006)
27. Chevallier-Mames, B., Fouque, P., Pointcheval, D., Stern, J., Traoré, J.: On Some Incompatible Properties of Voting Schemes. In: Proceedings of the Workshop on Trustworthy Elections, WOTE 2006 (2006)
28. Participants of the Dagstuhl Conference on Frontiers of E-Voting. Dagstuhl Accord (2007), http://www.dagstuhlaccord.org/
29. Benaloh, J.: Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In: Proceedings of the Second Usenix/ACCURATE Electronic Voting Technology Workshop (2007)
30. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
31. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a Secure Voting System. In: Proceedings of the 29th Security and Privacy Symposium (S&P 2008), pp. 354–368 (2008)
32. Adida, B.: Helios: Web-based open-audit voting. In: 17th USENIX Security Symposium, pp. 335–348 (2008), http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf
33. Backes, M., Hriţcu, C., Maffei, M.: Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008), pp. 195–209 (2008)
34. Wikström, D.: Simplified Submission of Inputs to Protocols. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 293–308. Springer, Heidelberg (2008)
35. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.-J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (2009)
36. International association for cryptologic research. Election page at http://www.iacr.org/elections/2010
37. Cortier, V., Smyth, B.: Attacking and fixing Helios: An analysis of ballot secrecy Website with description and video at http://www.bensmyth.com/publications/10-attacking-helios/ (Cryptology ePrint Archive, Report 2010/625)
38. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 389–404. Springer, Heidelberg (2010)
39. Unruh, D., Müller-Quade, J.: Universally Composable Incoercibility. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 411–428. Springer, Heidelberg (2010)
40. Küsters, R., Truderung, T., Vogt, A.: A Game-Based Definition of Coercion-Resistance and its Applications. In: Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF 2010), pp. 122–136 (2010)
41. Loftus, J., May, A., Smart, N.P., Vercauteren, F.: On CCA-Secure Fully Homomorphic Encryption, http://eprint.iacr.org/2010/560

42. Cortier, V., Smyth, B.: Attacking and fixing Helios: An analysis of ballot secrecy. To appear in: Proceedings of the 24th Computer Security Foundations Symposium, CSF 2011 (2011)
43. Küsters, R., Truderung, T., Vogt, A.: Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. To appear at the 32nd Security and Privacy Symposium, S&P 2011 (2011) (preprint)
44. Persiano, G.: About the Existence of Trapdoors in Cryptosystems. Work in Progress, http://libeccio.dia.unisa.it/Papers/Trapdoor/
45. Helios voting. Website, http://heliosvoting.org
46. Helios Headquarters, Princeton University Undergraduate Student Government, http://usg.princeton.edu/officers/elections-center/helios-headquarters.html

Remote Timing Attacks Are Still Practical[*]

Billy Bob Brumley and Nicola Tuveri

Aalto University School of Science, Finland
{bbrumley,ntuveri}@tcs.hut.fi

Abstract. For over two decades, timing attacks have been an active area of research within applied cryptography. These attacks exploit cryptosystem or protocol implementations that do not run in constant time. When implementing an elliptic curve cryptosystem with a goal to provide side-channel resistance, the scalar multiplication routine is a critical component. In such instances, one attractive method often suggested in the literature is Montgomery's ladder that performs a fixed sequence of curve and field operations. This paper describes a timing attack vulnerability in OpenSSL's ladder implementation for curves over binary fields. We use this vulnerability to steal the private key of a TLS server where the server authenticates with ECDSA signatures. Using the timing of the exchanged messages, the messages themselves, and the signatures, we mount a lattice attack that recovers the private key. Finally, we describe and implement an effective countermeasure.

Keywords: Side-channel attacks, timing attacks, elliptic curve cryptography, lattice attacks.

1 Introduction

Side-channel attacks utilize information leaked during the execution of a protocol. These attacks differ from traditional cryptanalysis attacks since side-channels are not part of the rigorous mathematical description of a cryptosystem: they are introduced by implementation aspects and are not modeled as input and/or output of the cryptosystem. A timing attack is a side-channel attack that recovers key material by exploiting cryptosystem implementations that do not run in constant time: their execution time measured by the attacker is somehow state-dependent and hence key-dependent.

In light of these attacks, implementations of elliptic curve cryptosystems that execute in environments where side-channels are a threat seek to fix the execution time of various components in said implementation. Perhaps the most critical is that of scalar multiplication that computes the k-fold sum of a point with

[*] The work described in this paper has been supported in part by Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (Hecse), Academy of Finland (project #122736), and the European Commission through the ICT program under contracts ICT-2007-216499 CACE and ICT-2007-216676 ECRYPT II.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 355–371, 2011.

itself. Leaking any internal algorithm state during this computation can reveal information about some of the inputs, some of which should critically remain secret.

As a practical example of utilizing said key material, consider lattice attacks. Lattices are mathematical objects that have many uses in cryptography from cryptographic primitives to attacking schemes with partially known secret data. They are generally useful for finding small solutions to underdetermined systems of equations. Lattice methods are an effective endgame for many side-channel attacks: combining public information with (private) partial key material derived in the analysis phase, i.e., procured from the signal, to recover the complete private key. Repeatedly leaking even a small amount of ephemeral key material can allow these attacks to succeed at recovering long-term private keys.

Montgomery's ladder is a scalar multiplication algorithm that has great potential to resist side-channel analysis. The algorithm is very regular in the sense that it always executes the same sequence of curve and field operations, regardless of the value that a key bit takes. Contrast this with, for example, a basic right-to-left double-and-add scalar multiplication algorithm that only performs point additions on non-zero key bits.

This paper describes a timing attack vulnerability in OpenSSL's ladder implementation for elliptic curves over binary fields. The timings are procured by measuring the execution time of parts of the TLS handshake between an attacker client and OpenSSL's own TLS server where the server provides an ECDSA signature on a number of exchanged messages. We utilize this timing information to mount a lattice attack that exploits this vulnerability and recovers the ECDSA private key given a small number of signatures along with said timing data. We provide extensive experiment results that help characterize the vulnerability. Lastly, we propose, implement, and evaluate a simple and efficient countermeasure to the attack that proves effective.

The remainder of the paper is organized as follows. Section 2 reviews the concept of timing attacks and selective related literature. Section 3 contains background on elliptic curve cryptography and its implementation in OpenSSL. Section 4 identifies said vulnerability and describes all stages of the proposed attack. Section 5 contains the experiment and attack implementation results. We close in Section 6 with a discussion on countermeasures and draw conclusions.

2 Timing Attacks

P. Kocher gives a number of remarkably simple timing attacks in his seminal work [1]. Consider a right-to-left square-and-multiply algorithm for exponentiation. If the exponent bit is a 1, the algorithm performs the assignments $B := B \cdot A$ then $A := A^2$. Otherwise, a 0-bit and the algorithm performs only the assignment $A := A^2$. The attacker chooses operand A hence its value in each iteration is known. To mount a timing attack, the attacker is tasked with finding input A that distinguishes former cases from the latter. This could be done by choosing A such that the former case incurs measurably increased execution time over the entire exponentiation yet the latter case does not. Varying the number of computer words in A could be one method to induce this behavior. Starting with the least significant bit, the attacker repeats this process to recover the key iteratively. In this manner, the attacker traces its way through the states of the exponentiation algorithm using the timings as evidence. The author gives further examples of software mechanisms that lead to timing vulnerabilities as well as attack experiment results. The work mostly concerns public key cryptosystems with a static key such as RSA and static Diffie-Hellman.

D. Brumley and D. Boneh [2,3] present ground breaking results, demonstrating that timing attacks apply to general software systems, defying contemporary common belief. They mount a timing attack against OpenSSL's implementation of RSA decryption based on (counteracting but exploitable) time dependencies introduced by the Montgomery reduction and the multiplication routines used by the OpenSSL implementation. The key relevant fact about the Montgomery reduction is that an extra reduction step may be required depending on the input, while for the multi-precision integer multiplication routines (heavily used in RSA computation) the relevant fact is that one of two algorithms with different performances (Karatsuba and schoolbook) is used depending on the number of words used to represent the two operands. Exploiting these two facts and adapting the attack to work even when using the sliding window modular exponentiation algorithm, the authors devise an attack that is capable of retrieving the complete factorization of the key pair modulus.

The authors mount a real-world attack through a client that measures the time an OpenSSL server takes to respond to RSA decryption queries during the SSL handshake. The attack is effective between two processes running on the same machine and two virtual machines on the same computer, in local network environments and in case of lightly loaded servers. The authors also analyze experiments over a WAN and a wireless link to evaluate the effects of noise on the attacks. Finally, they devise three possible defenses and as a consequence several cryptography libraries including OpenSSL feature RSA blinding by default as a countermeasure. As a tangible result of their work:

- OpenSSL issued[1] a security advisory;
- CVE assigned[2] the name CAN-2003-0147 to the issue;
- CERT issued[3] vulnerability note VU#997481.

3 Elliptic Curve Cryptography

In the mid 1980s, Miller [4] and Koblitz [5] independently proposed the use of elliptic curves in cryptography. Elliptic curves are a popular choice for public key cryptography because no sub-exponential time algorithm to solve discrete logarithms is known in this setting for well-chosen parameters. This affords Elliptic

[1] `http://www.openssl.org/news/secadv_20030317.txt`

[2] `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0147`

[3] `http://www.kb.cert.org/vuls/id/997481`

Curve Cryptography (ECC) comparatively smaller keys and signatures. For the purposes of this paper, it suffices to restrict to curves of the form

$$E(\mathbb{F}_{2^m}) : y^2 + xy = x^3 + a_2x^2 + a_6$$

where $a_i \in \mathbb{F}_{2^m}$ and $a_2 = 1$ is common. NIST standardizes two types of curves for each $m \in \{163, 233, 283, 409, 571\}$:

1. a_6 is chosen pseudo-randomly: i.e., B-163.
2. $a_6 = 1$ and $a_2 \in \{0, 1\}$: Koblitz curves [6], i.e., K-163.

With Intel's recent `pclmulqdq` carry-less multiplication instruction facilitating multiplication in $\mathbb{F}_2[x]$, curves over binary fields are likely to become the standard choice for high-speed ECC implementations in software.

3.1 Digital Signatures

We use the following notation for the ECDSA. The parameters include a hash function h and point $G \in E$ that generates a subgroup of prime order n. In fact $\#E = cn$ where $c \in \{2, 4\}$ for the standard curves considered in this paper. A common current choice for these would be roughly a 160-bit n, i.e., computations on B-163 or K-163. Parties select a private key d uniformly from $0 < d < n$ and publish the corresponding public key $D = [d]G$. To sign a message m, parties select nonce k uniformly from $0 < k < n$ then compute the signature (r, s) by

$$r = ([k]G)_x \bmod n \tag{1}$$

$$s = (h(m) + dr)k^{-1} \bmod n. \tag{2}$$

This work omits the details of signature verification as they are not particularly relevant here. The performance bottleneck for generating these signatures is the scalar multiplication in (1). Extensive literature exists on speeding up said operation: a description of one common method follows.

3.2 Scalar Multiplication

The speed of an ECC implementation is essentially governed by the scalar multiplication operation that, for an integer k and point $P \in E$, computes the point $[k]P$. There are many methods to carry out this computation, but we focus on the Montgomery power ladder, originally proposed for speeding up integer factorization using the elliptic curve method [7, Sect. 10.3.1].

López and Dahab improve the algorithm efficiency for curves over binary fields [8]. Fig. 1 illustrates the main parts of the algorithm and is an excerpt from the implementation in OpenSSL 0.9.8o. The nested `for` loop is where the majority of the work takes place and performs one point doubling and one point addition to process one bit of k in each iteration; assume $k_i = 1$. The point addition formula, i.e. implemented in the `gf2m_Madd` function called in Fig. 1, is

$$(Z_0, X_0) = ((X_0 \cdot Z_1 + X_1 \cdot Z_0)^2, x \cdot Z_0 + (X_0 \cdot Z_1) \cdot (X_1 \cdot Z_0))$$

```
/* find top most bit and go one past it */
i = scalar->top - 1; j = BN_BITS2 - 1;
mask = BN_TBIT;
while (!(scalar->d[i] & mask)) { mask >>= 1; j--; }
mask >>= 1; j--;
/* if top most bit was at word break, go to next word */
if (!mask)
  {
  i--; j = BN_BITS2 - 1;
  mask = BN_TBIT;
  }

for (; i >= 0; i--)
  {
  for (; j >= 0; j--)
    {
    if (scalar->d[i] & mask)
      {
      if (!gf2m_Madd(group, &point->X, x1, z1, x2, z2, ctx)) goto err;
      if (!gf2m_Mdouble(group, x2, z2, ctx)) goto err;
      }
    else
      {
      if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx)) goto err;
      if (!gf2m_Mdouble(group, x1, z1, ctx)) goto err;
      }
    mask >>= 1;
    }
  j = BN_BITS2 - 1;
  mask = BN_TBIT;
  }
```

Fig. 1. Montgomery's ladder scalar multiplication for curves over binary fields as implemented in OpenSSL 0.9.8o at `crypto/ec/ec2_mult.c`.

and point doubling, i.e. implemented in the `gf2m_Mdouble` function called in Fig. 1, is

$$(Z_1, X_1) = ((X_1 \cdot Z_1)^2, X_1^4 + a_6 \cdot Z_1^4).$$

An intriguing feature is that when $k_i = 0$, the same steps are performed: only the operands are transposed. That is, replacing Z_1 with Z_0 and X_1 with X_0 describes the above formulae for a zero bit. This means the cost per bit is fixed at an impressive six field multiplications, one involving a constant. For curves over binary fields, OpenSSL uses this algorithm as the default for any single scalar multiplication, e.g., in signature generation, and in fact iterates it twice for the sum of two scalar multiplications, e.g., in signature verification.

The ladder applied to ECC has numerous advantages: fast computation, no large memory overhead, and a fixed sequence of curve operations. This last feature is particularly attractive as a side-channel countermeasure. The following quote concisely captures this [9, p. 103].

> Another advantage is that the same operations are performed in every iteration of the main loop, thereby potentially increasing resistance to timing attacks and power analysis attacks.

While this feature cannot be denied, the quoted authors duly qualify the statement with *potentially*: the side-channel properties are those of the algorithm implementation, not the algorithm itself. It should be noted that the ladder was originally proposed only for efficient computation. Its potential to resist side-channel analysis seems to be an unintentional consequence.

4 A Timing Attack

The ladder implementation in Fig. 1 introduces a timing attack vulnerability. Denote the time required to process one scalar bit and compute one ladder step as t: that is, one iteration of the nested `for` loop that performs the double and add steps. Said time is (reasonably) independent of, for example, any given bit k_i or the Hamming weight of k. On the other hand, consider the preceding `while` loop: its purpose is to find the index of the most significant set bit of k and optimize the number of iterations of the nested `for` loop. As a result, there are exactly $\lceil \lg(k) \rceil - 1$ ladder step executions and the time required for the algorithm to execute is precisely $t(\lceil \lg(k) \rceil - 1)$. This shows that there is a direct correlation between the time to compute a scalar multiplication and the logarithm of k.

This section describes an attack exploiting this vulnerability. The attack consists of two phases.

1. The attacker collects a certain amount of signatures and exploits the described time dependency to filter a smaller set of signatures. The signatures in the filtered set will have a high probability of being generated using secret nonces (k) having a leading zero bits sequence whose length is greater or equal to a fixed threshold.
2. The attacker mounts a lattice attack using the set of signatures filtered in the collection phase to recover the secret key used to generate the ECDSA signatures.

For this attack to succeed, we assume to be able to collect a sufficient amount of ECDSA signatures made under the same ECDSA key, and to be able to measure, with reasonably good accuracy, the wall clock execution time of each collected sign operation. For concreteness we focus on the NIST curve B-163, but the concepts can be more generally applied for any curve over a binary field, and furthermore to any scalar multiplication implementation with a main loop that has a constant iteration time but not a constant number of iterations.

4.1 Overview of the Collection Phase

To verify and evaluate the actual exploitability of the described time dependency for mounting a practical side-channel attack, we implemented two different versions of the collection phase that share the same basic structure and differ only for the sequence of operations used to perform a signature:

– a “local” attack, where the collecting process directly uses the OpenSSL ECDSA routines, accurately measuring the time required by each sign operation; this version models the collection phase in ideal conditions, where noise caused by external sources is reduced to the minimum;
– a “remote” attack, where the collecting process uses the OpenSSL library to perform TLS handshakes using the ECDHE_ECDSA suite; this version models a real-world use case for this vulnerability and allows to evaluate how practical the attack is over different network scenarios.

In general, regardless of the internal implementation, the sign routines of both versions simply return a signature, a measure of the time that was required to generate it, and the digest value fed to the sign algorithm. The collecting process repeatedly invokes the sign routine and stores the results in memory using a fixed-length binary tree heap data structure, where the weight of each element is represented by the measured time and the root element contains the maximum, using the following algorithm:

```
Heap h=Heap.new(s); //fixed size=s
from 1 to t:
  Result res=sign_rtn(dgst, privk);
  if ( !h.is_full() ):
    h.insert(res); //O(log n) time
  else if ( res.t < h.root().t ):
    h.root()<-res; //O(1) time
    h.percolate_down();//O(log n) time
  else:
    ; //discard res
```

With this algorithm we are able to store the smallest (in terms of time) s results using a fixed amount of memory and less than $O(t(1+\lg(s)))$ time in the worst case; the total number of signatures collected (t) and the size of the filtered set (s) are the two parameters that characterize the collection process. As we expect the fastest signatures to be related to nonces with a higher number of leading zeros, we can use the ratio t/s to filter those signatures associated with leading zero bits sequences longer than a certain threshold.

Statistically for a random $1 \leq k < n$ with overwhelming probability the most significant bit will be set to zero since n for B-163 (and indeed many curves over binary fields) is only negligibly over a power of two. For the next leading bits the probability of having a sequence of zero bits of length j is equal to 2^{-j}. Hence if the total amount of collected signatures t is large enough, the set composed of the quickest s results should contain signatures related to nonces with leading zero bits sequences of length longer than $\lg(t/s)$, that are then fed to the lattice attack phase.

4.2 Collection Phase in Ideal Conditions

This version of the collecting process was implemented to verify that the described time dependency is actually exploitable for mounting a side-channel attack. We directly invoked the OpenSSL ECDSA routines from within the collecting process to generate ECDSA signatures of random data, accurately measuring the time required by each sign operation.

The high resolution timings were taken using the `rdtsc` instruction provided in recent Pentium-compatible processors. As the host CPU used for testing was dual core and supported frequency scaling, to ensure accuracy of the measurements we disabled frequency scaling and forced the execution of the collecting process on just one core.

As the time needed to generate a signature does not depend on the value of the message digest, for simplicity and to speed up the experiments we chose to generate multiple signatures on the same message, precalculating the message digest just once to avoid generating a new random message for each signature.

The implemented sign routine takes as input the digest of the message to be signed and the private key, and returns the computed signature, the time required to compute the ECDSA signature and the number of leading zero bits in the nonce. The latter value is obviously not used to mount the actual attack, but used only to verify the dependency between the execution time of the signature computation and the number of leading zero bits in the nonce.

4.3 Collection Phase over TLS

This implementation of the collecting process was developed to show a relevant real-world use case for this vulnerability and to evaluate its practicality in different network scenarios.

Here the signatures collected are those generated during the TLS handshake using the ECDHE_ECDSA cipher suite illustrated by Fig. 2. We briefly highlight the relevant features of the messages exchanged during the portion of the handshake targeted by this attack, referring to RFC 4492 [10] for the normative and detailed technical description of the full protocol handshake:

- The Client initiates the handshake sending a ClientHello message to the Server; this is a regular TLS ClientHello message, proposing the ECDHE_ECDSA cipher suite and intended to inform the Server about the supported curves and point formats. This message contains a random nonce generated by the Client.
- The Server replies with a ServerHello message, selecting the proposed ECDHE_ECDSA cipher suite and using an extension to enumerate the point formats it is able to parse. This message contains a random nonce generated by the Server.
- The Server sends a Certificate message, conveying an ECDSA-signed certificate containing the ECDSA-capable public key of the Server, and possibly a certificate chain.
- The Server sends a ServerKeyExchange message, conveying the ephemeral ECDH public key of the Server (and the relative elliptic curve domain parameters) to the Client. This message is divided in two halves, the first one containing the Server ECDH parameters (namely the EC domain parameters and the ephemeral ECDH public key, consisting of an EC point)

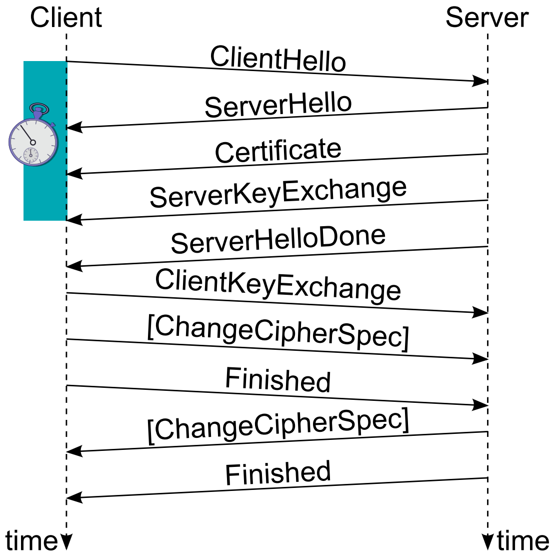


Fig. 2. TLS Handshake using the ECDHE_ECDSA suite described in RFC 4492

and the latter consisting of a digitally signed digest of the exchanged parameters. The digest is actually computed as SHA(ClientHello.random + ServerHello.random + ServerKeyExchange.params), and the signature is an ECDSA signature generated using the Server's private key associated with the certificate conveyed in the previous message.

- The handshake then continues, but other messages do not influence the implemented attack.

This version of the collecting process uses the OpenSSL library to perform an ECDHE_ECDSA TLS handshake every time a signature is requested. The sign routine creates a new TLS socket to the targeted IP address, configured to negotiate only connections using ECDHE_ECDSA and setting a message callback function that is used to observe each TLS protocol message. After creating the TLS socket the sign routine simply performs the TLS handshake and then closes the TLS connection. During the handshake the message callback inspects each TLS protocol message, starting a high resolution timer when the ClientHello message is sent and then stopping it upon receiving the ServerKeyExchange message, which is then parsed to compute the digest fed to the sign algorithm and to retrieve the generated signature.

In designing this attack, we assumed to be unable to directly measure the actual execution time of the server-side signature generation, hence we are forced to use the time elapsed between the ClientHello message and the ServerKeyExchange message as an approximation. To assess the quality of this approximation, the collecting process takes the private key as an optional argument. If provided, the message callback will also extrapolate the nonce used internally by the server to generate the signature and will report the number of leading zero bits in it.

Lastly, at first glance it seems possible that the Server's computation of its ECDHE key also influences the measured time. When creating an SSL context within an application, the default behavior of OpenSSL is to generate a key pair and buffer it for use before any handshake begins. This is done to improve efficiency. OpenSSL's internal `s_server` used in these experiments behaves accordingly, so in practice that step of the handshake does not affect the measured time since only one scalar multiplication takes place server-side during these handshake steps, namely that corresponding to the ECDSA signature. Applications can modify this behavior by passing options to the SSL context when creating it. This is a moot point when attacking ECDH_ECDSA modes.

4.4 The Lattice Attack

Using lattice methods, Howgrave-Graham and Smart show how to recover a DSA key from a number of signatures under the same key where parts of the nonces are known [11]. For completeness, a discussion on implementing the lattice attack follows. Observing j signatures, rearranging (2) yields j equations of the form

$$m_i - s_i k_i + d r_i \equiv 0 \pmod{n} \tag{3}$$

for $1 \leq i \leq j$ where here m_i are message digests to simplify notation. Using one such (3) to eliminating the private key yields $j-1$ equations of the form

$$k_i + A_i k_j + B_i \equiv 0 \pmod{n} \tag{4}$$

for $1 \leq i < j$ and some $0 \leq A_i, B_i < n$. From here, the equations in [11] simplify greatly since all the known bits are in the most significant positions and are in fact all zeros: (4) should be considered the same as Equation 3 of [11]. That is, express the nonces as $k_i = z_i' + 2^{\lambda_i} z_i + 2^{\mu_i} z_i''$ in their notation but all λ_i are zero setting all z_i' to zero and from the timings deducing all $\mu_i = 156$ (for example) setting all z_i'' to zero, leaving z_i as the only unknown on the right where in fact $k_i = z_i$. This is nothing more than a rather laborious way of expressing the simple fact that we know $\lg(k_i)$ falls below a set threshold. Consider a j-dimensional lattice with basis consisting of rows of the following matrix.

$$\begin{pmatrix} -1 & A_1 & A_2 & \dots & A_{j-1} \\ 0 & n & 0 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & n \end{pmatrix}$$

From here, the implementation uses the Sage software system to produce a reduced basis for this lattice using the LLL algorithm [12], orthogonalize this basis by the Gram-Schmidt process, and approximate the closest vector problem given input vector $(0, B_1, B_2, \dots, B_{j-1})$ using Babai rounding [13]. This hopefully finds the desired solutions to the unknown portions of the k_i.

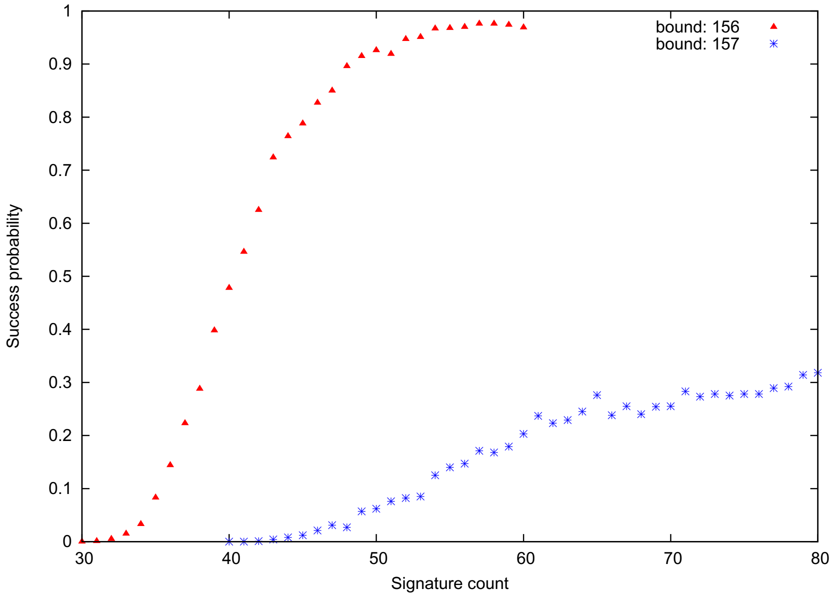


Fig. 3. Selective lattice attack parameters and observed success probabilities

Figure 3 contains experiment results of running the lattice attack with different parameters based on B-163, assuming upper bounds on $\lceil \lg(k_i) \rceil$ of $\mu_i \in \{156, 157\}$. The x-axis is the signature count (j) and the y-axis the observed lattice attack success probability. It shows that as the amount of known key material decreases (μ_i increases), this mandates an increase in the lattice dimension j (the number of such required signatures j increases), and the approximations are less likely to hold. To effectively apply this as part of the timing attack, on one hand the lower we set μ_i the less likely it is that a k_i will satisfy the bound and more signatures must be collected. On the other hand, collecting more signatures increases the probability of error in the measurements, i.e., incorrectly inferring a given signature with a low timing has a k_i that satisfies the bound and the lattice attack is more likely to fail. An interesting property of this particular lattice attack is that in fact μ_i does not feature in the equations used to populate the basis matrix. In practice, this means that even if some k_i does not satisfy the bound there is still a chance the attack will succeed.

5 Results

5.1 Collection Phase Parameters

Using the first implementation of the collecting process, we were able to empirically verify the dependency between the length of the leading zero bits sequence in the nonce and the execution time of the signature operation. Figure 4 compares the distributions of the execution time required by signatures generated

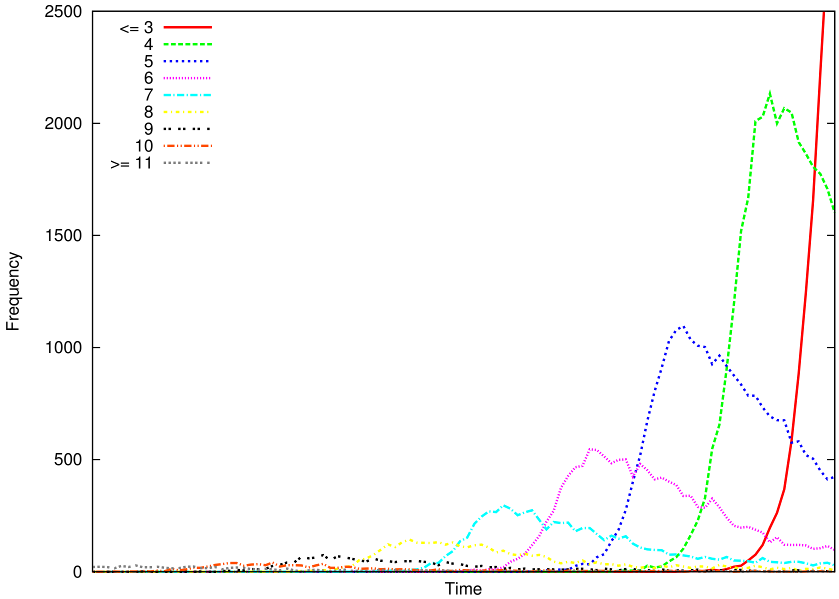


Fig. 4. Dependency between number of leading zero bits and wall clock execution time of the signature operation

using nonces with different leading zero bit sequence lengths. In the lattice attack notation, seven leading zero bits corresponds to $\mu_i = 156$ and six leading zero bits $\mu_i = 157$.

We then evaluated the effectiveness of the method used for filtering the signatures related to nonces with longer leading zero bit sequences by using different values for the collection phase parameters. The lattice attack phase, which takes as input the output of the collecting process, determines the size of the filtered set and the minimum length of the leading zero bit sequence of the nonce associated with the signature. Fixing the filtered set size to 64 signatures and varying the total number of signatures collected by the collecting process, we evaluated the number of "false positives" over multiple iterations, i.e., those signatures in the filtered set generated using nonces whose leading zero bit sequence length is below the threshold determined by the tuning of the lattice attack phase. Table 1 summarizes the obtained results and shows that the effectiveness of the filtering method may be adjusted by varying the t/s ratio.

The number of "false positives" in the filtered set is an important parameter of the attack. The lattice attack phase has a higher success probability if all the signatures used to populate the matrix satisfy the constraint on the number of leading zero bits. But as mentioned in Sec. 4, even in the presence of limited "false positives" the lattice attack still succeeds with a small probability.

Consulting Fig. 3, setting the threshold on the minimum number of leading zero bits to 7 we only needed 43 valid signatures to successfully perform the lattice attack with high probability. Naïvely, this allows up to 21 "false positives"

Table 1. Observed results of the local attack

Collected signatures count (t)	4096	8192	16384
Filtered set size (s)	64	64	64
Average “false positives” count	17.92	1.48	0.05

in the filtered set obtained from the collecting process using the lattice attack in a more fault-tolerant way:

```
Signatures[] filtered_set; // <-- collection_phase()
EC_point known_pubkey; // <-- server certificate

while(True)
{
  tentative_privkey=lattice_attack(filtered_set[0:43]);
  tentative_pubkey=generate_pub_key(tentative_priv_key);
  if ( tentative_pub_key == known_pubkey )
    break;  // successfully retrieved the priv key
  randomly_shuffle(filtered_set);
}
```

What follows is a rough estimate for the number of required lattice attack iterations in the presence of “false positives”. The number of iterations, and thus the computation time, needed to correctly retrieve the private key is inversely proportional to the probability of selecting a subset of the filtered set without “false positives”:

$$\Pr[\text{subset without “false positives”}] = \frac{\binom{64-e}{43}}{\binom{64}{43}}$$

where e is the number of “false positives” in the filtered set, 43 is the size of the subsets, 64 is the size of the filtered set, the numerator is the number of possible subsets without “false positives” in the filtered set, and the denominator is the number of possible subsets in the filtered set. Figure 5 shows how this probability varies with e.

5.2 Remote Attack

We used the described “remote” implementation of the collecting process to attack TLS servers over two different network scenarios. As a reference server we used the OpenSSL `s_server` configured to emulate a TLS-aware web server using an ECDSA-capable private key. In theory, any server using the OpenSSL ECDSA implementation to support ECDHE_ECDSA TLS can be vulnerable.

The first scenario we considered consists of a collecting process running on the same host as the server process. The messages are exchanged over the OS TCP/IP stack using the localhost address on the loopback interface. In this scenario we successfully retrieved the server private key, even repeating the tests

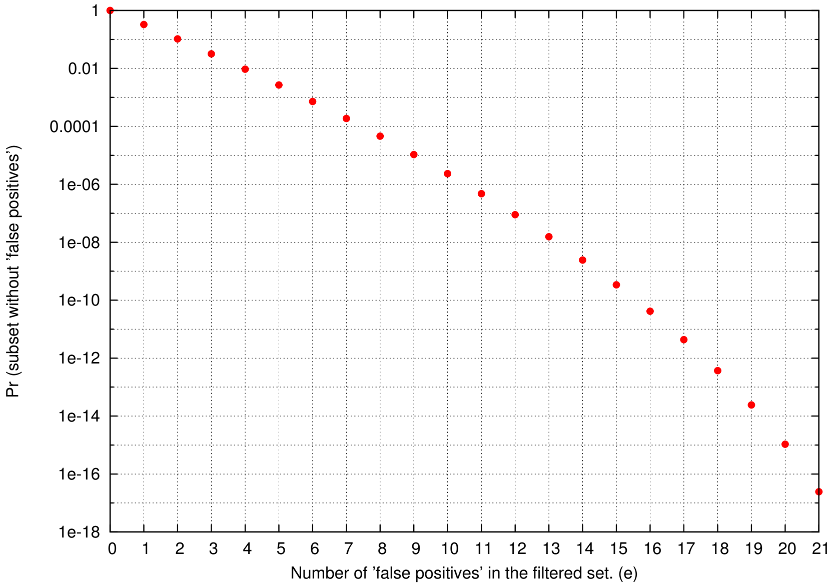


Fig. 5. Probability of selecting a subset without "false positives" in a filtered set with e "false positives"

using different private keys, randomly generated using OpenSSL itself, and targeting both the OpenSSL 0.9.8o and 1.0.0a versions of the reference server.

Table 2 shows that, even if unable to directly measure the execution time of the signature computation, using the measure of the time elapsed between the ClientHello and the ServerKeyExchange messages as an approximation and tuning the total number of collected signatures, the attacker is able to filter a set of signatures with a low average of "false positives".

We also note that in the "remote" attack, only the collection phase is performed online, as the lattice attack phase does not require the attacker to exchange messages with the attacked server, and that even collecting a total of 16384 signatures is not particularly time consuming, requiring just a few minutes.

Once verified that the attack is practical when run over the loopback interface on the same host of the attacked server, we performed the same attack in a slightly more complex network scenario: the attacker collects the signatures from a host connected to the same network switch of the server. The tests were run between two hosts residing in the same room in time frames with reasonably low network loads, trying to minimize the noise introduced by external causes in the time measures.

From Table 3 we see that the time dependency is still observable. The average rates of "false positives" in the filtered sets increases, but from Fig. 5 this is still easily within reach. The lattice attack can take hours to run in this case, but again the work is offline and can be distributed. In some cases we achieved success

Table 2. Observed results of the remote attack over the loopback interface

Collected signatures count (t)	4096	8192	16384
Filtered set size (s)	64	64	64
Average "false positives" count	17.06	4.01	0.90

Table 3. Observed results of the remote attack over a switched network segment

Collected signatures count (t)	4096	8192	16384
Filtered set size (s)	64	64	64
Average "false positives" count	19.40	8.96	11.81

in only a few minutes. We also note that in this particular network environment we cannot arbitrarily decrease the "false positives" rate by increasing the parameter t, as already with $t = 16384$ the average number of "false positives" starts to increase.

This demonstrates the feasibility of the attack in a remote scenario. Naturally, individual results will vary due to different network characteristics. The attack success rate decreases dramatically with the increase of "false positives". Regardless, a vulnerability exploitable to perform a successful attack from the same host where the targeted server is run poses a threat even for remote attacks. For example, in virtual hosting or cloud computing scenarios an attacker may be able to obtain access to run code on the same physical machine hosting the target server, as suggested by [14].

6 Conclusion

This paper identifies a timing attack vulnerability in OpenSSL's implementation of Montgomery's ladder for scalar multiplication of points on elliptic curves over binary fields. This is used to mount a full key recovery attack against a TLS server authenticating with ECDSA signatures. In response to this work, CERT issued[4] vulnerability note VU#536044. Ironically, in the end it is the regular execution of the ladder that causes this side-channel vulnerability. For example, a dependency on the weight of k (that might leak from, say, a simple binary scalar multiplication method) seems much more difficult to exploit than that of the length of k that led to full key recovery here.

The work of D. Brumley and D. Boneh [2,3] and this work are related in that both exploit implementation features that cause variable time execution, and that both demonstrate full key recovery in both local and remote scenarios. However, the fundamental difference with the former is that the attacker can leverage well-established statistical techniques and repeat measurements to compensate for noise because the secret inputs are not changing, i.e., the RSA exponent. Contrasting with the latter, the secret inputs are always distinct, i.e.,

[4] http://www.kb.cert.org/vuls/id/536044

the nonces in ECDSA. The former is a stronger attack than the latter in this respect.

Lastly, a brief discussion on countermeasures follows. One way to prevent this attack is by computing $[k]G$ using the equivalent value $[\hat{k}]G$ where

$$\hat{k} = \begin{cases} k + 2n & \text{if } \lceil \lg(k+n) \rceil = \lceil \lg n \rceil, \\ k + n & \text{otherwise.} \end{cases}$$

With $\#\langle G \rangle = n$ then $[k]G = [\hat{k}]G$ holds and the signature remains valid. This essentially changes $\lceil \lg(\hat{k}) \rceil$ to a fixed value. We implemented this approach as a patch to OpenSSL and experiment results show that applying said padding thwarts this particular attack and does not entail any performance overhead to speak of. Note that the scope of this paper does not include microarchitecture attacks for which additional specific countermeasures are needed.

This work further stresses the importance of constant time implementations and rigorous code auditing, adding yet another entry to an already long list of cautionary tales surrounding cryptography engineering.

References

1. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
2. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: Proceedings of the 12th USENIX Security Symposium (2003)
3. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks 48, 701–716 (2005)
4. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
5. Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48, 203–209 (1987)
6. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
7. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. 48, 243–264 (1987)
8. López, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
9. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
10. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B.: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational) (2006) (updated by RFC 5246)
11. Howgrave-Graham, N., Smart, N.P.: Lattice attacks on digital signature schemes. Des. Codes Cryptography 23, 283–290 (2001)
12. Lenstra, A.K., Lenstra, J. H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. 261, 515–534 (1982)

13. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica 6, 1–13 (1986)
14. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 199–212. ACM, New York (2009)

A Countermeasure as OpenSSL Source Code Patch

```
--- openssl-0.9.8o-orig/crypto/ecdsa/ecs_ossl.c 2009-12-01 19:32:16.000000000 +0200
+++ openssl-0.9.8o-patch/crypto/ecdsa/ecs_ossl.c 2011-06-08 11:23:41.188104470 +0300
@@ -144,6 +144,13 @@
  }
  while (BN_is_zero(k));

+ /* We do not want timing information to leak the length of k,
+  * so we compute G*k using an equivalent scalar of fixed
+  * bit-length. */
+ if (!BN_add(k, k, order)) goto err;
+ if (BN_num_bits(k) <= BN_num_bits(order))
+ if (!BN_add(k, k, order)) goto err;
+
  /* compute r the x-coordinate of generator * k */
  if (!EC_POINT_mul(group, tmp_point, k, NULL, NULL, ctx))
  {
```

Multi-run Security

Arnar Birgisson and Andrei Sabelfeld

Chalmers University of Technology, 412 96 Gothenburg, Sweden

Abstract. This paper explores information-flow control for batch-job programs that are allowed to be re-run with new input provided by the attacker. We argue that directly adapting two major security definitions for batch-job programs, termination-sensitive and termination-insensitive noninterference, to multi-run execution would result in extremes. While the former readily scales up to multiple runs, its enforcement is typically over-restrictive. The latter suffers from insecurity: secrets can be leaked in their entirety by multiple runs of programs that are secure according to batch-job termination-insensitive noninterference. Seeking to avoid the extremes, we present a framework for specifying and enforcing multi-run security in an imperative language. The policy framework is based on tracking the attacker's knowledge about secrets obtained by multiple program runs. Inspired by previous work on robustness, the key ingredient of our type-based enforcement for multi-run security is preventing the dangerous combination of attacker-controlled data and secret data from affecting program termination.

1 Introduction

Imagine a scenario of a web service with a medical database at the back-end. Analysts are allowed to access the database through a web interface. The goal is to allow deriving interesting statistics (say, by age groups or by larger residential areas) but disallow leaking sensitive information about individuals. In this scenario, the server-side program that accommodates queries has two inputs: one is the database itself, which contains sensitive data and which is not controlled by the attacker, and the other one is a public query that originates from a possibly malicious analyst. For the program to function, it must have access to the entire database. At the same time, it must not reveal sensitive data about individual entries in the database. Hence, we are interested in securing *information flow* from secret inputs to public outputs. This problem arises both when the code is written by non-malicious developers, in which case we want prevent accidental leaks, and when the code is supplied by untrusted third parties, when we want to prevent malicious leaks. Settling for the worst case, we do not appeal to trust assumptions.

Language-based information-flow security [31] is focused on providing strong security guarantees for underlying programs. In the context of confidentiality, it is intended to prevent information flow from secret inputs to public outputs. The dominating baseline security policy is *noninterference* [14, 19] that requires that a variation of secret input does not result in a variation of public outputs.

However, the state of the art in the area consists of two extremes. One extreme is batch-job program models, where programs are run only once and where the initial

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 372–391, 2011.

memory is the only input and the final memory is the only output. A large body of research on language-based information-flow security is limited to batch-job models. In a language-based setting, noninterference has been largely considered for batch-job models [40, 31]. Major efforts on information flow in functional [28], object-oriented [43, 8, 20], concurrent [37, 42, 30], and other languages [31] assume a batch-job model.

While securing batch-job programs without being over-restrictive is feasible, the assumption that programs are run only once is often too strong. The other extreme is fully interactive programs with channels for input/output communication. While this model is more powerful, securing interactive programs is notoriously hard: intermediate observations can be exploited to leak information [2].

This paper explores middle ground between the extremes: batch-job programs that are allowed to be re-run with new input provided by the attacker. We believe this model captures many practical scenarios such as the medical database above. Our attacker model allows issuing queries to the database as described by a batch-job program whose secret input is the database and public input is the attacker-controlled part of the query. The goal is to prevent the attacker from learning sensitive information by re-running the program with modified public parameters and observing the public outcome.

Leaks via termination behavior of programs turn out to be the bottleneck for generalizing batch-job style security to multiple runs. We argue that directly adapting two major security definitions for batch-job programs, termination-sensitive and termination-insensitive noninterference, to multi-run execution would result in further extremes. The former, *termination-sensitive noninterference* [39, 31], readily scales up to multiple runs. This definition demands that the public outcome and termination behavior of the underlying program do not depend on secret data. No run leaks any information about secrets, and so we can safely re-run programs. Thus, batch-job termination-sensitive security implies multiple-run security. However, enforcing termination-sensitive noninterference withing being overly restrictive is far from trivial. Typically, enforcement mechanisms (e.g., [39]) place Draconian restrictions whenever abnormal termination is possible in sensitive context. For example, no sensitive data is allowed in loop guards.

The latter, *termination-insensitive noninterference* [40, 31], where secrets are allowed to affect termination behavior, suffers from insecurities in the multi-run case. Let us illustrate the problem with examples. The program

$$\texttt{while}\ h\ \texttt{do skip} \tag{1}$$

where h contains a secret, is deemed secure. Termination-insensitive noninterference quantifies over all possible input memories that agree on the public part and makes sure that terminating runs agree on the public part of the final memories. The termination behavior is not considered to have a significant effect, even though the termination depends on secret data. Although the condition quantifies over possible runs, its guarantees are only about differences between two single runs. The implicit assumption is that the program is run only once. A common argument is that if a batch-job program that satisfies termination-insensitive noninterference is run only once, then it leaks at most one bit [2].

With the same rationale, flavors of this program are also accepted by mainstream information-flow security tools Jif [26], FlowCaml [35], and the SPARK Examiner [9, 12] for Java, Caml, and Ada, respectively.

Similarly, the program

$$\texttt{while } h = l \texttt{ do skip} \tag{2}$$

where h contains a secret and l is an attacker-controlled public variable, is also considered secure.

However, the single-run assumption is in many cases inadequate. As in the database scenario above, attackers are often capable of re-running the program. Further, in a smartcard setting, the attacker may try to leak the secret key by multiple attempts of feeding public inputs and observe the properties of public outputs. A web attacker can initiate multiple runs of a server-side computation that involves secrets by providing a request with public input. Similarly, the attacker can initiate multi-run computation on the client side of an honest user by providing scripts that keep re-running after recovering from divergence (rather straightforward to accomplish with the modern browsers' interpretation of JavaScript). A recent exploit of ASP.NET analyzes the difference between error messages of multiple requests to collect information for a padding oracle attack [29].

This ability does not make a difference for program 1 (given the value of the secret is unchanged between the runs), but it is fatal for program 2: the attacker can learn the entire secret by brute-force guessing the value of h with different choices for l. Multi-run leaks are particularly devastating for single-run secure programs like:

$$\texttt{while } h\&\&l \texttt{ do skip} \tag{3}$$

where $\&\&$ is the bitwise "and" operation. By walking through the bits of h in subsequent runs, the attacker can learn the entire value in linear time (of the bit-size of the secret) bit-by-bit. Thus, secrets can be leaked in their entirety by multiple runs of programs that are single-run secure. A quick experiment with a Jif-certified program that contains a termination leak of this kind shows that it is straightforward to leak one secret bit per second even on a modest modern desktop machine (tested with Jif 3.0). This implies that a credit card number can be leaked within a minute. The Jif program and a simple Python script that exploits its termination leak are shown in Appendix B.

Seeking to avoid the extremes, we present a framework for specifying and enforcing multi-run security. For specification, we are inspired on knowledge-based attacker models [17, 4]. The policy framework is based on tracking the attacker's knowledge about secrets obtained by multiple program runs. The multi-run setting for such a framework is novel. The framework supports possibilities for intended information release (illustrated by examples below). Further, it connects to quantitative security, where we reason about how many bits of information can be leaked by multiple program runs.

For enforcement, we are inspired by previous work on robustness [41, 25, 3]. The key ingredient of our type-based enforcement for multi-run security is preventing the dangerous combination of attacker-controlled data and secret data from affecting program termination. It is particularly gratifying that we can draw on the type system for robustness for enforcing a policy that it has not been designed for. This connection leads us to clean enforcement, providing a simple solution to a nontrivial problem of multi-run security.

For information-flow tracking, we deploy data labels that combine confidentiality and integrity information. Confidentiality distinguishes secret information from public by *high* and *low* confidentiality labels. Integrity distinguishes untrusted information from trusted by *low* and *high* integrity labels. For confidentiality the use of high information is more restrictive: secrets may not leak to public; and dually for integrity use of low is restricted: untrusted data may not affect trusted. Typically, lattices [16] are used to reason about more complex structures than low/high for confidentiality and integrity. Of particular interest of us are product lattices that combine confidentiality and integrity labels. In the example of a product lattice that combines two low/high lattices, the top element is high confidentiality and low integrity. Data at this level is most restrictive to use. The bottom element is low confidentiality and high integrity, which may arbitrarily affect data at other levels. Integrity plays a key role for the enforcement: the enforcement ensures that combinations of high-confidentiality (secret) and low-integrity (attacked-controlled) data do not affect the termination behavior.

As foreshadowed above, we extend our approach to specify and track intentional information release (or *declassification*). The extended enforcement guarantees that the program does not release more information than described by *escape-hatch* [32] expressions. The purpose of escape hatches is to describe what is allowed to be released. The job of the underlying security condition is to ensure than *nothing else* about secret data may be learned by the attacker. For example, program

$$l := h\%4 \tag{4}$$

releases two least-significant bits about the secret variable h. When this is desired, it is expressed in our framework by the escape-hatch expression $h\%4$. The type system accommodates intentional release by labeling escape-hatch expressions as low confidentiality and high integrity. Hence, the program above is accepted while, for example, program

$$l := h\%6 \tag{5}$$

is rejected because the type system detects a mismatch with the escape hatch $h\%4$.

Next, assuming the same escape-hatch policy $h\%4$, consider the following program:

$$\texttt{while}\ h\%4 + l\ \texttt{do skip} \tag{6}$$

This program may also release two least-significant bits about h. Indeed, the attacker may experiment by supplying inputs -2, -1, and 0 for l and observing whether the program diverges. Our type system rightfully accepts this program because the loop guard $h\%4 + l$ has low integrity and low confidentiality, inheriting its restrictions from variable l (recall that $h\%4$ is labeled as low confidentiality and high integrity, which is least restrictive).

A final example illustrates how intended declassification is distinguished from unintended. Assuming the escape-hatch policy h, consider the program

$$h := h'; l := h \tag{7}$$

that attempts to leak the initial value of h' by *laundering* its value through the declassified (syntactic) variable h. This program is rejected because the enforcement mechanism detects that a variable involved in declassification has been modified.

2 Security Condition

This section presents some key definitions, in particular the definition of when we consider programs multi-run secure. Command c represents a deterministic program in the rest of the paper. As before, h and l represent secret (*high*) and public (*low*) variables. Without loss of generality, we treat a program as a function of two inputs (secret and public) coming from some finite domain D, to the set $D \cup \{\bot\}$. The result $c(h, l)$ expresses the observed low output of the program, with the special value $\bot$ representing nontermination.

Definition 1 (Single-run knowledge). *Let c be a program taking two inputs, a fixed secret one v_h and a (non-fixed) public one v_l each from some domain D, and yielding a public output $c(v_h, v_l) \in D \cup \{\bot\}$.*

An attacker (with full knowledge of c itself) is allowed to execute c, providing the public input v_l and observing only the public output $c(v_h, v_l)$. The attacker's knowledge *of the (fixed) secret input is then represented by the set of values that would lead to the observed outcome. This set is written as:*

$$k_{v_h}(c, v_l) = \{\, x \in D \mid c(x, v_l) = c(v_h, v_l) \,\}$$

Programs with more than two inputs are modeled by collecting all secret and public inputs into two separate tuples, and similar for programs which have more than one output.

Note that by allowing $c(v_h, v_l)$ to take the special value $\bot$ (meaning that c has an infinite derivation for those inputs), we make nontermination observable. This is important because in reality, nontermination can for example be (approximately) inferred from programs that time out.

Assume an attacker has some previous knowledge of h, represented by the set $K_0 \subseteq D$. Then the attacker is potentially able to increase that knowledge (which corresponds to shrinking the set of possibilities) by running the program. The attacker's new knowledge will be the single-run knowledge intersected with the previous knowledge. Repeating this process (possibly with different low inputs) results in a sequence of increasing knowledge (decreasing sets of possibilities). For an attacker with no initial knowledge we can simply start with $K_0 = D$. Obviously, a program may potentially leak more if the attacker gets the chance to invoke it multiple times and has control over some of the input.

The maximum knowledge attainable by the attacker is the result of the above process repeated for every possible low input. Note that this models a powerful attacker, as the number of possibilities is exponential in the bit-size of the input. This maximum knowledge, or *multi-run knowledge* is now defined as follows.

Definition 2 (Multi-run knowledge). *Let c, v_h, v_l, and D be as in Definition 1. The attacker's knowledge about v_h produced by multiple runs of program c is defined as:*

$$K_{v_h}(c) = \bigcap_{v_l \in D} k_{v_h}(c, v_l)$$

To highlight the contrast between multi-run knowledge and single-run knowledge captured by the definitions, we come back to the examples from Section 1. Assume $D = \{0, \ldots, 255\}$. Recall program 1:

$$\texttt{while}\ h\ \texttt{do skip}$$

The single-run knowledge $k_{v_h}(c, v_l)$ for this program is $\{0\}$, when $c(v_h, v_l) = v_l$, and $\{1, \ldots, 255\}$, when $c(v_h, v_l) = \bot$. The multi-run knowledge $K_{v_h}(c)$ is $\{0\}$, when $v_h = 0$, and $\{1, \ldots, 255\}$, when $v_h \neq 0$, which directly corresponds to the two cases for the single-run knowledge. Recall now program 2:

$$\texttt{while}\ h = l\ \texttt{do skip}$$

The single-run knowledge $k_{v_h}(c, v_l)$ for this program is $\{0, \ldots, v_l - 1, v_l + 1, \ldots, 255\}$, when $c(v_h, v_l) = v_l$, and $\{v_l\}$, when $c(v_h, v_l) = \bot$. However, the multi-run knowledge $K_{v_h}(c)$ is simply $\{v_h\}$, which corresponds to leaking all of v_h into variable l. The intersection in the definition of multi-run knowledge corresponds to traversing all possible low inputs in the attempt to match them to v_h, which is a worst-case model for multi-run attackers.

Now that we have definitions of attacker knowledge obtained after running the program, we wish to express a policy which sets limits on this knowledge. A *knowledge policy* states a lower bound on the attacker's uncertainty by partitioning the input domain into classes. Each class lists values that must remain indistinguishable to the attacker. In other words, the attacker may identify from which class the secret input comes, but any more precision is disallowed. This view corresponds to *partial release* [14, 33] of information. This leads to the following definition.

Definition 3 (Knowledge-policy). *A* knowledge policy *P for an input with domain D is a partition of D into* classes *P_i:*

$$P = \{P_1, \ldots, P_n\} \qquad P_i \subseteq D \qquad i \neq j \Longrightarrow P_i \cap P_j = \emptyset \qquad P_1 \cup \ldots \cup P_n = D$$

For a value $v \in D$ we write $[\![v]\!]_P$ to represent the class of P to which v belongs.

Note that as a partition of D, a policy represents an equivalence relation on values. Two values are equivalent if they come from the same class. This equivalence relation is often referred to as an *indistinguishability relation* [14, 33].

We illustrate the definition with simple examples. A policy that allows the attacker no knowledge is simply $P = \{D\}$. A policy that allows full knowledge is $\{\{x\} \mid x \in D\}$. If D is the set of unsigned 8-bit integers, then a policy that allows the attacker to know the parity of the secret is:

$$\{\{0, 2, \ldots, 254\}, \{1, 3, \ldots, 255\}\}.$$

We are now ready to state the formal definition of *multi-run secure programs*.

Definition 4 (Multi-run security). *Let c be a program that takes a secret input v_h and an arbitrary public, attacker-controlled input. c is* multi-run secure *(or simply secure) with respect to a knowledge policy P if and only if $[\![v_h]\!]_P \subseteq K_{v_h}(c)$.*

Observe that multi-run security with policy $\{D\}$ corresponds to *termination-sensitive noninterference* [39, 31] for single runs, which prevents the termination behavior of the program (as well as its public output) from being affected by secrets.

Recall programs 4 and 5 from Section 1. Program 4 ($l := h\%4$) is secure for all high input with respect to the policy $P = \{\{0, 4, \ldots\}, \{1, 5, \ldots\}\}, \{2, 6, \ldots\}, \{3, 7, \ldots\}$. Indeed, the multi-run knowledge from running the program has to be one of the four sets in the policy because the attacker only learns the two least-significant bits.

On the other hand, program 5 ($l := h\%6$) is insecure for all high input according to the policy P. To illustrate this, take $v_h = 0$. The multi-run knowledge $K_0(c)$ is $\{0, 6, \ldots\}$, while $[\![0]\!]_P = \{0, 4, \ldots\}$ which is clearly not contained in the knowledge.

As we are interested in an enforcement mechanism that allows limited leaks through the termination channel, Definition 4 will serve as the basis for a relaxed definition which we apply to the enforcement mechanism presented in Section 3. This relaxation draws on ideas from quantitative security. Smith [36] defines the notion of *vulnerability* $V(X)$, which is the worst-case probability of guessing the value of secret X by an adversary in one try. The measure of information quantity is then defined as $-\log V(X)$. Based on the intuition "information leaked = initial uncertainty - remaining uncertainty", Smith defines *information leakage* and shows that for deterministic programs and uniformly distributed secrets it amounts to $\log |S|$, where $|S|$ is the size of the set of possible public outputs given the public input is fixed. The intuition is that the more different observations the attacker can observe, the more secret information about might leaked through them. In the multi-run case, the size of the set of possible outputs translates to the number of indistinguishability classes for the high input, which, in effect, is the number of different values $K_{v_h}(c)$ can take when v_h varies. This is in line with Lowe [23], who measures the number of secret behaviors distinguished by an attacker in a nondeterministic setting. This motivation brings us to the following definition of security of programs that operate on uniformly distributed secrets:

Definition 5 (k-bit security). *Let c be a program that takes a uniformly distributed secret input v_h and an arbitrary public, attacker-controlled input v_l. c is k-*bit secure *if $k = \log n$ and $K_{v_h}(c)$ takes at most n distinct values as v_h varies.*

For example, program 1 is 1-bit secure because there are only two possibilities for $K_{v_h}(c)$ as v_h varies. On the other hand, program 2 is k-secure, where k is the bit size of h because $K_{v_h}(c)$ ranges over all possible singleton sets as v_h varies.

1-bit security is a particularly interesting case. Intuitively it means that an attacker can at most infer that v_h is in some set A or that it is in A's complement. In an extreme case, either set might contain only one element, meaning the attacker would know the exact value *of that particular* v_h, but since there are only two possible "knowledges" this is equivalent to the attacker being allowed only one boolean test on the secret.

Ultimately we will prove that our enforcement mechanism is multi-run secure with respect to a policy, with the relaxation that 1-bit leaks are allowed. For simplicity we combine Definition 4 and the 1-bit version of Definition 5 as follows.

Definition 6 (1-bit security w.r.t. a policy). *Assume c, v_h, v_l are as in Definition 5, and P is a knowledge policy. We say that c is* 1-bit secure with respect to P *if and only if for each class $P_i \in P$, $K_{v_h}(c)$ takes at most two distinct values (knowledges) K_1, K_2 as v_h varies within P_i, and furthermore $P_i \subseteq K_1 \cup K_2$.*

$$\begin{aligned}
&n \in D, \quad x \in \mathit{Vars}, \quad \mathit{op} \in \{+, -, \dots\} \\
&e ::= n \mid x \mid e\,\mathit{op}\,e \\
&c ::= \texttt{skip} \mid x := e \mid c; c \mid \texttt{if}\ e\ \texttt{then}\ c\ \texttt{else}\ c \mid \texttt{while}\ e\ \texttt{do}\ c
\end{aligned}$$

Fig. 1. Syntax

In other words, $K_{v_h}(c)$ can vary arbitrarily for v_h from different indistinguishability classes, but within each class we only allow for revealing one additional bit of information. The last part ensures that an attacker cannot otherwise exclude any values from the policy class of the secret, any value considered impossible in one knowledge must be considered possible according to the other knowledge.

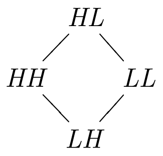
3 Enforcement

We illustrate our approach to enforcement for an imperative language. To keep the exposition clear, we have deliberately chosen a simple language, but the ideas here scale to more complex languages. Figure 1 shows the syntax of the language. Expressions take literals from a finite domain D (e.g., 32-bit integers) and variables from a set Vars. We present a type system for this language such that typable programs are robust against multi-run attacks that try to magnify single-run termination leaks into leaking more than one bit. The type system represents a static analysis, conveniently referring to security labels for variables and expressions as *security types*.

We will continue to treat programs as functions $D \times D \to D \cup \{\bot\}$, and in concrete examples the inputs will be represented by the variables h and l. The final value of l will be the output for terminating programs.

The important feature of this type system is that it does not allow looping on expressions that both depend on secrets *and* attacker input. Thus we need to consider both the confidentiality and integrity levels of expressions at the same time. To achieve this we label variables with labels from the following product lattice $\mathcal{L}$ that combines confidentiality and integrity.

Here a label lists first the confidentiality level and then the integrity level. For example, the attacker provided input l has level LL (low confidentiality, low integrity) since it is both known and controlled by the attacker, and the secret input h has level HH since it is neither. An expression combining a secret with untrusted input will be assigned level HL (high confidentiality, low integrity). We use the standard symbols $\sqsubseteq, \sqcup$, etc. for lattice operators. This lattice has been used for enforcing *robust declassification* [41, 25, 3], which demands that the attacker may not affect what is released by programs by ensuring that only high-integrity data can be declassified, and only in a high-integrity context. The work on robust declassification is a direct inspiration for our treatment of the termination channel in multi-run security. However, as we explain in Section 4, the policy that robust declassification enforces is rather different from our security model. Our observation that connects robust



$$\textsc{skip}\ \frac{}{pc \vdash \texttt{skip}} \qquad \textsc{assign}\ \frac{lev(e) \sqcup pc \sqsubseteq lev(x)}{pc \vdash x := e} \qquad \textsc{seq}\ \frac{pc \vdash c_1 \quad pc \vdash c_2}{pc \vdash c_1; c_2}$$

$$\textsc{if}\ \frac{pc \sqcup lev(e) \vdash c_1 \quad pc \sqcup lev(e) \vdash c_2}{pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2} \qquad \textsc{while}\ \frac{pc \sqcup lev(e) \vdash c \quad pc \sqcup lev(e) \neq \mathit{HL}}{pc \vdash \texttt{while}\ e\ \texttt{do}\ c}$$

Fig. 2. Typing rules

declassification with multi-run security enables us to cleanly reuse the enforcement technique, but still requires us to show soundness with respect to our security goals.

3.1 Enforcing 1-Bit Security

We start by showing that with a simple type system, we can make sure that typable programs cannot be used to magnify termination leaks beyond the traditional one-bit limit. The core idea is that the type system prevents information that is a mix of secrets and untrusted inputs from affecting termination behavior, by disallowing it in loop guards.

We equip the set of variables with a function giving the label of each variable, $label : Vars \to \mathcal{L}$. For expressions in general we define the function lev, assigning each expression with its security level. Function lev is defined as follows, pattern matching on the form of expression:

$$lev(n) = LH \qquad lev(x) = label(x) \qquad lev(e_1\, op\, e_2) = lev(e_1) \sqcup lev(e_2)$$

While variables have their corresponding label as a level, literals are always low confidentiality and high integrity, as we assume the program source to be public but trusted. Other expressions take the least upper bound of their component levels.

Figure 2 gives the typing relation. The typing context consists only of the level of the program counter, pc. This level represents expressions on which the control flow context depends, namely `if` and `while` guards. If a command c is typable under context pc, written $pc \vdash c$, the intention is that c does not leak when executed, even if the execution itself is conditioned on data of level pc or higher. Branches of an if-command must be typable under the outer pc level joined with the level of the guard expression (rule IF). The rule ASSIGN uses this to prevent implicit flows: assignments to a variable are only allowed when both the expression and the program counter are below or at the same level as the level of the assigned variable.

The rule WHILE propagates the level of the guard in the same way as IF, but in addition requires that the guard expression joined with the context pc is strictly below HL. The intention here is to prevent the attacker from selectively inducing nontermination that depends on the secret.

For example, program 1

$$\texttt{while}\ h\ \texttt{do skip}$$

is typable, because the level of the guard is HH. As we show below, this implies that it only leaks one bit and the attacker is not able to change termination behavior by varying the public input. Same goes for program

$$\texttt{while}\ l\ \texttt{do skip}$$

since although the attacker can control termination, it does not reveal anything about the secret. The level of the guard here is LL. However program 2

$$\texttt{while}\ h = l\ \texttt{do skip}$$

is not typable, as the level of the guard is HL. Indeed, recall that the attacker is able to try different inputs until one is found that corresponds to the secret, in which case the whole of h is revealed.

Our goal is to prove that the type system enforces that programs leak at most one bit (via a termination leak) even in the multi-run setting. To prove that typable programs leak at most one bit, we will start by excluding leaks other than termination leaks. This means that terminating programs satisfy noninterference, i.e., the observable output is independent of the secret input. First, we show that programs typable with a high-confidentiality pc cannot modify the low output.

Lemma 1. *Let c be a program. If $HL \vdash c$ or $HH \vdash c$, then for any choice of $v_h, v_l \in D$, if $c(v_h, v_l) \neq \bot$ then $c(v_h, v_l) = v_l$.*

The proofs of this lemma and other statements can be found in Appendix A.

We now establish noninterference for terminating runs.

Lemma 2. *Assuming a typable program c and ignoring diverging runs, c satisfies noninterference:*

$$\forall v_h, {v_h}', v_l \in D \ : \ \text{if } c(v_h, v_l) \neq \bot \neq c({v_h}', v_l) \quad \text{then} \quad c(v_h, v_l) = c({v_h}', v_l).$$

In particular, the above lemma tells us that (ignoring nonterminating runs), the single-run knowledge is unaffected by variation in the secret input. Thus, considering nontermination, the attacker can only observe one of two results, meaning the program only leaks one bit. The following lemma shows that this extends to the multi-run case, by showing that either termination depends only on the secret, or only on the public input. This means that the attacker can not improve their knowledge of v_h beyond the one bit already leaked, by varying the public input.

Lemma 3. *Assume c is a typable program. Then for arbitrary $v_h, {v_h}', v_l, {v_l}' \in D$ either one of the following condition holds.*

1. *Fixing the secret input, varying the public input reveals nothing:*

$$k_{v_h}(c, v_l) = k_{v_h}(c, {v_l}')$$

2. *Fixing the public input, varying the secret input reveals nothing:*

$$k_{v_h}(c, v_l) = k_{{v_h}'}(c, v_l)$$

The basic idea of the proof for this lemma, is that using Lemma 2 and assuming that neither condition holds, we can find a pair of high and low inputs that cause the program to diverge, while either of them can be combined with other inputs to cause the program to terminate successfully. By looking at the guard for the loop that causes divergence, its value must be governed by both high and low data, and so it cannot possibly have been allowed by the type system. Thus the assumption that neither condition holds must be false. As before, the full proof is presented in Appendix A.

We can now use the above results to prove that typable programs leak at most one bit.

Theorem 1. *Assume c is a typable program. Then c leaks at most one bit, i.e., for all $v_h \in D$ there are at most two distinct values for K_{v_h}.*

3.2 Enforcing General Knowledge Policies

We now draw on ideas of delimited release [32] to change our type system so that it enforces a general knowledge policy. Delimited release specifies a declassification policy as a set of expressions called *escape hatches*. Such expressions can refer to secret variables, but their computed values may be assigned to public variables. Thus, an escape hatch defines *what* secret information may be declassified as public. Note that the value of an escape hatch is not released automatically, but the program can use it to compute low confidentiality information that is then released explicitly as the public output.

The knowledge policy, a partition of D, is specified with an expression e_P. In terms of delimited release, this expression is an escape hatch, and to focus on the interesting ideas for this paper we assume it is the only one. Since the point of a policy expression is to partition the input space of h, any useful policy expression will only depend on h. Thus, we consider escape hatches that only involve high variables and generate policies from escape hatches as follows:

Definition 7. *An expression e, involving no other variables than h, generates a knowledge policy P as follows:*

$$P = \{P_1, \ldots, P_n\}$$

where for all v and v' we have $e(v) = e(v')$ if and only if $[\![v]\!]_P = [\![v']\!]_P$.

In order to support knowledge policies, we extend the type system with the possibility of *declassification*. The escape hatch expression is explicitly declassified to have the level LH, even though it may involve high confidentiality or low integrity variables. We adapt the definition of lev accordingly:

$$lev(e) = \begin{cases} LH & \text{if } e = e_P \text{ or } e = n \\ label(x) & \text{if } e \neq e_P \text{ and } e = x \\ lev(e_1) \sqcup lev(e_2) & \text{if } e \neq e_P \text{ and } e = e_1\, op\, e_2 \end{cases}$$

The only typing rule that needs to be changed from Figure 2 is the one for assignment, which disallows updates to any variable involved in the escape hatch:

$$\text{ASSIGN}\frac{lev(e) \sqcup pc \sqsubseteq lev(x) \qquad x \notin vars(e_P)}{pc \vdash x := e}$$

This is done in order to prevent information about the secret input being laundered through the escape hatch and is standard in delimited release [32]. See Program 7 for an example of laundering.

If the high input is a tuple of multiple high inputs, as described earlier, the ASSIGN rule should simply require that x is not one of them. We have left it as is in the interest of readability.

We return to the examples of Section 1 to illustrate the soundness and precision of the enforcement. Program 1 is still typable independently of escape hatches. Programs 2 and 3 are rightfully rejected in the absence of escape hatches because they might leak the entire secret. Given the escape hatch $h\%4$, the secure programs 4 and 6 are accepted by the type system because declassification relabels $h\%4$ to LH, which is under LL in the lattice, the label of l. Given the same escape hatch, the insecure program 5 is rejected because $h\%6$ of type HH is assigned to variable l of type LL. Program 7 is also rejected because variable h (which is involved in an escape hatch) is modified.

The soundness of the type system is guaranteed by the following theorem.

Theorem 2. *Assume c is a typable program and e_P is an escape hatch that induces a policy P. Then c is 1-bit secure with respect to the policy P.*

4 Related Work

Language-based information-flow security is a large and continuously-evolving field [31]. We focus on discussing most related work on knowledge-based security, interactive security, and declassification policies.

Knowledge-based security. Dima et al. [17] consider sets as representation of attacker's knowledge in nondeterministic systems. Askarov and Sabelfeld [4] present a knowledge-based condition of *gradual release* for declassification, as well as enforcement for a language with communication primitives. Gradual release allows the knowledge of the attacker to increase only when the program passes a declassification point.

Van der Meyden [38] expresses intransitive noninterference policies using a classical model of knowledge in terms of different agents' views of the world [18].

Banerjee et al. [7] enhance the knowledge-based representation of attackers with powerful program specification policies. As a result, they are able to express declassification policies of both *what* can be released and *where* in the code.

Askarov and Sabelfeld [5] use knowledge to describe both termination-insensitive and -sensitive security definitions with possibilities of expressing of *what* can be released and *where*, as well as dynamic enforcement for a language with dynamic code evaluation and communication primitives.

Broberg and Sands [11] describe *paralocks*, a knowledge-based framework for expressing versatile declassification policies, including role-based policies.

Demange and Sands [15] allow tuning sensitivity to (non)termination depending on the size of the secret that is involved in loop guards: looping is disallowed when loop guards depend on secrets of small size.

None of the above approaches model the attacker's knowledge obtained by running the program multiple times.

Interactive security. Multi-run security is related to interactive security. In particular, multi-run security of a batch-job program c that operates on secret variable h and public variable l can be related to single-run security of the following interactive program:

$$h' := h; \texttt{while } 1 \texttt{ do } (\texttt{in}(l); c; \texttt{out}(l); h := h')$$

where h' is an auxiliary variable. This encoding allows us for direct comparison with security definitions of interactive programs.

Le Guernic et al. [22] as well as Askarov and Sabelfeld [4] ignore diverging runs of interactive programs, which, as pointed out previously [7, 2], always allows program like c in the encoding above to be arbitrarily insecure.

ONeil et al. [27] investigate termination-sensitive security for programs that interact with input/output strategies, where strategies are represented as functions that compute the next input to the program based on the previous communication history. Being termination-sensitive and declassification-free, their condition rejects all of programs 1–7 from Section 1, if plugged to the encoding above.

Clark and Hunt [13] show that for deterministic programs, it makes no difference whether the user is represented by a strategy or an input/output stream. Askarov et al. [2] and Bohannon et al. [10] consider stream-based termination-insensitive security. However, as shown in [2], brute-force attacks similar to programs 2–3 are allowed.

Köpf and Basin [21] propose an information-theoretic model for multi-run security in the context of side-channel attacks. The timing side channel can be thought of as a generalization of the termination channel as nontermination manifests itself as long-lasting computation for a real-world attacker. Their model is based on refining the attacker's knowledge over multiple runs, well in line with our approach. However, as the motivation of Köpf and Basin's model is quantitative information leaks, they reason about finite numbers of runs and explore the space between our single-run and multi-run security definitions. Further, their enforcement is of rather different nature from ours: it is based on quantitative approximation using greedy heuristic.

Askarov and Sabelfeld [5] explore stream-based definitions for both termination-insensitive and -sensitive security in the presence of declassification policies. However, similar to the approaches above, the termination-sensitive condition rejects programs all of programs 1–7 and the termination-insensitive condition allows attacks 2–3, when plugged to the encoding above.

We have studied extensions of the multi-run secure type system presented here to interactive programs. Maintaining the 1-bit guarantee of termination-insensitive enforcement across all high inputs is non-trivial, as any public side effect (both input and output) will reveal information about the program counter to an attacker. If such an effect appears after a potentially diverging loop on high data, this will already leak one bit before the program has stopped. We envision that full integration of robust declassification and delimited release for interactive programs might be promising in this direction (see the discussion of dimensions of declassification below), but we expect problems with permissiveness of the enforcement. This indicates a fundamental trade-off between interactivity and security. Our paper identifies a niche, where it is possible to gain permissiveness without sacrificing security.

Declassification. As mentioned earlier, our declassification policy is an adaptation of *delimited release* [32]. Similarly to Askarov and Sabelfeld [5], we derive knowledge sets from escape-hatch expressions. The treatment of integrity by the type system is inspired by *robust declassification* [41, 25, 3]. Robust declassification guarantees that the attacker may not affect what is released by programs by ensuring that only high-integrity data can be declassified, and only in high-integrity context. In a similar spirit, our type system demands that loop context and guards may not mix high confidential data with attacker-controlled data.

In order to prevent unintended laundering of secrets, delimited release ensures that values of escape-hatch expressions do not change within a single run. In general, this guarantee does not extend over multiple runs, which potentially provides a laundering opportunity if the expression depends on data that is provided by an attacker, or is otherwise non-deterministic between runs. We avoid this issue at its root by not allowing non-secrets in escape hatches.

As we have foreshadowed earlier, we are able to cleanly reuse the robust declassification enforcement technique. However, note that we cannot automatically extract soundness guarantees from soundness results for robust declassification (e.g., [25]). The reason is that robust declassification addresses the *where* dimension of declassification: ignoring exactly *what* is leaked, but making sure the active attacker may not affect the declassification mechanism to leak more than the passive attacker. In contrast, our declassification policies are strict about *what* is leaked: the escape hatches describe the upper bound on leaks in programs.

Other, less related, work on declassification is described in a recent overview of the area [34]. The overview is organized by the dimensions of declassification.

5 Conclusions

We have showed how that extremes of insecurity (as with termination-insensitive noninterference) and over-restrictiveness of enforcement (as with termination-sensitive noninterference) can be avoided when generalizing batch-job security to multiple runs. Addressing the problem, we have presented a knowledge-based framework for specifying and enforcing multi-run security policies. The policy framework includes possibilities for declassification. The type-based enforcement tracks both confidentiality and integrity labels and guarantees multi-run security.

We expect interesting implications of our result for multi-threaded programs. The termination channel can be magnified in single-run multi-threaded programs in a fashion similar to using multiple runs of sequential programs. Assume we have as many threads as there are bits in secret h. Then, the multi-threaded program, where individual thread i is described as follows

$$T_i: \quad (\texttt{while}\ h\&\&b_i\ \texttt{do skip});\ \texttt{out}(i)$$

where b_i contains all zeros in the boolean representation except for bit i, leaks the entire secret in a single run. Our type-based enforcement can be straightforwardly applied to prevent this kind of leaks by considering the thread-dependent data b_i as low integrity. We expect that whenever a collection of threads is typable according to our type system, then the multi-threaded program that consists of the collection of threads is both single-run and multi-run secure (for a notion of *possibilistic* [24, 37, 33] security suitable for reasoning about nondeterministic programs).

As mentioned earlier, the termination channel can be seen as an instance of the timing side channel as nontermination manifests itself as long-lasting computation for a real-world attacker. We can offer protection against timing attacks that is similar to the protection against termination attacks: when the computation does not mix secret and attacker-controlled data in branch guards, then the timing leaks cannot be magnified. Otherwise, we resort to such existing approaches as cross-copying [1] and predictive black-box mitigation [6].

Note that there is nothing fundamental about our enforcement being static. We expect a dynamic mechanism, such as a monitor for delimited-release like policies by Askarov and Sabelfeld [5] to be easily adaptable to dynamically track both confidentiality and integrity in order to enforce our security condition.

Although the paper operates on a simple two-level security lattice, we do not anticipate difficulties with extending our approach to arbitrary lattices. Requiring the confidentiality level of a loop guard to be bounded by its integrity level gives us a way to prevent the dangerous mix of high-confidentiality and low-integrity data to affect the termination behavior. Other future work focuses on expressing multi-run security for richer languages. Further, we plan to extend the framework to take into account modifications of secret data between program runs. We are also exploring decentralized security policies by knowledge-based representations of multiple attackers.

Acknowledgments. Thanks are due to John Hughes and Dave Sands for helpful feedback. This work was funded by the European Community under the WebSand project and the Swedish research agencies SSF and VR. Arnar Birgisson is a recipient of the Google Europe Fellowship in Computer Security, and this research is supported in part by this Google Fellowship.

References

[1] Agat, J.: Transforming out timing leaks. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 40–53 (January 2000)

[2] Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 333–348. Springer, Heidelberg (2008)

[3] Askarov, A., Myers, A.: A semantic framework for declassification and endorsement. In: Gordon, A.D. (ed.) ESOP 2010. LNCS, vol. 6012, pp. 64–84. Springer, Heidelberg (2010)

[4] Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: Proc. IEEE Symp. on Security and Privacy, pp. 207–221 (May 2007)

[5] Askarov, A., Sabelfeld, A.: Tight enforcement of information-release policies for dynamic languages. In: Proc. IEEE Computer Security Foundations Symposium (July 2009)

[6] Askarov, A., Zhang, D., Myers, A.: Predictive black-box mitigation of timing channels. In: ACM Conference on Computer and Communications Security, pp. 297–307 (2010)

[7] Banerjee, A., Naumann, D., Rosenberg, S.: Expressive declassification policies and modular static enforcement. In: Proc. IEEE Symp. on Security and Privacy (May 2008)

[8] Banerjee, A., Naumann, D.A.: Stack-based access control and secure information flow. Journal of Functional Programming 15(2), 131–177 (2005)

[9] Barnes, J., Barnes, J.: High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)

[10] Bohannon, A., Pierce, B.C., Sjöberg, V., Weirich, S., Zdancewic, S.: Reactive noninterference. In: ACM Conference on Computer and Communications Security, pp. 79–90 (November 2009)

[11] Broberg, N., Sands, D.: Paralocks: role-based information flow control and beyond. In: Proc. ACM Symp. on Principles of Programming Languages (January 2010)

[12] Chapman, R., Hilton, A.: Enforcing security and safety models with an information flow analysis tool. ACM SIGAda Ada. Letters 24(4), 39–46 (2004)

[13] Clark, D., Hunt, S.: Non-interference for deterministic interactive programs. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 50–66. Springer, Heidelberg (2009)

[14] Cohen, E.S.: Information transmission in sequential programs. In: DeMillo, R.A., Dobkin, D.P., Jones, A.K., Lipton, R.J. (eds.) Foundations of Secure Computation, pp. 297–335. Academic Press, London (1978)

[15] Demange, D., Sands, D.: All secrets great and small. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 207–221. Springer, Heidelberg (2009)

[16] Denning, D.E.: A lattice model of secure information flow. Comm. of the ACM 19(5), 236–243 (1976)

[17] Dima, C., Enea, C., Gramatovici, R.: Nondeterministic nointerference and deducible information flow. Technical Report 2006-01, University of Paris 12, LACL (2006)

[18] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, Cambridge (1995)

[19] Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. IEEE Symp. on Security and Privacy, pp. 11–20 (April 1982)

[20] Hammer, C., Snelting, G.: Flow-sensitive, context-sensitive, and object-sensitive informationflow control based on program dependence graphs. International Journal of Information Security 8(6), 399–422 (2009); Supersedes ISSSE and ISoLA 2006

[21] Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In: ACM Conference on Computer and Communications Security, pp. 286–296 (2007)

[22] Le Guernic, G., Banerjee, A., Jensen, T., Schmidt, D.A.: Automata-based confidentiality monitoring. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 75–89. Springer, Heidelberg (2008)

[23] Lowe, G.: Quantifying information flow. In: Proc. IEEE Computer Security Foundations Workshop, pp. 18–31 (June 2002)

[24] McLean, J.: A general theory of composition for a class of "possibilistic" security properties. IEEE Transactions on Software Engineering 22(1), 53–67 (1996)

[25] Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing robust declassification and qualified robustness. J. Computer Security 14(2), 157–196 (2006)

[26] Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif: Java information flow. Software release. (July 2001), Located at http://www.cs.cornell.edu/jif

[27] O'Neill, K., Clarkson, M., Chong, S.: Information-flow security for interactive programs. In: Proc. IEEE Computer Security Foundations Workshop, pp. 190–201 (July 2006)

[28] Pottier, F., Simonet, V.: Information flow inference for ML. ACM TOPLAS 25(1), 117–158 (2003)

[29] Rizzo, J., Duong, T.: Padding oracles everywhere (2010), http://ekoparty.org/juliano-rizzo-2010.php

[30] Russo, A., Sabelfeld, A.: Securing interaction between threads and the scheduler. In: Proc. IEEE Computer Security Foundations Workshop, pp. 177–189 (July 2006)

[31] Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Selected Areas in Communications 21(1), 5–19 (2003)

[32] Sabelfeld, A., Myers, A.C.: A model for delimited information release. In: Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 174–191. Springer, Heidelberg (2004)

[33] Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher Order and Symbolic Computation 14(1), 59–91 (2001)

[34] Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. J. Computer Security 17(5), 517–548 (2009)

[35] Simonet, V.: The Flow Caml system. Software release. (July 2003), Located at http://cristal.inria.fr/~simonet/soft/flowcaml

[36] Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)

[37] Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 355–364 (January 1998)

[38] van der Meyden, R.: What, indeed, is intransitive noninterference? In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 235–250. Springer, Heidelberg (2007)

[39] Volpano, D., Smith, G.: Eliminating covert flows with minimum typings. In: Proc. IEEE Computer Security Foundations Workshop, pp. 156–168 (June 1997)

[40] Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. J. Computer Security 4(3), 167–187 (1996)

[41] Zdancewic, S., Myers, A.C.: Robust declassification. In: Proc. IEEE Computer Security Foundations Workshop, pp. 15–23 (June 2001)

[42] Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proc. IEEE Computer Security Foundations Workshop, pp. 29–43 (June 2003)

[43] Zheng, L., Chong, S., Myers, A.C., Zdancewic, S.: Using replication and partitioning to build secure distributed systems. In: Proc. IEEE Symp. on Security and Privacy, pp. 236–250 (May 2003)

Appendix A: Proofs

Proof of Lemma 1. By induction on typing derivation, c contains no assignments to l, since those are rejected by the constraint on pc in typing rule ASSIGN. Since nontermination is excluded, the output can only be the initial value of l, namely v_l. □

Proof of Lemma 2. We prove the lemma by induction on the structure of the program.

Case $c = \mathtt{skip}$ The statement holds because $c(v_h, v_l) = v_l$ for any v_h, v_l.

Case $c = x := e$ The case obviously holds when $x \neq l$. Otherwise, since c is typable under some pc, we know that $lev(e) \sqcup pc \sqsubseteq lev(l) = LL$. In particular $lev(e)$ is either LH or LL. From the definition of lev it is easy to show that e does not contain the high input, so the value of e must be the same whether the high input is v_h or $v_h{}'$.

Case $c = c_1; c_2$ Since c is typable under pc, then so are c_1 and c_2. By induction we have that $c_1(v_h, v_l) = c_1(v_h{}', v_l)$, let's call this intermediate low output $v_l{}''$. Then $c(v_h, v_l) = c_2(v_h, v_l{}'')$, which again by induction is equal to $c_2(v_h{}', v_l{}'') = c(v_h{}', v_l)$.

Case $c = \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2$ By induction the theorem holds for both branches c_1 and c_2. If $lev(e)$ is LL or LH, the value of e is unaffected by the secret input being v_h or $v_h{}'$, so in either case the same branch is selected and the theorem holds.

If $lev(e)$ is HL or HH then, since both branches are typable under $pc \sqcup lev(e)$, Lemma 1 gives that $c_i(v_h, v_l) = c_i(v_h{}', v_l) = v_l$ for both $i = 1$ and $i = 2$.

Case $c =$ `while` e `do` c Since we know that c terminates, we can write any program of this form as a sequence of sufficiently many `if` -statements with the same guard and body. This case can thus be reduced to the cases for conditionals and sequential composition. □

Proof of Lemma 3. Assume that condition 2 does not hold, i.e., $k_{v_h}(c, v_l) \neq k_{v_h{}'}(c, v_l)$ for some choice of v_h, v_l and $v_h{}'$. Lemma 2 gives that if both terminate, then $c(v_h, v_l)$ and $c(v_h{}', v_l)$ return the same value. Thus there are only two observable outcomes, that value or $\perp$. Given our assumption, we thus know that one of the runs diverges. Relabeling if necessary, we can assume that $c(v_h, v_l) = \perp$.

Now assume that there is a choice of $v_l{}'$ such that condition 1 also does not hold. If we can show this leads to a contradiction, either assumption is false and the theorem holds. Since $c(v_h, v_l) = \perp$, this must mean that $c(v_h, v_l{}')$ terminates. Based on our assumptions, the facts can be summarized as follows: $c(v_h, v_l)$ does not terminate; $c(v_h{}', v_l)$ terminates; and $c(v_h, v_l{}')$ terminates.

Consider the `while` loop that causes $c(v_h, v_l)$ to diverge, and in particular its guard expression e. We start by noting that if $lev(e) \sqsubseteq HH$ then its valuation can not be affected by the value of l. This is easy to prove (for expressions in general) from our type system, but laborious so for the sake of our discussion we leave out a precise proof. Symmetrically, if $lev(e) \sqsubseteq LL$ then its value can not depend on the value of h.

The typing rule for loops guarantees that the guard expression is not HL, so either we have $e \sqsubseteq HH$ or $e \sqsubseteq LL$. In the first case, $c(v_h, v_l{}')$ cannot terminate, since e will always evaluate to the same values (it is evaluated many times) as in $c(v_h, v_l)$. In the second case, $c(v_h{}', v_l)$ must similarly diverge.

Both cases contradict at least one of our assumptions, so one of them is false. □

Proof of Theorem 1. The proof is by contradiction. Assume that there are three values v_{h1}, v_{h2}, v_{h3} that would generate three distinct multirun knowledges. If we can show that at least two of $K_{v_{h\,i}}(c)$, for $i = 1, 2, 3$, must coincide then we have the contradiction we need. If $v_1, \ldots, v_n$ is an enumeration of D, then we can write the three multi-run knowledges as follows.

$$\begin{aligned}
K_{v_{h\,1}}(c) &= k_{v_{h\,1}}(c, v_1) \cap k_{v_{h\,1}}(c, v_2) \cap \cdots \cap k_{v_{h\,1}}(c, v_n)\\
K_{v_{h\,2}}(c) &= k_{v_{h\,2}}(c, v_1) \cap k_{v_{h\,2}}(c, v_2) \cap \cdots \cap k_{v_{h\,2}}(c, v_n)\\
K_{v_{h\,3}}(c) &= k_{v_{h\,3}}(c, v_1) \cap k_{v_{h\,3}}(c, v_2) \cap \cdots \cap k_{v_{h\,3}}(c, v_n)
\end{aligned}$$

Consider any "column" from this layout, say $k_{v_{h\,i}}(c, v_k)$ for some k and $i = 1, 2, 3$. If the (single-run) knowledges in this column are not all equal, we start by noting that there can be only two possibilities. Lemma 2 gives that if $c(v_{h\,i}, v_k)$ terminates, then the result is unique, so only that value and $\perp$ can be observed, and thus only two distinct knowledges are possible in the column. Let's call them A and B and say $k_{v_{h\,1}}(c, v_k) = k_{v_{h\,2}}(c, v_k) = A$ and $k_{v_{h\,3}}(c, v_k) = B$ (we can rearrange the indexes if necessary). Now Lemma 3 tells us that since the knowledges differ by altering the secret input, they cannot differ by altering the low input. This means that in this case all the single-run knowledges in the same "row" are the same and we obtain that $K_{v_{h1}}(c) = K_{v_{h2}}(c) = A$ and $K_{v_{h3}}(c) = B$. Note that it is enough for only one column to contain different values to force this equality in the rows.

On the other hand, if each column has three equal knowledges, then it is clear that $K_{v_{h1}}(c) = K_{v_{h2}}(c) = K_{v_{h3}}(c)$. In either case at least two of them must be equal. □

Proof of Theorem 2. To profe this theorem, we apply the same approach as for Theorem 1. This includes proving a modified version of Lemma 2 where instead of letting high inputs be arbitrary, we restrict them to one indistinguishability class of the policy. We'll note the use of these lemmas, and how their proofs differ from above, as we use them.

First we observe that during any derivation of the program c, the variable h remains constant and equal to v_h, since the modified type system disallows updates to it. This in turn means, that since e_P mentions no variables besides h, that it also remains constant. We can use this to show that ignoring diverging runs, c is non-interferent when secrets are taken from the same policy class. I.e.

$$\begin{aligned} &\forall v_h, {v_h}', v_l \in D \ : \\ &\qquad \text{if } c(v_h, v_l) \neq \bot \neq c({v_h}', v_l) \text{ and } [\![v_h]\!]_{e_P} = [\![{v_h}']\!]_{e_P} \\ &\qquad \text{then} \quad c(v_h, v_l) = c({v_h}', v_l). \end{aligned}$$

where $[\![v]\!]_{e_P}$ represents the value of e_P when the variable h is assigned the value v. The notation becomes clear when one considers the fact that this value represents the policy indistinguishability class of v. The proof of this statement is the same as the proof of Lemma 2, with a minor change: In the case for assignment, one must note that the typing judgement $LL \vdash x := e_P$ holds, regardless of $lev(x)$. This case is harmless, since the extra condition of $[\![v_h]\!]_{e_P} = [\![{v_h}']\!]_{e_P}$ provides that the same value will be assigned in both cases. A similar argument must, and can easily be made for Lemma 1.

After establishing this, the rest of the proof follows exactly the same argument as the proof of Theorem 1. We assume towards a contradiction, that there exists three secret inputs v_{h1}, v_{h2}, v_{h3}, all belonging to the same security class, which give rise to three distinct multi-run knowledges $K_{v_{h1}}, K_{v_{h2}}, K_{v_{h3}}$. We arrange their definitions as intersections of single-run knowledges as above, and note that either each single-run knowledge in a particular column is the same (i.e. the observed outcome does not depend on the secret input) or that all single-run knowledges in one line are equal (the observed outcome does not depend on the public input). In the latter case, all multi-run knowledges must be equal, and in the second one the non-interference above for terminating runs thus gives that there can be only two observable outputs. □

Appendix B: Multi-run Exploit

The following type-safe Jif [26] program contains a termination leak that can be amplified to leak more bits with multiple runs.

```
public class Leaky {

    public Leaky() {}
    public static void main{}(principal p, String[]{} args)
        throws (SecurityException)
    {
        int{Alice:} secret = 42;
        int{} input;

        try {
            input = Integer.parseInt(args[0]);
            while (0 != (secret & (1 << input))) {  }
        }
        catch (NullPointerException ignored) {}
        catch (ArrayIndexOutOfBoundsException ignored) {}
        catch (NumberFormatException ignored) {}
    }

}
```

A simple Python program can be used to make the 32 invocations of the above program needed to leak the complete secret.

```
import subprocess, time, sys, os

JIF_DIR = "../../../jif-3.3.1"

def run_with_timeout(command, timeout):
    proc = subprocess.Popen(command, bufsize=0,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE)
    start = time.time()
    while start + timeout > time.time() and proc.poll() is None:
        time.sleep(0.1)

    if proc.poll() is None:
        proc.terminate()
        return None

    out, err = proc.communicate()
    return out, err, proc.returncode

if __name__ == "__main__":
    secret = 0
    for bit in range(32):
        print "Checking bit %d... " % bit,
        r = run_with_timeout(
              ["jif",
               "-classpath", os.path.join(JIF_DIR, "tests"),
               "Leaky", str(bit)],
              1)
        if r is None:
            # This indicates timeout, assume non-termination
            secret = secret | (1 << bit)
            print "1"
        else:
            # This indicates normal termination
            print "0"

    print "Secret is: %d" % secret
```

Automated Information Flow Analysis of Virtualized Infrastructures

Sören Bleikertz[1], Thomas Groß[1], Matthias Schunter[1], and Konrad Eriksson[2]

[1] IBM Research - Zurich
{sbl,tgr,mts}@zurich.ibm.com
[2] InfraSight Labs
konrad.eriksson@infrasightlabs.com

Abstract. The use of server virtualization has been growing steadily, but many enterprises still are reluctant to migrate critical workloads to such infrastructures. One key inhibitor is the complexity of correctly configuring virtualized infrastructures, and in particular, of isolating workloads or subscribers across all potentially shared physical and virtual resources. Imagine analyzing systems with half a dozen virtualization platforms, thousands of virtual machines and hundreds of thousands of inter-resource connections by hand: large topologies demand tool support.

We study the automated information flow analysis of heterogeneous virtualized infrastructures. We propose an analysis system that performs a static information flow analysis based on graph traversal. The system discovers the actual configurations of diverse virtualization environments and unifies them in a graph representation. It computes the transitive closure of information flow and isolation rules over the graph and diagnoses isolation breaches from that. The system effectively reduces the analysis complexity for humans from checking the entire infrastructure to checking a few well-designed trust rules on components' information flow.

1 Introduction

Large-scale virtualized infrastructures and cloud deployments are a common and still growing phenomenon. The goals of server virtualization include high utilization of today's hardware, fast deployment of new virtual machines and load balancing through migration of existing virtual machines. Virtualized infrastructures provide standardized computing, virtual networking, and virtual storage resources. Correspondingly, infrastructure clouds provide simple machine creation and migration mechanisms as well as seemingly unlimited scalability, while the costs incurred are only proportional to the resources actually used.

The growth of IT infrastructures and the ease of machine creation have lead to substantial numbers of servers being created (server sprawl). Furthermore, then led to large and complex configurations that arise by rank growth and evolution rather than by advance planning and design. Indeed, the configuration

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 392–415, 2011.

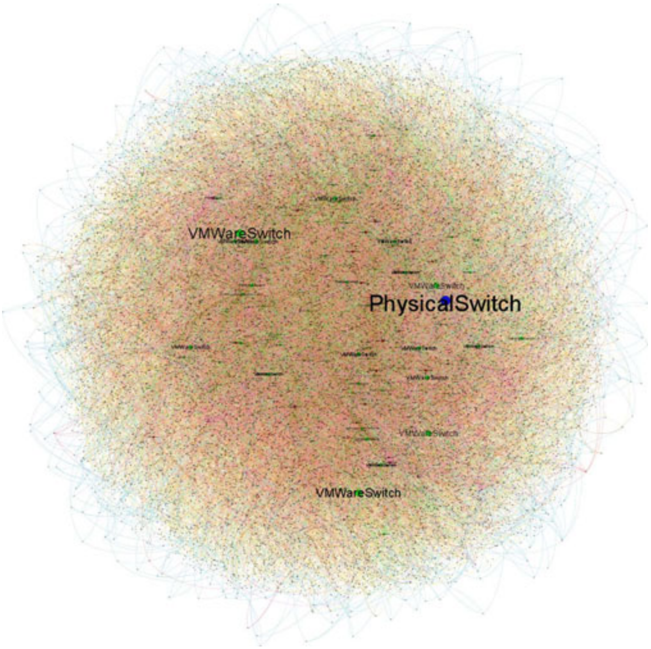


Fig. 1. Illustration of the overwhelming complexity of a mid-size infrastructure with 1,300 VMs

complexity often exceeds the analysis and management capabilities of human administrators. We depict an example for a mid-size infrastructure in Figure 1. This, by itself, calls for automated security analysis of virtualized infrastructures. The high complexity of an analysis is amplified when considering security properties such as isolation, because then the analysis of individual resources must be complemented with an analysis of their composition.

In addition, virtualization providers often aim at establishing multi-tenancy, that is, the capability to host workloads from different subscribers on the same infrastructure. Also, they provide an open environment, in which arbitrary subscribers can register without trust between subscribers being justified. Therefore, we need to assume that workloads as well as VMs are under the control of an adversary, and that an adversary will use overt and covert channels in its reach.

Industry partially approaches isolation with automated management and deployment systems constraining the users' actions. However, these mechanisms can fail, lack enforcement, or be circumvented by human intervention.

1.1 Contributions

The goal of this paper is to study automated information flow analysis for large-scale heterogeneous virtualized infrastructures. We aim at reducing the analysis complexity for human administrators to the specification of a few well-designed trust assumptions and leave the extrapolation of these assumptions and analysis of information flow behavior to the tools.

We propose an information flow analysis tool for virtualized infrastructures. The tool is capable of discovering and unifying the actual configurations of different virtualization systems (Xen, VMware, KVM, and IBM's PowerVM[1]) and running a static information flow analysis based on explicitly specified trust rules. Our analysis tool models virtualized infrastructures faithfully, independent of their vendor, and is efficient in terms of absence of false negatives as well as adjustable false positive rates.

Our approach transforms the discovered configuration input into a graph representing all resources, such as virtual machines, hypervisors, physical machines, storage and network resources. The analysis machinery takes a set of graph traversal rules as additional input, which models the information flow and trust assumptions on resource types and auxiliary predicates. It checks for information flow by computing a transitive closure on an information flow graph coloring with the traversal rules as policy. From that, the tool diagnoses isolation breaches and provides refinement for a root causes analysis. The challenge of information flow analysis for virtualized infrastructures lays in the faithful and complete unified modeling of actual configurations, a layered analysis that maintains completeness and correctness through all stages, and a suitable refinement to infer the root causes for isolation breaches.

Our method applies strict over-abstraction to minimize false negatives. This means that we only assume absence of flows for components that are known to isolate. This enables us to reduce the analysis correctness to the correctness of the traversal rules. As this method accepts an increase in the false positive rate, we allow administrators to fine-tune the trust assumptions with additional traversal rules and constraint predicates to obtain a suitable overall detection rate.

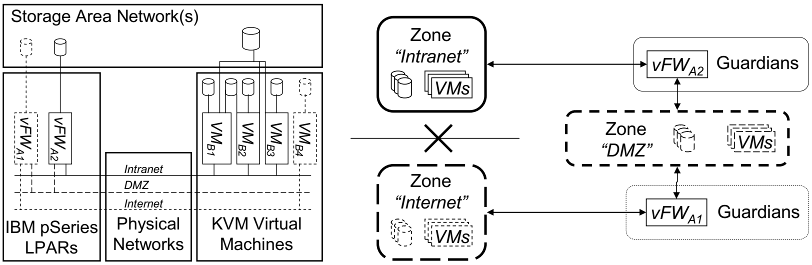
We report on a case study for a mid-sized infrastructure of a financial institution production environment in §7.

1.2 Applications

Our technique is applicable to the isolation analysis of complex configurations of large virtualized datacenters. Such datacenters include different types of server hardware, implementations of virtual machine monitors, as well as physical and virtual networking and storage resources.

Let us consider a simplified version of such a configuration in Figure 2(a). This simplified version includes the following hardware: A IBM pSeries[1] server, an x86 server, a virtual networking infrastructure providing VLANs, and a Storage Area Networking providing virtual storage volumes. The virtual resources (networks, storage, machines, and virtual firewalls) are depicted inside these hardware resources. Keep in mind that sizable real-world configurations contain thousands of virtual machines and hundreds of thousands of connections.

Figure 2(b) depicts a desired isolation topology for this example: we have three example virtual security zones "Intranet", "DMZ", and "Internet". Furthermore,



(a) Architecture Overview (b) Isolation Zones

Fig. 2. An example setup of a virtualized datacenter with an isolation policy for three virtual security zones

we permit communication between Intranet and DMZ that is mediated by a trusted guardian, such as a virtual firewall vFW_{A2}. Similarly, firewall vFW_{A1} moderates and restricts the communication between the Intranet and Internet zones, respectively. The isolation analysis must check that there do not exist components that connect two zones or are shared by two zones, while not being trusted to sufficiently mediate direct and covert information flows.

Note that we focus on validating the virtualized infrastructure's configuration. Once we have guaranteed that no undesired information flow exists except through the specified guardians, we would need to employ techniques from firewall filtering analysis, e.g. [17,18,28], to ensure that the guardians have been configured correctly.

2 Related Work

Virtual systems introduce several new security challenges [7]. Two important drivers that inspired our work are the increase of scale and the transient nature of configurations that render continuous validation more important.

The first area of related work is security of virtual machine monitors. This knowledge is needed to underpin the user's individual decisions whether to trust a given component. Analysis of well-known attacks such as jailbreaks [27] allows one to detect vulnerable configurations. This includes information leakage vulnerabilities of today's infrastructure clouds that allow covert or overt communication between multiple tenants that should be isolated. Examples include co-hosting validation [21] and cache-based side channels [1,20].

To our knowledge, there do not exist any research contributions on the static high-level information flow analysis in virtualized infrastructures. Still, we draw inspiration from information flow analysis such as research in separation [12,23], channel control [22], and non-interference [8,9,16,24]. We discuss these influences on our own definition on structural information control in §3. More often than not, we find research in this space focused on the information flow between high and low variables and not on the information flow in larger topologies.

A second area of related work is *reachability analysis* in networks and the related configuration analysis of firewalls. Our isolation analysis draws from known work on reachability analysis. Analyzing firewalls and complex network infrastructures allows one to decide to what extent two known networks are connected. In particular, Al-Shaer *et al.* [2,13,29] analyze entire network infrastructures including packet filters, transformers, and routers. In [5], it was shown how reachability analysis can be applied to infrastructure clouds. *Firewall configuration analysis* allows the understanding and validation of firewall rules [17,18,28]. While this work focuses on the TCP/IP level, our goal is to ensure 'physical' isolation by ensuring that VLANs and virtual networks are disjoint. This approach is similar to the approach proposed in [14]. If L2 networks are connected while isolation is implemented on the TCP/IP level, we see potential to further extend our work by using these concepts for TCP/IP isolation analysis that is then fed into our analysis concept.

This area of research is also important for modeling the behavior of imperfect guardians. Whereas this paper assumes that guardians always make correct decisions and stop dangerous information flow, reachability and firewall configuration analyses allow one to model imperfections as explicit traversal rules. Guardians with packet inspection and stateful analysis may even discover illegal information flow hidden in legal flows.

Whereas earlier security analyses considered stand-alone elements of a virtualized infrastructure, a tool-supported information flow analysis of a full virtualized environment is still missing, not to speak of complex heterogeneous and large-scale virtualized infrastructures with a diversity of underlying platforms. The research areas of information flow and reachability analysis underpin our efforts, yet so far have not produced a mechanized approach for this problem statement.

3 A Model for Isolation Analysis

In our work, we consider *overt* and *covert* channels. *Covert channels* [15] are not intended for information transfer at all, yet seem to be a common phenomenon in virtualized infrastructures. Requiring the absence of all covert channels from hypervisors, physical hosts and resources will render many resulting system impractical. Therefore, we allow administrators to specify a certain amount of covert channel information flow as tolerable.

In the quest for a suitable requirements definition, we review information flow types [25,15,21,8,9,24,16,10,23,12,11] in Appendix A. At this point, we note i. that we need intransitivity and ii. that *channel control* [22] captures our requirement to specify *exceptions* to the general zoning requirements. Thus, we introduce a property we call *structural information control* that essentially lifts channel control to topology:

Definition 1 (Structural Information Control). *A static system topology provides* structural information control *with respect to a set of information flow* *assumptions on system nodes if there does not exist an inter-zone information flow unless mediated by a dedicated guardian.*

Observe that we aim at the detection of isolation breaches (information flow traces), which renders our approach loosely similar to model checking, and not at the verification of absence of information flow, which would be similar to theorem proving.

3.1 Modeling Isolation

Modeling Configurations. Our static information flow analysis is graph-based. Each element of a virtualization configuration is represented by (at least) one vertex (VMs, VM hosts, virtual storage, virtual network). Connections between elements are represented by edges in the graph and model *potential* information flow. Note that our approach requires completeness of the edges: While not all edges may later actually constitute information flows, we require that all relations that allow information flow are actually modeled as an edge.

The vertices of the graph are typed: our model distinguishes VM nodes, VM host nodes, storage and network nodes, etc.

Definition 2 (Graph Model). *Let $\mathbb{T} \subset \Sigma^+$ a set of vertex types and $\mathbb{P} \subset \Sigma^+$ a set of vertice properties. A virtualization graph model contains a set of typed vertices $V \subset \mathbb{V} := (\Sigma^+ \times \mathbb{T} \times \mathbb{P})$ and a set of edges $E \subseteq (V \times V)$. A vertice v is a triple of label, type and properties set $(l, t, p) \in \mathbb{V}$. An edge e is a pair of start and end vertice $(v_i, v_j) \in (V \times V)$. A set of edges E is called* valid *with respect to a set of vertices V', if $E \subseteq (V' \times V')$. A graph $(\bar{V}, \bar{E})$ is called a* valid subgraph *of graph (V, E), if $\bar{V} \subseteq V$ and $\bar{E} \subseteq E$ is valid with respect to $\bar{V}$. An edge set $\boldsymbol{E} \subseteq E$ is called a* path *if the edges and their respective vertices form a connected valid sub-graph of (V, E).*

We represent complex structures of the virtualization infrastructure by subgraphs of multiple vertices. For instance, we construct guardians such as firewalls with complex information flow rules by a firewall vertex connected to multiple port vertices.

Information is output at one or more *information source* nodes, propagates according to *traversal rules* along the nodes and edges of the graph, and is consumed at an *information sink*. We treat information sources as independent and information as untyped and unqualified.

Definition 3 (Information Sources and Sinks). *For a set of vertices V, we define a set of information sources $\hat{V} \subseteq V$ and a set of information sinks $\check{V} \subseteq V$. A vertice $\hat{v} \in \hat{V}$ is called* information source*, a vertice $\check{v} \in \check{V}$* information sink*.*

Modeling Information Flow Assumptions. A *traversal rule* models an assumption on information flow from one vertex type to another vertex type. For instance, a traversal rule will specify that if a VM host is connected to a storage provider, this edge constitutes a direct information flow and is to be traversed. Also, a traversal rule may specify that if two VMs are connected to the same

VM host, this implies the risk of covert channel communication and, therefore, constitutes an information flow.

Definition 4 (Traversal Rules). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, the* traversal rules *are a propositional function of source type, destination type, source properties, and destination properties over a type relation R and a predicate P:*

$$f_{\mathbb{T},\mathbb{P}} : (\mathbb{T} \times \mathbb{T} \times \mathbb{P} \times \mathbb{P}) \rightarrow \{\mathsf{stop}, \mathsf{follow}\} :$$

$$f_{\mathbb{T},\mathbb{P}}(t_i, t_j, p_i, p_j) := \begin{cases} (t_i, t_j) \in R \wedge P(p_i, p_j) & \mathsf{follow} \\ (t_i, t_j) \notin R \vee \neg P(p_i, p_j) & \mathsf{stop} \end{cases}$$

We call traversal rules simple*, if P is always true.*

Definition 5 (Completeness). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, traversal rules $f_{\mathbb{T},\mathbb{P}}$ are called* complete *if R and P associated to $f_{\mathbb{T},\mathbb{P}}$ are complete. We call a default rule a* completion *of incomplete traversal rules $f_{\mathbb{T},\mathbb{P}}$, if it maps all undetermined cases to either* stop *or* follow*. We call non-default rules* explicit*.*

Whereas completeness is a property of a set of traversal rules, we define *coverage* as in how far a set of traversal rules determines the analysis of a graph deterministically without invoking the default rule.

Definition 6 (Coverage). *For the set of vertice types $\mathbb{T} \subset \Sigma^+$ and a set of vertice properties $\mathbb{P} \subset \Sigma^+$, consider a virtualization graph (V, E) as in Def. 2 and the subset of edges $E' \subseteq E$ that are matched by explicit traversal rules $f_{\mathbb{T},\mathbb{P}}$. We call the quotient of number of explicitly matched edges to total number of edges* coverage*: $c = |E'| \,/\, |E|$*

Observe that a complete coverage, that is, $c = 100\%$ is significant for achieving a low false-positive rate.

The traversal rules specify general assumptions on information flow in virtualized environments and, thereby, embodies a part of the overall trust assumptions. The specification of traversal rules is therefore orthogonal to the isolation policy of a system. Whereas our system comes with a root set of traversal rules as base line trust assumptions, we allow users to specify multiple sets of *user-defined traversal rules* and thereby *user-defined trust assumptions.*

Similar to the tainted variable method for static information flow analysis, we employ the metaphor of color propagation. We associate colors to information sources $\hat{v} \in \hat{V}$ and to vertices that have received information flow from a certain source by the evaluation of traversal rules $f_{\mathbb{T},\mathbb{P}}$. The total information flow of a system is the *transitive closure* of the graph traversal governed by the traversal rules $f_{\mathbb{T},\mathbb{P}}$. This means, that the information flow from any source to any sink can be efficiently statically analyzed by a reachability analysis between source and sink.

We define graph coloring recursively.

Definition 7 (Graph Coloring). *Let traversal rules $f_{\mathbb{T},\mathbb{P}}$, a graph (V, E) and an information source $\hat{v} \in \hat{V} \subseteq V$ with color c be given. Then, $\hat{v}$ is* colored *with c by definition. A vertice $v \in V$ is* colored *with c, if there exists an edge $e = (\cdot, v) \in E$, which is* colored *with c. An edge $e = (v_s, v_d) \in E$ with $v_s = (\cdot, t_s, p_s)$ and $v_d = (\cdot, t_d, p_d)$ is* colored *with c iff (i) v_s is* colored *with c and (ii) $f_{\mathbb{T},\mathbb{P}}(t_s, t_d, p_s, p_d) = \mathsf{follow}$.*

4 Isolation Analysis of Virtual Infrastructures

We apply the foundations from the preceding section to virtualized infrastructures. Our approach (see Figure 3) consists of four steps organized into two phases: 1) building a graph model from platform-specific configuration information and 2) analyzing the resulting model. The graph model is formally defined in Def. 2.

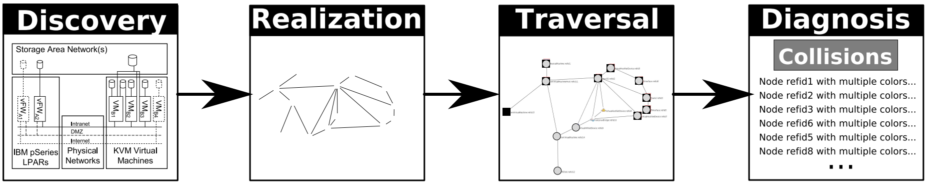

Discovery
Storage Area Network(s)
vFW$_{A1}$
vFW$_{A2}$
Intranet
DMZ
Internet
VM$_{B1}$
VM$_{B2}$
VM$_{B3}$
VM$_{B4}$
IBM pSeries LPARs
Physical Networks
KVM Virtual Machines
Realization
Traversal
Diagnosis
Collisions
Node refid1 with multiple colors...
Node refid2 with multiple colors...
Node refid3 with multiple colors...
Node refid6 with multiple colors...
Node refid5 with multiple colors...
Node refid8 with multiple colors...
...

Fig. 3. Overview over the analysis flow

The first phase of building a graph model is realized using a discovery step that extracts configuration information from heterogeneous virtualized systems, and a translation step that unifies the configuration aspects in one graph model. For the subsequent analysis, we apply the graph coloring algorithm defined in Def. 7 parametrized by a set of traversal rules and a zone definition. The assessment of the resulted colored graph model enables a diagnosis of the virtualized infrastructure with respect to isolation breaches.

4.1 Discovery

The goal of the discovery phase is to retrieve sufficient information about the configuration of the target virtualized infrastructure. To this end, platform-specific data is obtained through APIs such as VMware VI, XenAPI, or libVirt, and then aggregated in one discovery XML file. The target virtualized infrastructure, for which we will discover its configuration, is specified either as a set of individual physical machines and their IP addresses, or as one management host that is responsible for the infrastructure (in the case of VMware's vCenter or IBM pSeries's HMC). Additionally, associated API or login credentials need to be specified.

For each physical or management host given in the infrastructure specification, we will employ a set of discover probes that are able to gather different aspects of the configuration. We realized multiple hypervisor-specific probes for Xen, VMware, IBM's PowerVM, and LibVirt. Furthermore, if the management VM is running Linux[1], we also employ probes for obtaining Linux-specific configuration information. Currently, we do not discover the configuration of the physical network infrastructure. However, the framework easily be extended beyond the existing probes or use configuration data from a third-party source.

4.2 Transformation into a Graph Model

We translate the discovered platform-specific configuration into a unified graph representation of the virtualization infrastructure, the *realization model*. The realization model is an instance of the graph model defined in Def. 2. It expresses the low-level configuration of the various virtualization systems and includes the physical machine, virtual machine, storage, and network details as vertices. We generate the realization model by a translation of the platform-specific discovery data. This is done by so-called *mapping rules* that obtain platform-specific configuration data and output elements of our cross-platform realization model. Our tool then stitches these fragments from different probes into a unified model that embodies the fabric of the entire virtualization infrastructure and configuration.

For all realization model types in $\mathbb{T}$ (cf. Def. 8), we have a mapping rule that maps hypervisor-specific configuration entries to the unified type and, therefore, establishes a node in the realization model graph. We obtain a complete iteration of elements of these types as graph nodes. The mapping rules also establish the edges in the realization model.

This approach obtains a complete graph with respect to realization model types. Observe that configuration entries that are not related to realization model types are not represented in the graph. This may introduce false negatives if there exist unknown devices that yield further information flow edges. To test this, we can introduce a default mapping rule to include all unrecognized configuration entries as dummy nodes.

4.3 Coloring through Graph Traversal

The graph traversal phase obtains a realization model and a set of information source vertices with their designated colors as input. According to Def. 7, the graph coloring outputs a colored realization model, where a color is added to a node if permitted by an appropriate traversal rule. We use the following three types of traversal rules (see Def. 4 and the definition of traversal rules below) that are stored in a ordered list. We apply a first-matching algorithm to select the appropriate traversal rule for a given pair of vertices.

Flow rules model the knowledge that information can flow from one type of node to another if an edge exists. For example, a VM can send information onto a connected network. These rules model the "follow" of Def. 4. *Isolation rules* model the knowledge that certain edges between trusted nodes do not allow information flow. For example, a trusted firewall is known to isolate, i.e., information does not flow from the network into the firewall. These rules model the "stop" of Def. 4. *Default rule* means that ideally, either isolation or else flow rules should exist for all pairs of types and all conditions, that is, we want to achieve complete coverage according to Def. 6: For any edge and any two types, the *explicit traversal rules* should determine whether this combination allows or disallows flow. In practice, the administrator may lack knowledge for certain types. As a consequence, we included a default rule as *completion*. Here, we establish a *default flow rule*: whenever two types are not covered by an isolation or flow rule, then we default to "follow". To be on the safe side, i.e., reducing false negatives, we assume that flow is possible along this unknown type of edges.

Given this set of rules, we then traverse the realization model by applying the set of traversal rules and color the graph according to information flows from a given source. The traversal starts from the information sources and computes the transitive closure over the traversal rule application to the graph.

4.4 The Traversal Rules

The graph coloring algorithm requires a set of traversal rules that model information flows, isolation properties, and trust assumptions. We will propose a set of rules and explain their purposes, and leave the correctness argument to the security analysis in §5.2 and a detailed discussion in Appendix B.2.

Definition 8 (Traversal Rule). *Let F be a set of follow types $\{\mathsf{stop}, \mathsf{follow}\}$, $\mathbb{T}' \subset \mathbb{T}$ be a set of realization model types {Port, NetworkSwitch, PhysicalSwitch, ManagementOS, PhysicalDevice, VirtualMachine, VirtualMachineHost, StorageController, PhysicalDisk, FileSystem, File, any}, and D be a set of flow directions $\{\Rightarrow, \Leftarrow, \Leftrightarrow\}$, where $\Rightarrow$ and $\Leftarrow$ denote a unidirectional, and $\Leftrightarrow$ a bi-directional flow. A traversal rule is a tuple (f, t_0, t_1, d, P, g) with $f \in F$, $t_0, t_1 \in \mathbb{T}'$, $d \in D$, P is a predicate over properties and colors of the realization model, and g is a color modification function. During graph coloring (see Def. 7), g can transform the color c of a colored vertex $\hat{v}$ while coloring a new vertex v, i.e., $c(v) = g(c(\hat{v}))$.*

The traversal rules specified in Table 1 are a ordered list of rules (as defined in Def. 8). In case the condition is left empty, a *true* predicate is assumed, and in case the color modification is empty, g is the identity function.

Definition 9 (Matching Rule). *Given a traversal rule (f, t_0, t_1, d, P, g) as defined in Def. 8 and a source and destination vertex from the graph traversal: v_s and v_d respectively. The rule matches iff i) $(t_0 = t(v_s) \vee t_0 = any) \wedge (t_1 = t(v_d) \vee t_1 = any)$ where $t(v)$ denotes the type of a given vertex v, ii) $d \in \{\Rightarrow, \Leftrightarrow\}$, iii) $P = true$.*

The first-matching algorithm iterates over the ordered list of traversal rules (Table 1) and applies the matching rule defined in Def. 9. If the matching evaluates to true, the iteration stops and the matched rule is returned. The matching of the traversal rules induces a function representation of the traversal rules as defined in Def. 4.

Our trust assumptions are specified in the rules namely, Rule 1, Rule 2, Rule 3, and Rule 4. These model that VLANs are isolated on physical switches, that the

Table 1. Traversal Rules

#	Type	Flow	Condition +Color Modification
		Trust Rules	
1	stop	$PhysicalSwitch \Rightarrow Port$	Has any *vlan* color
2	stop	$ManagementOS \Leftrightarrow any$	
3	stop	$PhysicalMachine \Leftrightarrow PhysicalDevice$	
4	stop	$VirtualMachine \Leftrightarrow VirtualMachineHost$	
		Network Switches	
5	stop	$Port \Leftrightarrow NetworkSwitch$	Port is disabled
		VLAN	
6	follow	$Port \Rightarrow NetworkSwitch$	Port has VLAN tagging with tag \$VLAN + Create *vlan-\$VLAN*
7	follow	$NetworkSwitch \Rightarrow Port$	Port's VLAN tag matches color's one +Remove *vlan-\$VLAN*
8	follow	$NetworkSwitch \Rightarrow Port$	Port is trunked
9	stop	$NetworkSwitch \Rightarrow Port$	Port's VLAN tag mismatches color's one
10	stop	$NetworkSwitch \Rightarrow Port$	Has any *vlan* color
		Storage	
11	stop	$StorageController \Rightarrow PhysicalDisk$	
12	stop	$FileSystem \Rightarrow File$	
		Default	
13	follow	$any \Leftrightarrow any$	

privilege VM and the physical machine are trusted and do not leak information, and that we exclude cross-VM covert channels (see §5.2).

Rule 5 simply stops an information flow if a network port is disabled. Rule 6 and Rule 7 model the VLAN en- and decapsulation of network traffic. Traffic with a VLAN tag is modeled as a new color *vlan* with the VLAN tag appended, which is created in case of encapsulation and removed in case of decapsulation. In the case of VMware, the VLAN tag for a VM is modeled as a non-zero *defaultVLAN* property of the port. Rule 8 specifies that if a port is marked as trunked, which is required in the case of VMware to allow traffic from the VMs to the physical network interface, the VLAN traffic is also allowed to flow. Otherwise, if the *vlan* color tag mismatches the port's VLAN tag, we isolate and stop the information flow (see Rule 9). This also applies to Rule 10, which is the default isolation rule for VLAN traffic, if one of the previous rules did not match.

On the storage side, we model the behavior of the storage controller not to leak information from one disk to another with Rule 11. Furthermore, the filesystem will not leak information from one file to another (Rule 12).

The default rule Rule 13 allows any information flow that was not handled by a previous rule due to the first-matching algorithm.

We make three observations about the traversal rules: *First*, administrators can modify existing and specify further traversal rules, for instance, to relax trust assumptions or to model known behavior of specific components. *Second*, traversal rules serve as generic interface to include analysis results of other information flow tools into the topology analysis (e.g., firewall information flow analysis). *Third*, the behavior of explicit guardians (see Def. 1) is introduced by traversal rules specific to these nodes. For instance, the guardians in the exemplary Figure 2, §1.2, would receive a stop-rule.

4.5 Detecting Undesired Information Flows

The goal of the detection phase is to produce meaningful outputs for system administrators. For detecting undesired information flows, we color a set of information sources that mark types of critical information that must not leak. The idea of the color spill method is to introduce nodes called 'sinks' (see Def. 3). Each sink is colored with a subset of the colors corresponding to the information that it is allowed to receive. In practice, the administrator provides a list of clusters or zones that shall be isolated, and we add/mark sources and sinks according to the isolation policy with respect to these zones. In our example from Figure 2, §1.2, we would mark nodes from the zones "Intranet" ($VM_{B1}, VM_{B2}, VM_{B3}$) and "Internet" ($VM_{B4}$) as sources and the guardians of the opposite zones (vFW_{A1}, vFW_{A2}) as sinks, to determine isolation breaches in both directions. After the transitive closure of the traversal rules, we check whether any additional colors "spilled" into a given sink. If a sink gets connected to an additional color, then we have found a potential isolation breach. You could imagine the dedicated color sinks as a honey pot, waiting for colors from other zones to spill over.

Observe that the detection of a color spill only indicates the existence of a breach and between which zones (source-sink pairs) it has occurred. The color flow can be visualized and of some use for administrators to fix the problem. In addition, we study different refinement methods for root-cause analysis, in order to pinpoint critical edges responsible for the information flow in a industry case study (§7).

5 Security Analysis of the Automated Information Flow Analysis

Definition 10 (System Assumptions). *We assume correctness of discovery/translation and isolation behavior as defined in §4.4.*

- Completeness of Discovery*: We assume that the configuration output of virtualized infrastructures contains all elements that might solicit information flow (cf. §4.1)*
- Correct Translation Modules*: We assume that the discovery modules analyzing concrete systems are capable to correctly identify configuration elements that translate to vertices and edges in the realization model (cf. §4.2).*
- Hypervisor Separation*: We assume that the hypervisor sufficiently prevents cross-VM information flow through covert channel down to a tolerable level (cf. Rule 4, §4.4).*[1]

[1] This assumption is modeled by the hypervisor traversal rules and can be explicitly specified by administrators.

- VLAN Separation by Physical Switches*: We assume that physical switches isolate different VLANs from each other (cf. Rule 1, §4.4).*

5.1 Reduction to Correctness of the Traversal Rules

The graph coloring establishes the following events through a transitive closure of traversal rules $f_{\mathbb{T},\mathbb{P}}$ over a graph (V, E) with sources $\hat{V}$ and sinks $\check{V}$.

Definition 11 (Events). Wlog.*, we model admissible colors of an information sink $\check{v} \in \check{V}$ as colors associated with $\check{v}$. Then we have:*

- *We call an event* B *an* isolation breach *if a information sink $\check{v} \in \check{V}$ is colored with an inadmissible color of a information source $\hat{v} \in \hat{V}$ such that $\hat{v} \neq \check{v}$.*
- *We call an event* A *an* alarm *if an isolation audit reports an information flow between a distinct information source $\hat{v} \in \hat{V}$ and information sink $\check{v} \in \check{V}$.*
- *We call the set of events* $\neg\mathsf{A} \wedge \mathsf{B}$ *a* false negative*.*
- *We call the set of events* $\mathsf{A} \wedge \neg\mathsf{B}$ *a* false positive*.*

Corollary 1 (Structural Information Control). *Under the assumptions from Def. 10 and correct traversal rules $f_{\mathbb{T},\mathbb{P}}$, from the absence of false negatives in an isolation analysis ($\neg\mathsf{A} \wedge \mathsf{B} = \emptyset$) follows that a breach of structural information control is indicated by an alarm event* A*.*

Because we modeled the mediation by dedicated guardians explicitly by traversal rules and inter-zone information flow by B events, this is by construction.

Note that the goal of the analysis system is to detect isolation breaches, that is, breaches of structural information control. We cannot prove an absence of information flow, i.e., verify structural information control, but only detect attack states. We optimize the detection thereof by minimizing the false negative rate through reduction to correct traversal rules (making sure we miss as few breaches as possible) and maximizing the Bayesian detection rate through mitigation of false positives (finding the actual needles in the haystack).

Theorem 1 (Reduction to Traversal Rules, proven in Appendix B.1). *The correct isolation modeling by traversal rules $f_{\mathbb{T},\mathbb{P}}$ implies absence of false negatives.*

5.2 Correctness of the Given Traversal Rules

The correctness of the traversal rules from Table 1, §4.4 remains to be shown, where we need to analyze on two levels: i. correctness of individual rules and ii. correctness of their composition.

Individual Rules. We examine the traversal rules in detail in Appendix B.2 and highlight the most important points here.

- *Network*: We model correct implementation of physical switches (Rule 5), VLAN en- and decapsulation and lift the properties of cryptographic secure channels (e.g., [6,19]) to VLAN tags (Rules 1, and 8, to 10).

- *Physical Machine, Hypervisor, VM Stack*: We claim secure isolation by management OS and physical machine (Rules 2 and 3) as well as cross-VM isolation (Rule 4). The former rules are elementary for virtualization security, the latter rule is arguable as it models the hypervisor's multi-tenancy capability and needs to be reconsidered depending on the actual environment (cf. [21,1] and discussion in Appendix B.2).
- *Storage*: We model secure separation by physical disks as well as by the file system (Rules 11 and 12), where the latter rule is systematically enforced by virtualization vendors (e.g., [26]) and can be checked automatically [30].

Correct Traversal Rule Composition. The composition establishes the following robustness principles:

- *Explicit Knowledge Model*: The explicit traversal rules model all and only known facts about information flow and isolation. Thus, traversal rules focus on preventing false negatives introduced by invalid assumptions.
- *Strict Over-abstraction*: When in doubt, the traversal rules must be a conservative estimate towards information flow, that is, model a super-set of potential information flow. By that, traversal rules will never introduce false negatives at the cost of additional false positives.
- *Default-Traversal Behavior*: The default rules establishing completion on the traversal rules must all be *default-follow rules*, that is, evaluate undetermined cases to 1 and log such results. Thus, the completion will only introduce false positives but never false negatives.

We conclude that the traversal rule robustness principles are all lined up to fence off false negatives, yet at the cost of false positives. Whereas this trade-off benefits a conservative security analysis, it impacts its effectiveness, as becomes manifest in its *overall detection rate.*

5.3 Overall Detection Rate

The overall detection rate of an analysis system establishes a relation between alarm and breach events, A and B, as defined in Def. 11. We analyze the effectiveness of the analysis system, in particular with respect to rejection of *false positives*, whose influence through the base-rate fallacy rate was established by Axelsson [3] in the area of intrusion detection systems.

Definition 12 (Detection Rates). *The* detection rate *is* $\mathsf{P}[\mathsf{A}|\mathsf{B}]$*, alarm contingent on breach, obtainable by testing the analysis system against scenarios known to constitute a* B *event. The* false alarm rate *is* $\mathsf{P}[\mathsf{A}|\neg\mathsf{B}]$*, the false positive rate, obtainable analogously. The* false negative rate *is* $\mathsf{P}[\neg\mathsf{A}|\mathsf{B}] = 1 - \mathsf{P}[\mathsf{A}|\mathsf{B}]$. *The* Bayesian detection rate *is* $\mathsf{P}[\mathsf{B}|\mathsf{A}]$*, that is the rate with which an alarm event implies an actual breach event. The* all-is-well rate *is* $\mathsf{P}[\neg\mathsf{B}|\neg\mathsf{A}]$*, that is the rate with which the absence of an alarm implies that all is well.*

Our goal is to maximize the Bayesian detection rate and the all-is-well rate, which we determine with Bayes Theorem:

$$\mathsf{P[B|A]} = \frac{\mathsf{P[B]} \cdot \mathsf{P[A|B]}}{\mathsf{P[B]} \cdot \mathsf{P[A|B]} + \mathsf{P[\neg B]} \cdot \mathsf{P[A|\neg B]}}$$

If we assume that the rate of breaches is low compared to the rate of non-breaches, $\mathsf{P[B]} \ll \mathsf{P[\neg B]}$, we see that the false positive rate $\mathsf{P[A|\neg B]}$ dominates the denominator of the Bayesian detection rate. This analysis asks for caution. Even though the focus on absence of false negatives implies a conservative security analysis, the presence of false positives can diminish the effectiveness of the analysis system easily.

Conjecture 1. The correctness of the traversal rules determines the absence of false negative events. The coverage of explicit correct traversal rules determines the absence of false positive events.

Although the absence of false negatives is important for the system's security, effectiveness requires the absence of false positives, as well. This is to ensure that administrators are able to find the actual breaches in the set of all alarm events. Therefore, administrators need to fine-tune the traversal rules to maximize coverage and, thus, the Bayesian detection rate.

5.4 Discussion

The transitive closure over the graph coloring securely lifts the isolation analysis to an analysis of the traversal rules $f_{\mathbb{T},\mathbb{P}}$. Therefore, the correctness of the traversal rules becomes a make-or-break criterion for the analysis method and impacts the detection rate.

We observe a *complexity reduction*: the simple traversal rules have a complexity of their relation $R \subseteq \mathbb{T} \times \mathbb{T}$. In practice, $|\mathbb{T}| \ll |V|$ as well as $|\mathbb{T}|^2 \ll |E| \leq |V|^2$, with the number of properties set for $f_{\mathbb{T},\mathbb{P}}$ being small. Therefore, the complexity of analyzing the traversal rules $f_{\mathbb{T},\mathbb{P}}$ is much smaller than the complexity of isolation analysis. This allows administrators to explicitly model and thoroughly and efficiently check their knowledge and trust assumptions about information flow and isolation.

Because our traversal rules base on the principle of *over-abstraction*, that is, resort to default-traversal in undetermined cases, the method excludes false negatives, at the cost of additional false positives. The method is therefore always on the conservative side, even though we are well aware that the false positive rate impacts the overall detection rate [3]. We provide the general analysis framework and offer user-defined traversal rules to fine-tune the analysis method to reduce false positives and maximize the Bayesian detection rate. Also, we experiment with refinement methods for a subsequent root-cause analysis to pinpoint critical information flow edges.

In principle, our tool is in a similar situation as the first intrusion detection systems. There do not exist standardized data sets to quantify and calibrate false positive and false negative rates. We approach this situation by obtaining real-world data from third parties and are currently testing the analysis method in sizable real-world customer deployments, such as the case study discussed below, to establish the detection rates.

6 Implementation

We have implemented a prototype of our automated information flow analysis in Java[1] that consists of roughly twenty thousand lines of code. Furthermore, we have additional scripts written in Python that perform post-processing for visualization purposes and refinement for root-cause analysis. The prototype consists of two main programs, that is, the discovery, and a processing and analysis program. The result of the discovery is written into an XML file and is used as the input for the analysis.

6.1 Discovery

The functionality of the discovery and its different probes were already outlined in §4.1. There exist different ways to implement a discovery probe. A probe can establish a secure console (SSH) connection to the virtualized host or the management console where commands are executed and the output is processed. Typically, the output is either XML, which is stored in the discovery XML file directly, or the output has to be parsed and transformed into XML. As alternative to the secure console, a probe can connect to a hypervisor-specific API, such as a web service, that provides information about the infrastructure configuration.

We illustrate the discovery procedure with VMware as example. Here, the discovery probe connects to *vCenter* to extract all configuration information of the managed resources. It does so by querying the VMware API with the retrieveAllTheManagedObjectReferences() call, which provides a complete iteration of all instances of ExtensibleManagedObject, a base class from which other managed objects are derived. We ensure completeness by fully serializing the entire object iteration into the discovery XML file, including all attributes.

6.2 Processing

The *processing* program consists of the transformation of the discovery XML into the realization model, the graph coloring, and the analysis of the colored realization graph.

The realization model is a class model that is used for generating Java class files. During the transformation of the XML into the realization model, instances of these classes are created, their attributes set, and linked to instances of other classes according to the mapping rules (cf. §4.2). Again, we illustrate this process for VMware. Each mapping rule embodies knowledge of VMware's ontology of virtualized resources to configuration names, for instance, that VMware calls storage configuration entries storageDevice. The edges are encoded implicitly by XML hierarchy (for instance, that a VM is part of a physical host) as well as explicitly by Managed Object References (MOR). The mapping rules establish edges in the realization model for all hierarchy links and for all MOR links between configuration entries for realization model types.

The traversal rules used for the graph coloring (cf. §4.3 and §4.4) are specified in XML. Intermediate results, such as the paths of the graph coloring, can be captured and used for further processing, i.e., visualization. We implemented Python scripts that generate input graphs for the *Gephi visualization framework* [2], such as illustrated in Figure 1.

7 Case Study

We launched a case study with a global financial institution for a performance evaluation and for further validation of detection rates and behavior in large-scale heterogeneous environments. The analyzed virtualized infrastructure is based on VMware and consists of roughly 1,300 VMs, the corresponding realization model graph of 25,000 nodes and 30,000 edges. The production system has strong requirements on isolation between clusters of different security levels, such as high-security clusters, normal operational clusters, backup clusters and test clusters. In addition, we can work with a comprehensive inventory of virtualized resources that serves as specification of an ideal state (machine placement, zone designation and VLAN configuration) and as basis for alarm validation.

We examine preliminary lessons learned, where we first consider the operation of the tool itself. The phases discovery, transformation to realization model and graph coloring executed successfully. The visualization of all results presented a challenge as a 25,000-node/30,000-edge graph overburdened the built-in visualization of the tool.

From a performance perspective, the discovery of the infrastructure using the VMware probe in combination with vCenter requires about *seven minutes*, and results in a discovery XML file with a size of *61MB*. The discovery was performed in a production environment, where network congestion and other tools using the same vCenter can have a negative effect on the discovery performance. The overall analysis of the infrastructure using the discovery XML file requires *53 seconds*, where *46 seconds* are spent on the graph coloring. This demonstrates a reasonable performance for the discovery and analysis of a mid-sized infrastructure, such as the one in our case study.

From a security perspective, the tool indeed found several realistic isolation breaches, which we highlighted by adding virtual edges between breached clusters. All isolation breaches constituted potential information flows. By that we could show actual breaches between high-security, normal operational and test clusters. We have furthermore shown that the documentation of the permitted flows was incomplete: One breach that the system identified violated the initial policy given by the customer and was fixed by augmenting the policy.

[2] `http://gephi.org/`

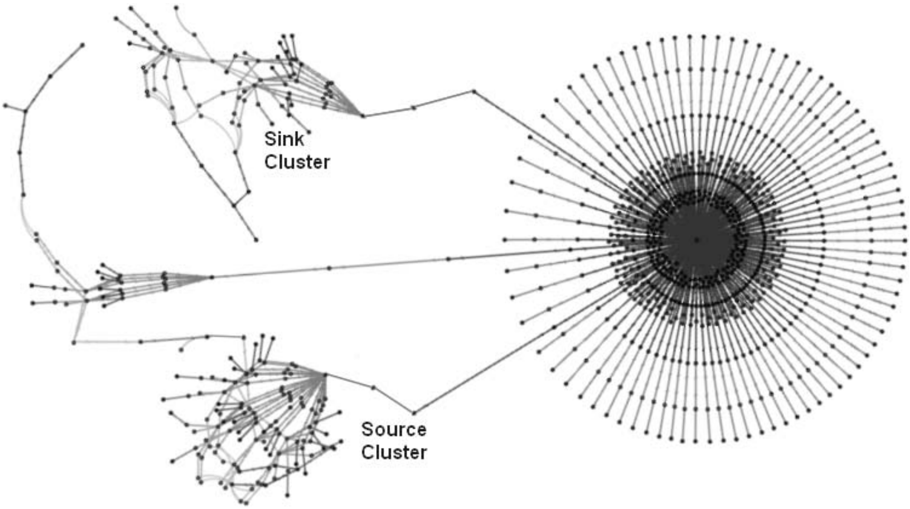


Fig. 4. Root-cause analysis of a source cluster with information flow to a sink cluster. The tree refinement derives only the sub-graphs relevant for an isolation breach. The "flower" is a large-scale switch.

Root-cause analysis answers the question which edges and nodes are ultimately responsible for the breach. We found that color spill after a traversal to a new cluster may hamper the subsequent root-cause analysis. We therefore introduced multiple automated refinement mechanisms after the graph-coloring phase to support the elimination of classes of potential root causes. *First*, we benefited greatly from a process of elimination, that is, to exclude, for instance, that information has flown over storage edges. *Second*, it was helpful to allow partial coloring, in particular to stop color propagation after detecting a breach to another cluster. *Third*, we introduced a reverse flow tree that captured which path information flow took as prelude to a breach. Figure 4 depicts an example of such a color tree: the tree is a subgraph highlighting a cross-cluster information flow path. *Fourth*, we further refined this tree by extracting critical edges, such as passed VLANs, to pinpoint routes of information flow.

In conclusion, we added a refinement phase driven by reusable Python scripts. We obtained multiple realistic alarms and could trace their root causes. The graph export to Gephi enabled the efficient visualization of root causes and information flows for human validation.

8 Conclusions and Future Work

We demonstrated an analysis system that discovers the configuration of complex heterogeneous virtualized infrastructures and performs a static information flow analysis. Our approach is based on a unified graph model that represents the configuration of the virtualized infrastructure and a graph coloring algorithm that uses a set of traversal rules to specify trust assumptions and information flow properties in virtualized systems. Based on the colored graph model, the system is able to diagnose isolation breaches, which would violate the customer isolation requirements in multi-tenant datacenters. We showed in our security analysis that we can reduce the correctness and detection rate to the correctness and coverage of the graph traversal rules. Based on existing research and systems knowledge, we submit that the present traversal rules are natural and correct.

To make this a comprehensive solution for validating virtual infrastructures in practice, there are several open questions. The first area of future work is the kind of information flows the system can model. Whereas it is currently restricted to a binary decision, i.e., whether information flows or not, future work may include different flow types (traffic types) and the bandwidth of covert channels. We also pursue research on topological covert channels, i.e., to what extent the composition of components with associated information flow rules exceeds the information flow risks of individual components. Part of this area is the modeling of imperfect guardians. For instance, firewalls let certain kinds of traffic pass and block others. We can build on the results of firewall and reachability analyses discussed in §2 to capture the behavior of such guardians and transform it into detailed guardian- and flow-type-specific traversal rules.

The second open problem is a platform-independent language to express security requirements and trust assumptions. In our current system, isolation assumptions and rules are provided in XML format, and a domain-specific language could ease the specification of requirements for customers. Bleikertz and Groß [4] proposed a first formal language to that end.

A third area of future work is to extend our approach towards other configuration properties, such as dependability. Today, misconfiguration of redundant components (network, disks, machines) often is only detected if the main component fails. We believe that our approach can be used to validate the correct configuration of such backup components to ensure correct fail-over. In addition, dynamic analysis becomes more important with increasing size of the topology and change frequency. Our current approach performs a static analysis of a given configuration state. A dynamic analysis can be emulated by running multiple static analyses and comparing the resulting realization models. However, a truly dynamic analysis needs to analyze small configuration changes and efficiently determine their effect on the topology.

Acknowledgments. We would like to thank Ray Valdez, Michael Steiner, Stefan Berger, and Dimitrios Pendarakis of the IBM Watson Research Center for valuable feedback and productive collaboration during our research. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n257243 (TClouds project: `http:/www.tclouds-project.eu`).

References

1. Aciiçmez, O.: Yet another microarchitectural attack: exploiting i-cache. In: CSAW 2007: Proceedings of the 2007 ACM Workshop on Computer Security Architecture, pp. 11–18. ACM, New York (2007)
2. Al-Shaer, E., Marrero, W., El-Atawy, A., ElBadawi, K.: Global Verification and Analysis of Network Access Control Configuration. Tech. rep., DePaul University (2008)
3. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Trans. Inf. Syst. Secur. 3(3), 186–205 (2000)
4. Bleikertz, S., Groß, T.: A virtualization assurance language for isolation and deployment. In: Proceedings of the 12th IEEE International Symposium on Policies for Distributed Systems and Networks (IEEE POLICY 2011). IEEE, Los Alamitos (2011)
5. Bleikertz, S., Schunter, M., Probst, C.W., Pendarakis, D., Eriksson, K.: Security audits of multi-tier virtual infrastructures in public infrastructure clouds. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security, CCSW 2010, pp. 93–102. ACM, New York (2010), http://doi.acm.org/10.1145/1866835.1866853
6. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002), extended version in IACR Cryptology ePrint Archive 2002/059, http://eprint.iacr.org/
7. Garfinkel, T., Rosenblum, M.: When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. In: HOTOS 2005: Proceedings of the 10th Conference on Hot Topics in Operating Systems, p. 20. USENIX Association, Berkeley (2005)
8. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20. IEEE, Los Alamitos (1982)
9. Gray III, J.W.: Toward a mathematical foundation for information flow security. In: IEEE Symposium on Security and Privacy, pp. 21–35. IEEE, Los Alamitos (1991)
10. Haigh, J.T., Young, W.D.: Extending the non-interference version of MLS for SAT. In: IEEE Symposium on Security and Privacy, p. 60. IEEE, Los Alamitos (1986)
11. Jacob, J.: Separability and the detection of hidden channels. Inf. Process. Lett. 34, 27–29 (1990), http://portal.acm.org/citation.cfm?id=79804.79852
12. Kelem, N.L., Feiertag, R.J.: A Separation Model for Virtual Machine Monitors. In: IEEE Symposium on Security and Privacy, pp. 78–86. IEEE, Los Alamitos (1991)
13. Khakpour, A.R., Liu, A.: Quarnet: A Tool for Quantifying Static Network Reachability. Tech. Rep. MSU-CSE-09-2, Department of Computer Science, Michigan State University, East Lansing, Michigan (January 2009)
14. Krothapalli, S.D., Sun, X., Sung, Y.W.E., Yeo, S.A., Rao, S.G.: A toolkit for automating and visualizing VLAN configuration. In: SafeConfig 2009: Proceedings of the 2nd ACM Workshop on Assurable and Usable Security Configuration, pp. 63–70. ACM, New York (2009)
15. Lampson, B.W.: A note on the confinement problem. Communications of the ACM 16(10), 613–615 (1973)
16. Mantel, H.: Information flow control and applications - bridging a gap -. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, pp. 153–172. Springer, Heidelberg (2001)

17. Marmorstein, R., Kearns, P.: A Tool for Automated iptables Firewall Analysis. In: ATEC 2005: Proceedings of the USENIX Annual Technical Conference, p. 44. USENIX Association, Berkeley (2005)
18. Mayer, A., Wool, A., Ziskind, E.: Fang: A Firewall Analysis Engine. In: SP 2000: Proceedings of the 2000 IEEE Symposium on Security and Privacy, p. 177. IEEE, Washington, DC, USA (2000)
19. Mödersheim, S., Viganò, L.: Secure pseudonymous channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009)
20. Percival, C.: Cache missing for fun and profit (May 2005), `http://www.daemonology.net/papers/htt.pdf`
21. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: CCS 2009: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 199–212. ACM, New York (2009)
22. Rushby, J.: Design and verification of secure systems. In: Proceedings of the Eighth ACM Symposium on Operating Systems Principles, SOSP 1981, pp. 12–21. ACM, New York (1981), `http://doi.acm.org/10.1145/800216.806586`
23. Rushby, J.: Proof of separability a verification technique for a class of security kernels. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming 1982. LNCS, vol. 137, pp. 352–367. Springer, Heidelberg (1982)
24. Rushby, J.: Noninterference, transitivity, and channel-control security policies. Tech. rep., SRI International (December 1992), `http://www.csl.sri.com/papers/csl-92-2/`
25. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21, 2003 (2003)
26. VMware: Providing LUN Security (March 2006), `http://www.vmware.com/pdf/esx_lun_security.pdf`
27. Wojtczuk, R.: Adventures with a certain Xen vulnerability (in the PVFB backend) (October 2008), `http://invisiblethingslab.com/pub/xenfb-adventures-10.pdf`
28. Wool, A.: Architecting the Lumeta Firewall Analyzer. In: SSYM 2001: Proceedings of the 10th Conference on USENIX Security Symposium, p. 7. USENIX Association, Berkeley (2001)
29. Xie, G., Zhan, J., Maltz, D., Zhang, H., Greenberg, A., Hjalmtysson, G., Rexford, J.: On static reachability analysis of IP networks. In: INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies, March 13-17, vol. 3, pp. 2170–2183. IEEE, Los Alamitos (2005)
30. Yang, J., Twohey, P., Engler, D., Musuvathi, M.: Using model checking to find serious file system errors. ACM Trans. Comput. Syst. 24, 393–423 (2006), `http://doi.acm.org/10.1145/1189256.1189259`

Notes

[1] IBM, PowerVM and pSeries are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Java is a registered trademark of Oracle and/or its affiliates. Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. Other product and service names might be trademarks of IBM or other companies.

A Information Flow Types

We borrow concepts from information flow analysis, where we focus on information flow as the deterministic propagation of discrete units of information throughout a system.

A.1 Flow Types

Information flow analysis of multi-tenant configurations in virtualized environments analyzes *overt* and *covert* channels.[3] An *overt channel* is intended for communication; a principal can read or write on that channel within the limits of some access control policy.

Lampson [15] introduced the term *covert channel* as a channel not intended for information transfer at all. Consider a malware in VM Alice which attempts to transfer information to another instance of the malware in VM Bob, both hosted on the same hypervisor. The malware on VM Alice can, for instance, monopolize a resource[4] to transmit a bit observed by the malware on VM Bob in performance or through-put decrease.

We perceive covert channels to be a common phenomenon in virtualized infrastructures. Requiring the absence of all covert channels from hypervisors, physical hosts and resources, will render many resulting system impractical. Therefore, we allow administrators to specify a certain amount of covert channel information flow as tolerable.

Requirement Definition. We informally stated our security goal as *isolation* between zones, which sounds similar to *non-interference* [8,9]. This requirement enforces that actions in one zone do not have any effect on subsequent behavior or outputs in another zone.

The transitivity of non-interference renders it, however, unsuitable to model our setting, in which information flow via guardians may be permitted, whereas the corresponding direct flow is disallowed. Agreeing to the arguments of Rushby [24] and Mantel [16], we would need *intransitive non-interference* [10] to start with. Furthermore, the existing definitions are based on traces of steps and, thus, inherently dynamic[5], whereas we aim at a high-level static information flow analysis (its topology and communication links). Therefore, we preclude a step/trace-based non-interference analysis.

Another candidate is the analysis for *separation*, e.g. [23,12]: one removes all guardians from the system and verifies that the remaining parts are perfectly separated; however this approach was criticized by Jacob [11].

[3] This is similar to the analysis of explicit and implicit information flow on high and low variables [25].

[4] Examples include reserving a bus, launching expensive computations, flooding a cache, sending many network packets.

[5] To that end, Haigh and Young [10] have shown that it is necessary to analyze the complete trace of actions subsequent to a given action to validate that the action is allowed to interfere with another zone.

The concept of *channel control* [22] sounds interesting, as it captures our requirement to specify *exceptions* to the general zoning requirements. For instance, two zones should not communicate with each other *unless* a guardian mediates and filters the communication. In our case, however, we are not studying single channels, but a complex topology of channels.

B Security of Information Flow Analysis

B.1 Reduction to Traversal Rules

Theorem 2 (Reduction to Traversal Rules, Theorem 1). *The correct isolation modeling by traversal rules $f_{\mathbb{T},\mathbb{P}}$ implies absence of false negatives.*

We prove Theorem 1 by contradiction and induction over the length n back-trace graph traversal. The proof by itself is straight-forward as graph coloring (Def. 7) is a recursive definition.

Proof. Let sets of types $\mathbb{T}$ and properties $\mathbb{P}$, a valid graph (V, E) and information sources $\hat{V}$ and sinks $\check{V}$ be given.

Suppose a false negative event $N \in (\neg \mathsf{A} \wedge \mathsf{B}) \neq \emptyset$. By definition, there exists a breach, that is a sink $\check{v} \in \check{V}$ for which holds that it is colored by a source $\hat{v} \in \hat{V}, \hat{v} \neq \check{v}$.

Initialize a set $\boldsymbol{E} = \emptyset$.

Induction start $n = 1$: the sink $\check{v}$ is colored because of the breach event B.

Assume the induction statement true for $n - 1$: A colored vertice v_{n-1} could only have been colored if

(a) v_{n-1} is source $\hat{v}$ with the corresponding color (then we are done and output $\boldsymbol{E}$) or

(b) there exists an edge $e = (v_n, v_{n-1})$ with $v_n = (\cdot, t_n, p_n)$ and $v_{n-1} = (\cdot, t_{n-1}, p_{n-1})$ for which holds: v_n is colored and the traversal rules $f_{\mathbb{T},\mathbb{P}}(t_n, t_{n-1}, p_n, p_{n-1})$ evaluate to follow. Accumulate $\boldsymbol{E} := \boldsymbol{E} \cup \{e\}$.

If the induction succeeds, then $\boldsymbol{E}$ is a construction of a path between source $\hat{v}$ and sink $\check{v}$, thus an alarm event, A. We obtain a contradiction against $N \in \neg\mathsf{A}$.

Consequently, for any case in which no sink-source path can be constructed, there exists an edge e for which the traversal rules $f_{\mathbb{T},\mathbb{P}}$ evaluate to stop. This reduces the false negative to the correctness for the traversal rules.

B.2 Inspection of Individual Traversal Rules

First, let us analyze the rules for network switches and VLAN traffic. Rule 5 assume a correct implementation of an isolation by network switches for switched-off ports. Rules 6 and 7 establish the VLAN en- and decapsulation by network switches and are interesting for the security analysis. The rules assign a VLAN-specific color to information flow for in-ports with VLAN tagging and only allow information traversal at out-ports with matching VLAN tags. This models the VLANs' traffic separation by encryption lifted to VLAN tags as well as a cross-session key separation assumption, standard for secure channels: messages encrypted under one VLAN tag cannot interfere with messages encrypt under other VLAN tags and can only be decrypted under the same VLAN tag. We can derive these properties from research on secure channels and their parallel composition (in cryptography for instance Canetti and Krawczyk's work on UC secure channels [6]; in formal methods for instance Mödersheim and Viganò's formalization of secure pseudonymous channels [19].) Rule 9 stops information flow at ports with non-matching VLAN tags accordingly. Rule 8 has information flow follow through for trunked VLAN ports. Otherwise, we assume that the network and physical switches securely configure and implement VLAN traffic isolation for flows from switch to port (Rules 10 and 1). We conclude that these assumptions are natural and model correct network behavior.

Second, let us consider the stack of physical machine, hypervisor and VMs. Rules 2 and 3 make the assumptions that a management OS and physical host provide secure isolation and that all information flow is accounted for explicitly. These assumptions are necessary for virtualization security, as information leakage from these components can subvert the entire system's security, and model standard trust assumptions. Rule 4 is interesting as it assumes that hypervisors sufficiently separate VMs against each other, that is, that information flow through cross-VM covert channels can be neglected. Research results exist that highlight cross-VM covert channels, for instance [21,1]. Therefore, this trust assumption on the hypervisor's multi-tenancy capability must be subject to thorough debate.[6] Whereas the isolation assumptions on physical machine and management OS are natural and well founded, we conclude that the modeling of covert channels is a key trust decision for the hypervisor model.

Third, let us consider the information model for storage. Rule 11 models that the storage controllers are capable of separating information flow to physical disks, whereas Rule 12 establishes that the file system prevents cross file information flow through its access control enforcement. The former property is systematically enforced by virtualization vendors, such as VMware [26] that do not allow reconfiguration of storage back-ends for VMs. The latter property found attention in research and can be checked with tool support [30]. We conclude that both assumptions are natural to make.

[6] For high-security environments, we recommend to set this rule to follow and therefore only relying on physical separation, yet dismissing hypervisor multi-tenancy.

Scalable Analysis of Attack Scenarios*

Massimiliano Albanese[1], Sushil Jajodia[2], Andrea Pugliese[3],
and V.S. Subrahmanian[1]

[1] University of Maryland
{albanese,vs}@umiacs.umd.edu
[2] George Mason University
jajodia@gmu.edu
[3] Unversity of Calabria
apugliese@deis.unical.it

Abstract. Attack graphs have been widely used for attack modeling, alert correlation, and prediction. In order to address the limitations of current approaches – scalability and impact analysis – we propose a novel framework to analyze massive amounts of alerts in real time, and measure the impact of current and future attacks. Our contribution is threefold. First, we introduce the notion of generalized dependency graph, which captures how network components depend on each other, and how the services offered by an enterprise depend on the underlying infrastructure. Second, we extend the classical definition of attack graph with the notion of timespan distribution, which encodes probabilistic knowledge of the attacker's behavior. Finally, we introduce attack scenario graphs, which combine dependency and attack graphs, bridging the gap between known vulnerabilities and the services that could be ultimately affected by the corresponding exploits. We propose efficient algorithms for both detection and prediction, and show that they scale well for large graphs and large volumes of alerts. We show that, in practice, our approach can provide security analysts with actionable intelligence about the current cyber situation, enabling them to make more informed decisions.

Keywords: Attack graphs, dependency graphs, vulnerability analysis, cyber situation awareness, scalability.

1 Introduction

Attack graphs have been widely used to model the possible ways an attacker can exploit network vulnerabilities, and to correlate alerts. There has been extensive work on automatically deriving attack graphs from network scans [10,11]. However, existing approaches to alert correlation typically have two major limitations. First, attack graphs do not provide a mechanism to evaluate the likelihood of each attack pattern or its impact on the enterprise. Second, although scalable

* The work presented in this paper is supported in part by the Army Research Office MURI award number W911NF-09-1-0525.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 416–433, 2011.

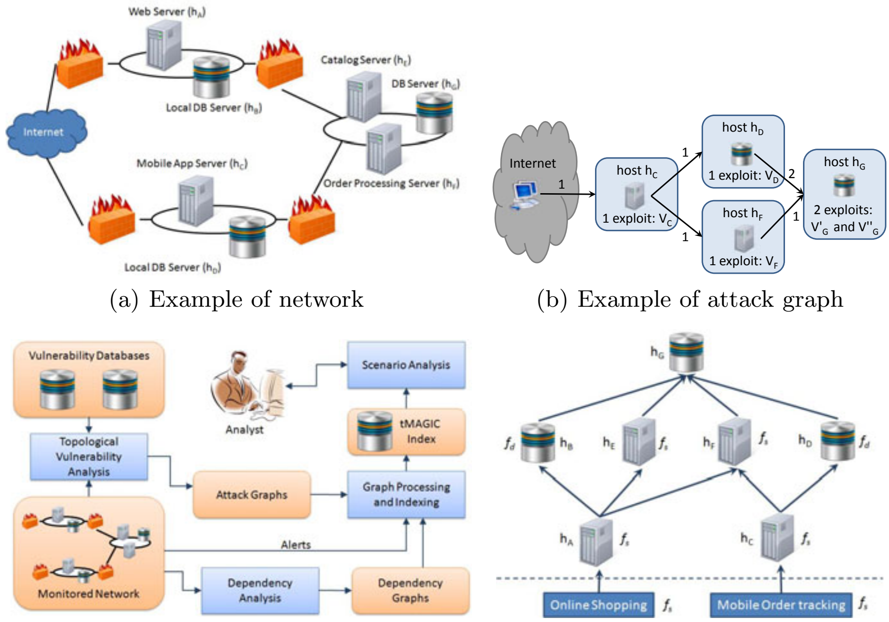
generation of attack graphs has been studied [10], scalability issues with respect to the alert correlation process have not been fully addressed yet.

In order to address the above limitations, we propose a novel framework to analyze massive amounts of raw security data in real time, and measure the impact of current and future attacks. First, we introduce the notion of *generalized dependency graph*, which captures how network components depend on each other, and how the services offered by an enterprise depend on the underlying infrastructure. Second, we extend the classical definition of attack graph with the notion of *timespan distribution*, which encodes probabilistic knowledge of the attacker's behavior. Third, we introduce the notion of *attack scenario graph*, which combines dependency and attack graphs, bridging the gap between known vulnerabilities and the missions or services that could be affected by the corresponding exploits. In the last several years, there has been significant work in dependency modeling [13] and automatic discovery of dependencies [3]. However, to the best of our knowledge, we are the first to combine attack and dependency graphs to enhance situation awareness. In practice, the proposed framework provides security analysts with a more complete picture of the cyber situation. In fact, combing attack and dependency graphs not only provides analysts with a set of possible future scenarios, but also estimates their probability and potential impact. Finally, in order to guarantee scalability, we propose efficient algorithms to track and index ongoing attacks and analyze future scenarios, and show that they scale well for large graphs and large volumes of incoming alerts.

Throughout this paper, we will use the network of Figure 1(a) as a running example. This network offers two services, *Online Shopping* and *Mobile Order Tracking*, and consists of three subnetworks delimited by firewalls. Two subnetworks implement the two services, and each of them includes a host accessible from the Internet. The third subnetwork implements the business logic, and includes a central database server. An attacker who wants to steal sensitive data from the main database server will need to breach the firewalls and gain privileges on several hosts before reaching the target. The attack graph of Figure 1(b) shows that, once a vulnerability V_C on the Mobile Application Server has been exploited, we can expect the attacker to exploit either V_D or V_F. However, the attack graph alone does not answer the following important questions: Which vulnerability has the highest probability of being exploited? Which attack pattern will have the largest impact? How can we mitigate the risk? Our framework is designed to answer these questions efficiently.

The architecture of the proposed framework is sketched in Figure 1(c). We assume that attack and dependency graphs are given, and combine them into an attack scenario graph (Section 5). Incoming alerts are matched against such data structure and indexed in real-time.

The paper is organized as follows. Section 2 discusses related work. Section 3 introduces the notion of *generalized dependency graph*, whereas Section 4 defines *probabilistic temporal attack graphs*. Section 5 introduces the notion of *attack scenario graph*, and Section 6 presents the proposed index data structure, along with an algorithm to update the index in real-time. Section 7 presents an



(a) Example of network (b) Example of attack graph

(c) System architecture (d) Example of dependency graph

Fig. 1. Running example and system architecture

algorithm to analyze and rank future possible scenarios. Finally, Section 8 reports experimental results, and Section 9 presents conclusions.

2 Related Work

To reconstruct attack scenarios from isolated alerts, some correlation techniques employ prior knowledge about attack strategies [5] or alert dependencies [17]. Other techniques aggregate alerts with similar attributes [21] or statistical patterns [20]. To the best of our knowledge, the limitation of the nested-loop approach [22], especially for correlating intensive alerts in high-speed networks, has not been addressed. In [19], Noel *et al.* adopt a vulnerability-centric approach to alert correlation, because it can effectively filter out bogus alerts. However, the nested loop procedure is still used in [19]. Attack scenarios broken by missed alerts are reassembled by clustering alerts with similar attributes [18]. Designed for a different purpose, the RUSSEL language is similar to our approach in that the analysis only requires one-pass of processing [8].

With respect to the general problem of modeling activities, Hidden Markov Models and their variants have been used extensively. For instance, Duong et al. [6] introduce the Switching Hidden Semi-Markov Model, a two-layered extension of the Hidden Semi-Markov Model. Dynamic Bayesian Networks [9] and probabilistic extensions of Petri Nets [1] have also been used for tracking and recognizing multi-agent activities. A survey of temporal concepts and data models used in unsupervised pattern mining from symbolic temporal data is presented in [16]. In recent years, there has been extensive research in Data Stream Management Systems [7]. However, the scope of our work is drastically different. In fact, we are not interested in retrieving sets of data items satisfying certain conditions and updating such sets as new data is received. Instead, we are interested in identifying sets of alerts that, with a probability above a given threshold, constitute the "evidence" that a cyber attack occurred.

In large enterprise networks, the performance of an application may depend on many hosts and network components. Recently, automated discovery of dependencies from network traffic [2,12] has been proposed. Chen et al. [4] present a comprehensive study of the performance and limitations of this class of dependency discovery techniques, and introduce a new system, Orion, that discovers dependencies using packet headers and timing information in network traffic.

To the best of our knowledge, there has been virtually no work on efficient indexing to support scalable and real-time attack detection. Similarly, there has been no work on integration of attack and dependency graphs to enhance situation awareness. In conclusion, our work differs from previous works in two major ways. First, we provide a mechanism to index alerts and recognize attacks in real-time. Second, we provide a mechanism to integrate attack and dependency graphs and enable real-time scenario analysis and better security decisions.

3 Generalized Dependency Graphs

Modern distributed systems typically consist of a large number of interdependent hardware and software components. Some dependencies may not even be explicit, and Leslie Lamport's famous quote *"a distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"* [14] greatly captures the essence of the problem. In the following, we will use the term *network entity*, or simply *entity*, to refer to any component of a distributed system at any level of abstraction (e.g., subnet, host, application, service). Dependencies amongst network entities can be broadly classified in the following three categories: *redundancy* (a network entity depends on a redundant pool of resources), *graceful degradation* (a network entity depends on a pool of entities; if one fails, the system can continue to work with degraded performance), and *strict dependence* (a network entity strictly depends on a pool of other entities; if one fails, the dependent entity becomes unusable).

In order to model all possible scenarios, we first assume that the status of each network entity can be represented as a performance indicator on a scale from 0 to 1, 1 meaning that the entity is fully working, and 0 meaning that the entity is unusable. We then assume the existence of a family $\mathcal{F}$ of functions of the form $f : [0,1]^n \rightarrow [0,1]$, which evaluate the performance of an entity given the performance of the n entities it depends on. We call these functions *dependency functions*, and require that $f(0,\ldots,0) = 0$ and $f(1,\ldots,1) = 1$.

The equations below define dependency functions corresponding to the three scenarios described above. If a dependency is described by f_r, then h will be fully operational if at least one of the entities it depends on is fully operational. If the dependency is described by f_s, all the entities need to be fully operational for h to be fully operational.

$$f_r(l_1,\ldots,l_n)=\begin{cases}1, \text{ if } \exists i\in[1,n] \text{ s.t. } l_i=1\\ 0, \text{ otherwise}\end{cases} \qquad f_d(l_1,\ldots,l_n)=\tfrac{1}{n}\textstyle\sum_{i=1}^{n} l_i$$

$$f_s(l_1,\ldots,l_n)=\begin{cases}1, \text{ if } \forall i\in[1,n]\, l_i=1\\ 0, \text{ otherwise}\end{cases}$$

We can now introduce a generalization of dependency graphs that can capture not only which entities depend on which other entities, but also the nature of such dependencies. Service dependency models have been widely studied. In [13], Kheir *et al.* propose a service dependency representation that enables intrusion and response impact evaluation. However, although it is possible to automatically discover dependencies [3], the task of understanding the nature of such dependencies has not been fully automated yet. For the purpose of the proposed framework, we will assume that dependency graphs generated via existing tools are manually augmented by system administrators. Further research will be needed to fully automate this process.

Definition 1 (Generalized Dependency Graph). *A* generalized dependency graph *is a labeled directed acyclic graph $D=(H,Q,\phi)$, where*

- *H is a set of nodes, corresponding to network entities;*
- *$Q=\{(h_1,h_2)\in H\times H \mid h_1 \text{ depends on } h_2\}$ is a set of edges;*
- *$\phi : H\rightarrow\mathcal{F}$ is a mapping that associates with each node $h\in H$ a function $f\in\mathcal{F}$, s.t. the arity of f is equal to the outdegree of h*[1].

For each node $h\in H$ we will use $h^{\cdot}$ to denote the set of entities that depend on h and ${}^{\cdot}h$ to denote the set of entities h depends on.

Figure 1(d) shows a dependency graph for the network of Figure 1(a). Both h_E and h_F strictly depend on h_G. Local database servers also depend on h_G, but they can continue to operate, with degraded performance, if h_G is compromised.

Without loss of generality, we assume an arbitrary but fixed time granularity, and use $\mathcal{T}$ to denote the set of all time points. We can now introduce the definition of *network status function*, which, for each time point t, maps a network entity to its performance at time t.

Definition 2 (Network Status Function). *Given a generalized dependency graph $D=(H,Q,\phi)$, a* network status function *for D is a function $s : H\times\mathcal{T}\rightarrow[0,1]$ such that $\forall h\in H$ and $\forall t\in\mathcal{T}$, $s(h,t)\leq f(s(h_{i_1},t),\ldots,s(h_{i_m},t))$, where $f=\phi(h)$, and ${}^{\cdot}h=\{h_{i_1},\ldots,h_{i_m}\}$ is the set of entities h depends on.*

[1] If h is a terminal node in the dependency graph (i.e., it does not depend on any other node), we assume $\phi(h)$ is the constant (0-ary) function 1.

The performance of an entity h is bounded by its corresponding dependency function, which represents a theoretical maximum. In practice, the performance may be lower if h has been directly compromised by an attack.

4 Attack Modeling

In this section, we start by recalling the definition of *attack graph* presented in [22], and then introduce the notion of *probabilistic temporal attack graph*. An attack graph is a directed graph having two types of vertices, exploits and security conditions. Exploits are host-bound vulnerabilities, whereas security conditions refer to the network states required or implied by exploits, such as privilege levels. In [22], an attack graph is formally defined as a directed graph $G = (V \cup C, R_r \cup R_i)$, where V is the set of known exploits, C is the set of relevant security conditions, and R_r and R_i denote the *require* and *imply* relationship between exploits and security conditions, defined as $R_r \subseteq C \times V$ and $R_i \subseteq V \times C$, respectively. The *prepare-for* relationship between exploits is the composite relation $R_i \circ R_r$. For the purpose of the type of analysis we present in this paper, we can assume, without loss of generality, that attack graphs are acyclic.

We extend the attack graph model discussed above with the notion of *timespan distribution*, which encodes probabilistic knowledge of the attacker's behavior as well as temporal constraints on the unfolding of attacks. We assume that each step of an attack sequence is completed within certain temporal windows after the previous exploit has been executed, each associated with a probability.

Example 1. Suppose an attacker has gained some privileges on host h_E in Figure 1. Using these privileges, he can then create the conditions to exploit a vulnerability on the main database server. However, this will take an amount of time depending on his skill level. The attacker will then have time to exploit the vulnerability until the vulnerability itself is patched, or the attack is discovered.

Leversage and Byres [15] describe how to estimate the *mean time to compromise* a system and relate that to the skill level of the attacker. This approach can be generalized to estimate timespan distributions for individual vulnerability exploits. In fact, we can assume that the time taken to exploit a vulnerability varies with the skill level of the attacker. Additionally, some vulnerabilities are easier to exploit than others, thus exhibiting higher probabilities. We will assume that timespan distributions are given, as a detailed discussion on how to derive them would be beyond the scope of this paper. The definitions of *timespan distribution* and *probabilistic temporal attack graph* are given below.

Definition 3 (Timespan Distribution). *A* timespan distribution $\omega \in \Omega$ *is a pair* (I, ρ) *where:*

- *I is a set of time intervals*[2] *s.t.* $\forall [x,y] \in I, x \leq y$*;*
- $\forall [x,y], [x',y'] \in I$ *s.t.* $[x,y] \neq [x',y']$*, intervals* $[x,y]$ *and* $[x',y']$ *are disjoint;*

[2] A time interval is a closed interval of the set $\mathcal{T}$ of all time points.

- *$\rho : I \rightarrow [0,1]$ is a function that associates a value $\rho(x,y) \in [0,1]$ with each time interval $[x,y] \in I$.*

Given a timespan distribution $\omega = (I,\rho)$, we use $S(\omega)$ to denote $\sum_{[x,y]\in I} \rho(x,y)$, and $\omega.t_{max}$ to denote $\max_{[x,y]\in I \,|\, \rho(x,y)>0} y$, i.e., the maximum time point for which ρ is not 0. We require that $S(\omega) \leq 1$. □

Intuitively, a timespan distribution (I,ρ) specifies a set I of disjoint time intervals when a given exploit might be executed, and an incomplete probability distribution ρ over I: $\rho(x,y)$ is the probability that the exploit will in fact be executed during the time interval $[x,y]$, following the execution of the previous exploit. ρ is not forced to be complete as the exploit might not be executed.

Definition 4 (Probabilistic Temporal Attack Graph). *Given an attack graph $G = (V \cup C, R_r \cup R_i)$ a* probabilistic temporal attack graph *built on G is a labeled directed acyclic graph $A = (V,E,\delta,\gamma)$ where:*

- *V is the finite set of vulnerability exploits in the attack graph;*
- *$E = R_i \circ R_r$;*
- *$V^s = \{v \in V \mid \nexists v' \in V \text{ s.t. } (v',v) \in E\} \neq \emptyset$, and $V^e = \{v \in V \mid \nexists v' \in V \text{ s.t. } (v,v') \in E\} \neq \emptyset$, i.e., there exists at least one* start node *and one* end node*;*
- *$\delta : E \rightarrow \Omega$ is a function that associates a timespan distribution with each edge in the graph, such that $(\forall v \in V) \sum_{\{v' \in V | (v,v') \in E\}} S(\delta(v,v')) = 1$;*
- *γ is a function that associates with each exploit $v_j \in V \setminus V^s$ the condition*

$$\gamma(v_j) = \bigwedge_{c_k \in C \text{ s.t. } (c_k,v_j)\in R_r} \left(\bigvee_{v_i \in V \text{ s.t. } (v_i,c_k)\in R_i} \nu_{i,j} \right),$$

where $\nu_{i,j}$ denotes that v_j must be preceded by v_i and must be executed within one of the time intervals defined by $\delta(v_i,v_j)$, following the execution of v_i. □

Intuitively, an edge $e = (v_i,v_j)$ in a probabilistic temporal attack graph indicates that vulnerability exploit v_i *prepares for* exploit v_j. The timespan distribution $\delta(e) = (I,\rho)$ labeling the edge indicates how the probability that v_j is executed after v_i changes with time. The condition $\gamma(v_j)$ labeling a node v_j encodes the dependencies between v_j and the exploits preparing for it. In the original attack graph, each exploit v_j may require one or more security conditions to be satisfied, and each such condition may be achieved through several alternative exploits. This explains the conjunctive normal form of $\gamma(v_j)$. In the following, we will often abuse notation, and use v_i instead of $\nu_{i,j}$ in condition $\gamma(v_j)$, when v_j is clear from the context. Additionally, we say that a set V^* of exploits satisfies condition $\gamma(v_j)$ if exploits in V^* are sufficient to satisfy all the security conditions *required* by v_i.

Attacks can be executed in many different ways, which we will refer to as *instances*. The notion of attack instance is formalized in the following definition.

Definition 5 (Attack Instance). *Given a probabilistic temporal attack graph $A = (V,E,\delta,\gamma)$, an* instance *of A is a tree $T = (V_T, E_T)$ over A, rooted at an end node of A and defined as follows:*

- $|V_T \cap V^e| = 1$, *i.e., there is exactly one end node of* A *in* T;
- $(\forall v \in V_T \backslash V^s)\ \exists V' \subset V_T$ *s.t.* V' *satisfies* $\gamma(v) \wedge \nexists V'' \subset V'$ *s.t.* V'' *satisfies* $\gamma(v)$. □

An instance is represented as a tree because each attack pattern aims at creating a certain target condition (e.g., gaining access to the database server of Figure 1(a)), which can be achieved by executing an exploit corresponding to an end node in the temporal attack graph (the root of the tree). Each exploit may require one or more exploits being executed in preparation for it, and the leaves of the tree represent exploits which depend on initial security conditions.

5 Combining Attack and Dependency Graphs

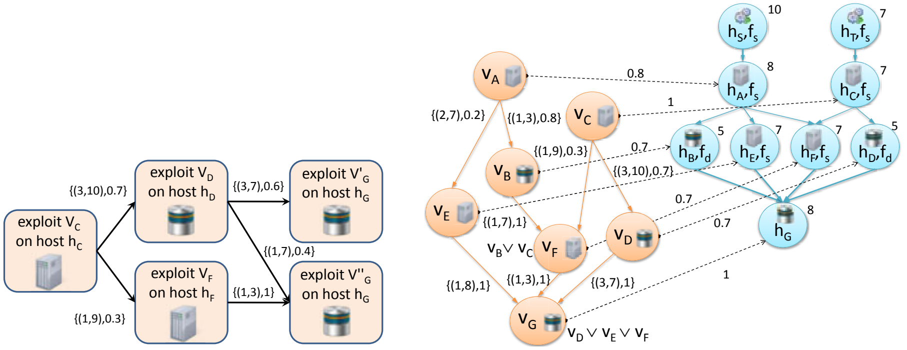
Dependency and attack graphs are great tools to analyze raw security data and derive useful insights on the structure and weaknesses of a network. The following example clearly shows that neither of these tools, if used alone, can provide a cyber security analyst with enough information to make good decisions. Instead, we need a tool that can combine information from the two graphs, also enabling the analyst to import his risk analysis knowledge.

Example 2. Consider the network of Figure 1(a), and suppose an attacker has compromised host h_C. Based on the attack graph of Figure 2(a), an automated system may recommend the analyst to protect host h_D, as a vulnerability on h_D is the most likely to be exploited next. However, when we also consider information in the dependency graph of Figure 1(d), this clearly turns to be the worst choice possible. In fact, the only entity which directly depends on h_D is host h_C, which is already compromised. Thus, the marginal damage caused by the attacker exploiting h_D is negligible. Conversely, exploiting the the vulnerability on host h_F, although less likely, would cause a much larger damage because it would also disrupt the Web Server and the Online Shopping service.

The key observation behind our approach to combining attack and dependency graphs is that the execution of a vulnerability (i.e., a node in the attack graph) might cause a more or less severe reduction in performance in one or more network entities (nodes in the dependency graph). This, in turn, may affect other entities not directly affected by the exploit. We can thus introduce the following fundamental definition.

Definition 6 (Attack Scenario Graph). *Given a probabilistic temporal attack graph* $A = (V, E, \delta, \gamma)$ *and a generalized dependency graph* $D = (H, Q, \phi)$, *an* attack scenario graph *is a 4-tuple* (A, D, F, η) *where*

- $F \subseteq V \times H$;
- $\eta : F \rightarrow [0, 1]$ *is a function that associates with each pair* $(v, h) \in F$ *a real number in the* $[0, 1]$ *interval representing the percentage reduction in the performance of entity* h *caused by vulnerability exploit* v. □



(a) Example of temporal attack graph (b) Example of attack scenario graph

Fig. 2. Attack and scenario graphs

Intuitively, attack scenario graphs merge attack and dependency graphs introducing edges between vulnerability exploits and network entities encoding how the latter are affected by the former. The numbers labeling these edges represent the percentage reduction in the performance of an entity caused by an exploit (if $\eta(v,h) = 0$ the edge is omitted). Figure 2(b) shows an example of attack scenario graph for the network of Figure 1(a), where h_S and h_T represents the two services offered by the enterprise.

We can use attack scenario graphs to track the evolution of an attack, monitor the status of the network, and assess damage at the same time. Without loss of generality, we assume that $\forall h \in H$, $s(h,t_0) = 1$, i.e., every component of the network is fully operational at time t_0. In the following, we will abuse notation and use $s_i(h)$ to denote $s(h,t_i)$. If exploit v has been executed at time t_i, we can evaluate the status of the network at time t_i as follows:

$$s_i(h) = \min\left((1-\eta(v,h)) \cdot s_{i-1}(h), f(s_i(h_{i_1}), \ldots, s_i(h_{i_m}))\right) \tag{1}$$

where $f = \phi(h)$, and $\eta(v,h) = 0$ when $(v,h) \notin F$. The status of the network needs to be updated every time a successful exploit is detected. However, given the structure of the dependency graph, we only need to update the status of components directly or indirectly affected by the exploit. In order to assess damage, we assume that each network entity h has a given theoretical utility $u(h)$ (corresponding to the case when h is fully operational). We also assume that the actual utility at time t_i is proportional to the performance of h at time t_i and is given by $u_i(h) = s_i(h) \cdot u(h)$. We then estimate the marginal damage caused by an exploit at time t_i as the loss of utility w.r.t. the situation at time t_{i-1}.

$$\Delta damage_i = \sum_{h \in H} (s_{i-1}(h) - s_i(h)) \cdot u(h) \tag{2}$$

Example 3. Consider the attack scenario graph of Figure 2(b). Suppose that $\forall h \in H$, $s_0(h) = 1$ and an attacker exploits vulnerability v_C at time t_1. This makes h_C totally unusable (as $\eta(v_C, h_C) = 1$), leading to $s_1(h_C) = 0$. As h_T strictly depends on h_C, we also have $s_1(h_T) = 0$. Therefore, the marginal damage of exploit v_C is $\Delta damage_1 = 14$. At time t_2, the attacker may take one of two possible steps: exploiting v_D (with probability 0.7) or v_F (with probability 0.3). In the first case, $\Delta damage_2 = 0.7 \cdot 5 = 3.5$ (h_C and h_T are already unusable because of the previous exploit). In the second case, $\Delta damage_2 = 0.7{\cdot}7+8+10 = 22.9$ (h_F is partially compromised, making h_A and h_S unusable). In conclusion, the second alternative, although less likely, can cause a much larger damage.

When attack and dependency graphs are very large (millions of nodes), portions of them could be maintained on disk rather than in main memory. In this case, a merged data structure would provide additional advantages over two separate graphs. If nodes from the two graphs pertaining the same group of machines are stored in the same block on disk, all the information needed to process alerts from that set of machines could be loaded with a single access to disk.

6 Real-time Scenario Analysis

In this section, we formalize the problem of real-time scenario analysis, and propose a data structure to monitor and index incoming alerts, as well as an algorithm to update the index in real-time. For the purpose of our analysis, we assume that each alert o is a tuple $(type, ts, h_{src}, h_{dest})$, where $type$ denotes the event type, $ts \in T$ is a timestamp, and h_{src}, h_{dest} are the source and destination host respectively. We refer to a sequence O of such alerts as the *observation sequence*. Finally, we assume the existence of a function $exploit : O \rightarrow V$ that maps alerts in O to vulnerability exploits. Informally, we define an *occurrence* of an attack as a sequence of alerts constituting the evidence that a given attack occurred. A formal definition is given below.

Definition 7 (Attack Occurrence). *Given a probabilistic temporal attack graph $A = (V, E, \delta, \gamma)$, and an observation sequence O, an occurrence of A in O with probability p is a sequence $O^* = \langle o_1^*, \ldots, o_k^* \rangle \subseteq O$ such that:*

- $o_1^*.ts \leq o_2^*.ts \leq \ldots \leq o_k^*.ts$;
- *there exists an instance $T = (V_T, E_T)$ of A s.t. $(\forall (v_i, v_j) \in E_T)\ \exists o_i, o_j \in O^*$ s.t. $exploit(o_i) = v_i \wedge exploit(o_j) = v_j \wedge o_i.ts \leq o_j.ts$, i.e., O^* includes alerts corresponding to all the exploits in T;*
- *$prob(root(V_T)) \geq p$, where prob is recursively defined as*

$$(\forall v \in V_T)\, prob(v) = \begin{cases} 1, \text{ if } v \text{ is a leaf node} \\ \Pi_{v' \in children(v)} \rho(x, y) \cdot prob(v') \end{cases}$$

where $(I, \rho) = \delta(v', v)$, $[x, y] \in I$ s.t. $x \leq o.ts - o'.ts \leq y$, and $exploit(o) = v$, $exploit(o') = v'$.

The span *of the occurrence above as the time interval $span(O^*) = [o_{i_1}.ts, o_{i_k}.ts]$.*□

In other words, an occurrence of a probabilistic temporal attack A is a sequence of alerts indicating that all the exploits in an instance of the temporal attack graph have been executed in the right order and satisfying all the temporal constraints, thus enabling the attacker to achieve his target security condition.

We are interested in recognizing occurrences of attacks in real-time. Note that multiple concurrent attacks generate interleaved alerts in the observation sequence. In general, the problem of finding all possible occurrences of attacks is exponential in the level of concurrency (i.e., the maximum number of concurrent attacks), and quadratic in the number of received alerts. However, we will show that, leveraging temporal features of attack graphs, it is possible to compute all the valid solutions in time linear to the number of received alerts.

In order to concurrently monitor multiple types of attacks, we first merge all probabilistic temporal attack graphs from a set $\mathcal{A} = \{A_1, \ldots, A_k\}$ – with $A_i = (V_i, E_i, \delta_i, \gamma_i)$ and $I_{\mathcal{A}} = \{id(A_1), \ldots, (A_k)\}$ – into a single *multi-attack graph* $M = (V_M, I_{\mathcal{A}}, \delta_M, \gamma_M)$ where (i) $V_M = \cup_{i=1}^{k} V_i$ is a set of vulnerability exploits; (ii) $\delta_M : V_M \times V_M \times I_{\mathcal{A}} \rightarrow \Omega$ is a function that associates a triple $(v, v', id(A_i))$ with $\delta_i(v, v')$, if $(v, v') \in E_i$, or `null` otherwise; (iii) γ_M is a function that associates each pair $(v, id(A_i)) \in V_M \times I_{\mathcal{A}}$ with $\gamma_i(v)$, if $v \in V_i \setminus V_i^s$, or `null` otherwise. In the following, we will assume that an attack scenario graph (Definition 6) is a 4-tuple (M, D, F, η), where M is a multi-attack graph, rather than a single attack graph. We can now give the definition of *attack scenario index*, a data structure that enables indexing and tracking of ongoing attacks, and provides support for analysis of future scenarios.

Definition 8 (Attack Scenario Index). *Let $G = (M, D, F, \eta)$ be an attack scenario graph, where $M = (V_M, I_{\mathcal{A}}, \delta_M, \gamma_M)$ is a multi-attack graph built over a set $\mathcal{A} = \{A_1, \ldots, A_k\}$ of probabilistic temporal attack graphs, and $D = (H, Q, \phi)$ is a generalized dependency graph. An* attack scenario index *is a 6-tuple $I_G = (G, start_G, end_G, tables_G, s_G, completed_G)$, where:*

- *$start_G$ (resp. end_G) $: V_M \rightarrow 2^{I_{\mathcal{A}}}$ is a function that associates with each node $v \in V_M$, the set of attack graph id's for which v is a start (resp. end) node;*
- *For each $v \in V_M$, $tables_G(v)$ is a set of records of the form $(curr, attackID, ts_0, prob, \Delta damage, prev, next)$, where curr is a reference to an alert, $attackID \in I_{\mathcal{A}}$ is an attack graph id, $ts_0 \in T$ is a timestamp, $prob \in [0,1]$ is the probability of the (partial) occurrence, $\Delta damage \in \mathbb{R}$ is the marginal damage, prev and next are sets of references to records in $tables_G$;*
- *$s_G : H \rightarrow [0,1]$ is the current status of the network;*
- *$completed_G : I_{\mathcal{A}} \rightarrow 2^{\mathcal{P}}$, where $\mathcal{P}$ is the set of references to records in $tables_G$, is a function that associates with each attack identifier $id(A)$ a set of references to records in $tables_G$ corresponding to completed occurrences of A.* □

Note that G, $start_G$, end_G can be computed a-priori, given the set $\mathcal{A}$ of attack graphs and the dependency graph D. All the tables that are part of the index ($tables_G$) will be initially empty. As new alerts are received, they will be updated accordingly, as described in Section 6.1. Index tables allow to track partially completed occurrences, and each record points to an alert, as well as to previous

Algorithm 1. *updateIndex*(o_{new}, I_G, p_t)

```
Input: New alert to be processed $o_{new}$, attack scenario index $I_G$, probability threshold $p_t$.
Output: Updated attack scenario index $I_G$.
1: $v_{new} \leftarrow exploit(o_{new})$ // Map the new alert to a known vulnerability exploit
2: $\Delta damage \leftarrow 0$
3: for all $h \in H$ s.t. $(v_{new}, h) \in F$ do
4:     $\Delta damage \leftarrow \Delta damage + assessDamage(h, \eta(v_{new}, h))$
5: // Look at start nodes
6: if $start_G(v_{new}) \neq \emptyset$ then
7:     for all $id \in start_G(v_{new})$ do
8:         add $(o_{new}^{\uparrow}, id, o_{new}.ts, 1, \Delta damage, \emptyset, \emptyset)$ to $tables_G(v_{new})$
9: // Look at intermediate nodes
10: $V \leftarrow \emptyset$
11: for all node $v \in V_M$ s.t. $\exists id \in I_{\mathcal{A}},\ \delta_M(v, v_{new}, id) \neq$ null do
12:     $r_{first} \leftarrow \min\{r \in tables_G(v) | o_{new}.ts - r.curr.ts \leq \max_{id \in I_{\mathcal{A}} | \delta_M(v, v_{new}, id) \neq null} \delta_M(v, v_{new}, id).t_{max}\}$
13:     for all records $r \in tables_G(v)$ s.t. $r \geq r_{first}$ do
14:         $id \leftarrow r.attackID$
15:         if $\delta_M(v, v_{new}, id) \neq \emptyset$ then
16:             $(I, \tau) \leftarrow \delta_M(v, t_{new}.obs, id)$
17:             $p \leftarrow \tau(x, y)$ where $[x, y] \in I$ and $x \leq t_{new}.ts - r.curr.ts \leq y$
18:             if $p \geq p_t$ then
19:                 $V \leftarrow V \cup \{(v, r, p)\}$
20:                 if $V$ satisfies $\gamma(v_{new})$ then
21:                     $prob \leftarrow 1$
22:                     $records \leftarrow \emptyset$
23:                     for all $(v_i, p_i, r_i) \in V^*$ s.t. $V^*$ is a minimal subset of $V$ satisfying $\gamma(v_{new})$ do
24:                         $prob \leftarrow prob \cdot r_i.prob \cdot p_i$
25:                         $records \leftarrow records \cup \{r_i^{\uparrow}\}$
26:                     if $prob \geq p_t$ then
27:                         $r_n \leftarrow (o_{new}^{\uparrow}, id, \min_{r' \in records} r'.ts_0, prob, \Delta damage, records, \emptyset)$
28:                         add $r_n$ to $tables_G(v_{new})$
29:                         for all $r' \in records$ do
30:                             $r'.next \leftarrow r'.next \cup \{r_n^{\uparrow}\}$
31:                         // Look at end nodes
32:                         if $id \in end_G(v_{new})$ then
33:                             add $r_n^{\uparrow}$ to $completed_G(id)$
```

and successor records. Each record also stores the time at which the partial occurrence began (ts_0), and the marginal damage caused by the corresponding exploit.

6.1 Index Update Algorithm

This section describes an algorithm (Algorithm 1, *updateIndex*) to update the index when a new alert is received. The algorithm takes as input an attack scenario index I_G, a new alert o_{new}, and a probability threshold p_t.

Line 1 maps the newly received alert o_{new} to a known vulnerability exploit v_{new}. Lines 2–4 compute the marginal damage caused by exploit v_{new} by invoking the *assessDamage* algorithm (Algorithm 2) for each network entity $h \in H$ directly affected by v_{new} and summing up all the contributions. Algorithm 2 starts by computing the reduction in the performance of h (Lines 1–5), and then iteratively propagates damage assessment to all the affected nodes (Lines 7–8).

Lines 6–8 of Algorithm 1 handle the case when v_{new} is the start node of an attack graph in $\mathcal{A}$. A new record is added to $start_G(v_{new})$ for each attack graph in $\mathcal{A}$ for which v_{new} is a start node. Lines 10–33 look at the tables associated with the nodes that precede v_{new} in the temporal multi-attack graph

Algorithm 2. *assessDamage*(h, η)

```
Input: Node $h$ in the generalized dependency graph, performance reduction $\eta$ due to an exploit
Output: marginal damage assessment
// Update status and assess direct damage to $h$
$s_{before} \leftarrow s_G(h)$
$s_G(h) \leftarrow \min((1-\eta) \cdot s_G(h), \phi(h)(\dot{}h))$
$s_{after} \leftarrow s_G(h)$
$\Delta damage \leftarrow (s_{before} - s_{after}) \cdot u(h)$
// Assess damage propagation
for all $h_i \in h^{\cdot}$ s.t. $s_G(h_i) > 0$ do
    $\Delta damage \leftarrow \Delta damage + assessDamage(h_i, 0)$
return $\Delta damage$
```

and check whether the new alert can be correlated to existing partially completed occurrences. For each predecessor v of v_{new}, Line 12 determines where the algorithm should start scanning $tables_G(v)$. Note that records are added to index tables as new alerts are received, thus they are ordered by *r.curr.ts*, i.e., the time at which the corresponding alert was received. Given two records r_1, $r_2 \in tables_G(v)$, we use $r_1 \leq r_2$ to denote the fact that r_1 precedes r_2 in $tables_G(v)$, i.e., r_1*.curr.ts* $\leq r_2$*.curr.ts*. To avoid scanning the entire table, only the "most recent" records in $tables_G(v)$ are considered, i.e., those that can still be linked to new alerts (Line 12). On Lines 16-17, timespan distributions are used to determine the probability p that the new alert can be linked to record r for the attack graph identified by *r.attackID*, given the amount of time elapsed between *r.curr.ts* and o_{new}*.ts*. Since we are interested in occurrences with a probability above the threshold p_t, we discard the partial occurrence as soon as $p < p_t$ (Line 18). Otherwise, we keep track in V of all the predecessor exploits that can be linked to v_{new}, and, if the condition $\gamma(v_{new})$ is satisfied (Line 20), we consider a subset V^* of V that minimally satisfies $\gamma(v_{new})$ and add a new record r_n to $tables_G(v)$ (Lines 27-28) if the overall probability of the corresponding (partial) occurrence is above the threshold. Additionally, we update predecessor records to have r_n as their successor (Lines 29-30). Note that r_n inherits ts_0 from its predecessors; this ensures that the starting and ending times can be quickly retrieved by looking directly at the last record for a completed occurrence. Finally, lines 32–33 check whether v_{new} is an end node for some attack graph. If yes, a pointer to r_n is added to $completed_G$, telling that a new occurrence has completed.

Algorithm *updateIndex*, can be used iteratively for loading an entire observation sequence at once (we refer to this variant as *bulkUpdate*). The following result characterizes the time complexity of the two algorithms.

Proposition 1. *Given a set $\mathcal{A}$ of probabilistic temporal attack graphs, a multi-attack graph $M = (V_M, I_{\mathcal{A}}, \delta_M, \gamma_M)$ over $\mathcal{A}$, an attack scenario graph G, and an attack scenario index $I_G = (G, start_G, end_G, tables_G, s_G, completed_G)$, the worst case complexity of algorithm* updateIndex *(resp.* bulkUpdate*) is $O(k^{|V_M|} \cdot |\mathcal{A}|)$ (resp. $O(k^{|V_M|} \cdot |\mathcal{A}| \cdot |O|)$), where O is the observation sequence and k is the level of concurrency, i.e., the maximum number of concurrent attacks.* □

Note that the complexity result above are based on the fact that the number of *recent* records in each table $tables_G(v)$ is independent of the size of O. This can be formally proved, and is largely confirmed by experimental results. Additionally, our experiments show that, in practice, time and space complexity are independent of the size of the activities. This result is expected since $|O| \gg |V_M|$.

7 Analysis of Future Scenarios

As Example 3 has shown, recommendations should be made to the analyst based on a compromise between likelihood and marginal damage of possible future scenarios. Ideally, an automated system should provide the analyst with a list of predicted scenarios, ranked by a measure of criticality accounting for both probability and marginal damage. The analyst would then be recommended to take action to prevent attack steps corresponding to the scenario with the highest criticality. In the simplest case, given a set of predicted occurrences $\{O_1^*, \ldots, O_n^*\}$, we can estimate the criticality of O_i^*, for each $i \in [1, n]$, as $p_i \cdot \Delta damage_i$, where p_i and $\Delta damage_i$ are the probability and marginal damage of O_i^* respectively. In general, a *criticality function* can be defined as any function of the form $f : [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}$ that satisfies the following monotonicity axioms:

$$(\forall \Delta d \in R)\ p_1 \geq p_2 \Rightarrow f(p_1, \Delta d) \geq f(p_2, \Delta d) \tag{3}$$

$$(\forall p \in [0, 1])\ \Delta d_1 \geq \Delta d_2 \Rightarrow f(p, \Delta d_1) \geq f(p, \Delta d_2) \tag{4}$$

Example 4. Consider the two scenarios described in Example 3. According to the simple computation model described above, their criticality is 2.45 and 6.87 respectively, confirming that protecting h_F would be a wiser choice.

We propose algorithm *rankFutureScenarios* (Algorithm 3) to predict and rank all possible future scenarios. We assume that the attacker's goal is to reach a network resource corresponding to the end node of an attack graph. The algorithm takes as input an attack scenario graph I_G, a time point ts, an integer $k > 0$, a probability threshold p_t, and a criticality function f, and returns a ranked list of possible future attack occurrences. The algorithm considers all the record in the index which do not have a successor yet (Lines 2-3). For each such record r, a recursive algorithm (*simGraphForward*) traverses the attack graph forward, and derives a set of partial future occurrences of length k or less along with their probability and marginal damage assessment (Line 5). This algorithm is straightforward and details are omitted for reasons of space. Finally, all the predicted partial occurrences are ranked using f. The following result characterizes the time complexity of algorithms *rankFutureScenarios*.

Proposition 2. *Given an attack scenario index $I_G = (G, start_G, end_G, tables_G,$ s_G, $completed_G)$ and an integer k, the worst case complexity of algorithm* rankFutureScenarios *is $O(|V_M| \cdot d^k)$, where V_M is the set of exploits in the multi-attack graph and d is the maximum out-degree of nodes in V_M.* □

Algorithm 3. $rankFutureScenarios(I_G, ts, k, p_t, f)$

Input: Attack scenario index I_G, time ts, integer $k > 0$, prob. threshold p_t, criticality function f.
Output: List of 4-tuples $(id(A), O, prob, \Delta damage)$ ranked by $f(prob, \Delta damage)$, where $id(A) \in I_{\mathcal{A}}$, O is a partial occurrence, $prob$ is the probability of O, and $\Delta damage$ is the marginal damage

```
$S \leftarrow \emptyset$
for all $v \in V_M$ do
    for all $r \in tables_G(v)$ s.t. $e.next = \emptyset \wedge r.curr.ts < ts \wedge r^{\uparrow} \notin completed_G(r.attackID)$ do
        // Find attack patterns starting at $v$, no longer than $k$, and with probability above $p_t$
        $\mathcal{O} \leftarrow simGraphForward(r.attackID, exploit(r.curr), k, p_t)$
        for all $(O, prob, \Delta damage) \in \mathcal{O}$ do
            $S \leftarrow S \cup \{(r.attackID, O, prob, \Delta damage)\}$
return $rankedList(S, f)$
```

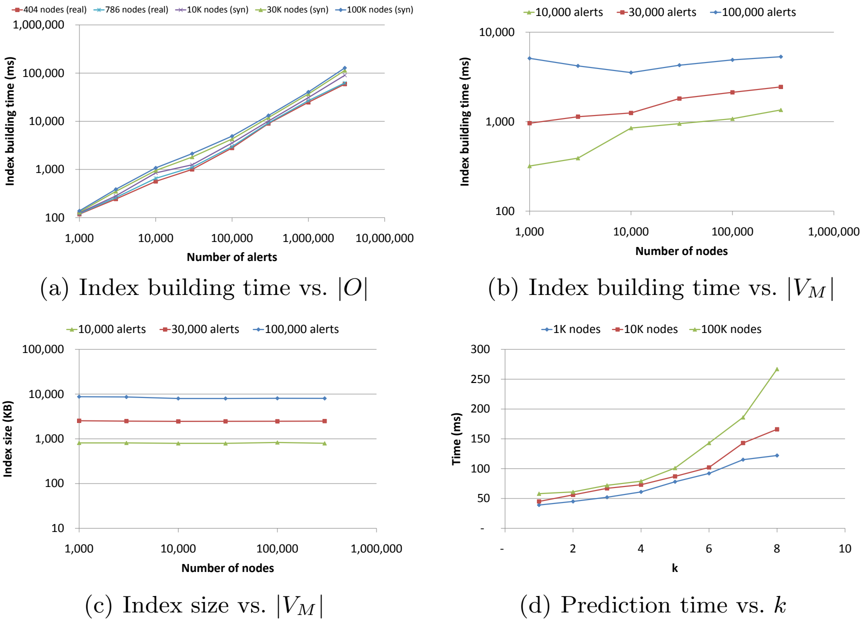
The above result, confirmed by the experiments, indicates that, although time complexity is exponential in the number k of steps forward, the effect of the exponential is not significant. In fact, typically $|V_M| \gg d$, and k is not large.

8 Experiments

In this section, we report the results of the experiments we conducted to evaluate the time and memory performance of the proposed index update algorithm, as well as the performance of the *rankFutureScenario* algorithm. We evaluated the system using both real and synthetic attack graphs. In both cases, we used the attack graphs to simulate a number of attacks and generate a stream of alerts. Specifically, we used an attack graph that was generated by scanning an existing network with the tool described in [11]. The resulting temporal attack graph includes 786 nodes, encompassing 64 machines. In order to test our framework on larger graphs, we generated synthetic graphs of up to 300 thousand nodes.

Figure 3(a) shows how the time to build the entire index using the *bulkUpdate* algorithm – for both synthetic and real attack graphs – increases as the number of alerts increases. It is clear that the index building time is linear in the number of alerts (note that both axes are on a log scale), and the algorithm can process between 25 and 30 thousands alerts per second. Also note that there is no significant difference between results on real and synthetic attack graphs, and that the size of the graphs does not significantly affects the index building time, as confirmed by Figure 3(b): when the size of the merged graph changes by orders of magnitude, the processing time increases slightly, but remains within the same order of magnitude. This can be easily explained by considering that, for a given number of alerts, when the size of the graphs increases, the number of index tables (one for each $v \in V_M$) increases as well, but at the same time the average number of occurrences of each alert and the average size of each table decrease, keeping total memory occupancy (Figure 3(c)) and processing time (Figure 3(b)) almost constant. The slight increase in processing time can then be attributed to the overhead of managing a larger number of tables. Figure 3(c) also indicates that memory occupancy increases linearly with the number of alerts processed.

Finally, execution times of algorithm *rankFutureScenarios* for varying values of k are reported in Figure 3(d). Note that the processing time is not



(a) Index building time vs. $|O|$

(b) Index building time vs. $|V_M|$

(c) Index size vs. $|V_M|$

(d) Prediction time vs. k

Fig. 3. Experimental results

significantly affected by the size of the graphs. This can be easily explained with the same argument used above for index building time and memory. Algorithm *rankFutureScenarios* looks at the most recent records in each index table, i.e., those in a given temporal window. As the size of the graphs increases, the number of records per table falling in such temporal window decreases, thus keeping the total number of records to examine almost constant. For any give graph size, when k increases, the processing time increases. However, we can note that, for smaller graphs, the processing time become stable as k increases, whereas, for larger graphs, it will continue to increase exponentially. This can be easily explained considering that, as k increases, it will be less likely for smaller graph to have partial patterns of length k, so in most cases the algorithm will end the recursive search earlier than k steps forward.

9 Conclusions

In this paper, we proposed a novel framework to integrate vulnerability and dependency analysis, and provide security analysts with a better picture of the cyber situation. Our contribution was threefold. First, we introduced the notion of generalized dependency graph, which captures how network components, at different levels of abstraction, depend on each other. Second, we extended the classical definition of attack graph to incorporate probabilistic knowledge of the attacker's behavior. Finally, we introduced the notion of attack scenario graph, which integrates dependency and attack graphs. We proposed efficient algorithms for detection and prediction, and showed that our framework can handle very large attack graphs and large volumes of alerts. Further research will be needed to fully automate the generation of attack scenario graphs.

References

1. Albanese, M., Chellappa, R., Moscato, V., Picariello, A., Subrahmanian, V.S., Turaga, P., Udrea, O.: A constrained probabilistic petri net framework for human activity detection in video. IEEE Transactions on Multimedia 10(8), 1429–1443 (2008)
2. Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D.A., Zhang, M.: Towards highly reliable enterprise network services via inference of multi-level dependencies. ACM SIGCOMM Computer Communication Review 37, 13–24 (2007)
3. Bahl, P.V., Barham, P., Black, R., Chandra, R., Goldszmidt, M., Isaacs, R., Kandula, S., Li, L., MacCormick, J., Maltz, D., Mortier, R., Wawrzoniak, M., Zhang, M.: Discovering Dependencies for Network Management. In: Proceedings of the 5th ACM Workshop on Hot Topics in Networking (HotNets) (November 2006)
4. Chen, X., Zhang, M., Mao, Z.M., Bahl, P.: Automating network application dependency discovery: experiences, limitations, and new solutions. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI 2008, pp. 117–130. USENIX Association, Berkeley (2008)
5. Dain, O., Cunningham, R.K.: Fusing a heterogeneous alert stream into scenarios. In: Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications, pp. 1–13 (2001)
6. Duong, T., Bui, H., Phung, D., Venkatesh, S.: Activity Recognition and Abnormality Detection with the Switching Hidden Semi-Markov Model. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005), vol. 1, pp. 838–845 (2005)
7. Golab, L., Özsu, M.T.: Issues in data stream management. SIGMOD Record 32, 5–14 (2003)
8. Habra, N., Charlier, B., Mounji, A., Mathieu, I.: Asax: Software architecture and rule-based language for universal audit trail analysis. In: Deswarte, Y., Eizenberg, G., Quisquater, J.-J. (eds.) ESORICS 1992. LNCS, vol. 648, pp. 435–450. Springer, Heidelberg (1992)
9. Hamid, R., Huang, Y., Essa, I.: ARGMode Activity Recognition Using Graphical Models. In: Proceedings of the IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR 2003), vol. 3, pp. 38–43 (2003)
10. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: Proceedings of 22nd Annual Computer Security Applications Conference (ACSAC 2006), pp. 121–130. IEEE Computer Society, Los Alamitos (2006)
11. Jajodia, S., Noel, S.: Topological Vulnerability Analysis. In: Cyber Situational Awareness: Issues and Research. Advances in Information Security, vol. 46, pp. 139–154. Springer, Heidelberg (2009)
12. Kandula, S., Chandra, R., Katabi, D.: What's going on?: learning communication rules in edge networks. ACM SIGCOMM Computer Communication Review 38, 87–98 (2008)

13. Kheir, N., Cuppens-Boulahia, N., Cuppens, F., Debar, H.: A service dependency model for cost-sensitive intrusion response. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 626–642. Springer, Heidelberg (2010)
14. Lamport, L.: Distributed system (May 1987), `http://research.microsoft.com/enus/um/people/lamport/pubs/distributed-system.txt`
15. Leversage, D.J., Byres, E.J.: Estimating a system's mean time-to-compromise. IEEE Security and Privacy 6, 52–60 (2008)
16. Mörchen, F.: Unsupervised pattern mining from symbolic temporal data. SIGKDD Explorations Newsletter 9(1), 41–55 (2007)
17. Ning, P., Xu, D.: Learning attack strategies from intrusion alerts. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003), pp. 200–209 (2003)
18. Ning, P., Xu, D., Healey, C.G., Amant, R.S.: Building attack scenarios through integration of complementary alert correlation methods. In: Proceedings of the 11th Annual Network and Distributed System Symposium (NDSS 2004), pp. 97–111 (2004)
19. Noel, S., Robertson, E., Jajodia, S.: Correlating intrusion events and building attack scenarios through attack graph distances. In: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004), pp. 350–359 (2004)
20. Qin, X., Lee, W.: Statistical causality analysis of INFOSEC alert data. In: Vigna, G., Krügel, C., Jonsson, E. (eds.) RAID 2003. LNCS, vol. 2820, pp. 73–93. Springer, Heidelberg (2003)
21. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Lee, W., Mé, L., Wespi, A. (eds.) RAID 2001. LNCS, vol. 2212, pp. 54–68. Springer, Heidelberg (2001)
22. Wang, L., Liu, A., Jajodia, S.: Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. Computer Communications 29(15), 2917–2933 (2006)

Usability of Display-Equipped RFID Tags for Security Purposes

Alfred Kobsa[1], Rishab Nithyanand[2], Gene Tsudik[1], and Ersin Uzun[3]

[1] University of California, Irvine, CA, USA
{kobsa,gtsudik}@uci.edu
[2] Stony Brook University, NY, USA
rnithyanand@cs.stonybrook.edu
[3] Palo Alto Research Center, CA, USA
ersin.uzun@parc.com

Abstract. The recent emergence of RFID tags capable of performing public key operations has enabled a number of new applications in commerce (e.g., RFID-enabled credit cards) and security (e.g., ePassports and access-control badges). While the use of public key cryptography in RFID tags mitigates many difficult security issues, certain important usability-related issues remain, particularly when RFID tags are used for financial transactions or for bearer identification.

In this paper, we focus exclusively on *techniques with user involvement* for secure user-to-tag authentication, transaction verification, reader expiration and revocation checking, as well as association of RFID tags with other personal devices. Our approach is based on two factors: (1) recent advances in hardware and manufacturing have made it possible to mass-produce inexpensive passive display-equipped RFID tags, and (2) high-end RFID tags used in financial transactions or identification are usually attended by a human user (namely the owner). Our techniques rely on user involvement coupled with on-tag displays to achieve better security and privacy. Since user acceptance is a crucial factor in this context, we thoroughly evaluate the usability of all considered methods through comprehensive user studies and report on our findings.

1 Introduction

Radio Frequency Identification (RFID) technology was initially envisaged as a replacement for barcodes in supply chain and inventory management. A small device with no power source of its own (called RFID tag) could be read from some distance away by a special device (called RFID reader), without line-of-sight alignment as is needed for barcodes. However, its many advantages have greatly broadened the scope of possible applications today. Current and emerging applications range from visible and personal tags (e.g., toll transponders, passports, credit cards, access badges, livestock/pet tracking devices) to stealthy tags in merchandize (e.g., clothes, pharmaceuticals and books/periodicals). The costs and capabilities of RFID tags vary widely depending on the target application. At the high end of the spectrum are the tags used in e-Passports, electronic ID (e-ID) Cards, e-Licenses, and contactless payment instruments. Such applications involve relatively sophisticated tags that only cost a few dollars (usually<10).

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 434–451, 2011.

Even though they are powerful enough to perform sophisticated public key cryptographic operations, security and privacy issues remain when these tags are used as a means of payment or for owner/bearer identification. In this paper, we address four such issues:

User-to-Tag Authentication: In many applications of RFID in electronic payment and in identification documents, authentication of the user to the tag before disclosing any information is necessary to prevent leaks of valuable or private information. Current systems require trust in readers for the purpose of authentication. For example, users must enter PINs into ATMs or Point-of-Sale (POS) terminals to authenticate themselves to the RFID tag embedded into their ATM or credit card. However, this leaves users vulnerable to attacks, since secret PINs are being disclosed to third party readers that are easy to hack and modify.

Transaction Verification: RFID tags are commonly used as payment and transaction instruments (e.g., in credit, debit, ATM and voting cards). In such settings, a malicious reader can easily mislead the tag into signing or authorizing a transaction different from the one that is communicated to, or intended by, the user. This is possible because there is no direct channel from a tag to its user on regular RFID tags (i.e., no secure user interface), and the only information a user gets (e.g., a receipt, or an amount displayed on the cash register) is under the control of a potentially malicious reader. Thus, it seems impossible for a user to verify (in real time) transaction details, e.g., the amount or the currency. This problem becomes especially important with current electronic credit cards.

Reader Revocation and Expiration: Any certificate-based Public Key Infrastructure (PKI) needs an effective expiration and revocation mechanism. In RFID systems, it intuitively concerns two entities, namely RFID tags and RFID readers. The former only becomes relevant if each tag has a "public key identity," and we claim that revocation of RFID tags is a non-issue since, once a tag identifies itself to a reader, the reader can use any current method for revocation status verification. In contrast, expiration and revocation of *reader* certificates constitutes a challenging problem in any public key-enabled RFID system. This is because RFID tags, being powerless passive devices, cannot maintain a clock. In other words, an RFID tag (on its own) has no means to verify whether a given certificate has expired or whether any revocation information is recent.

Secure Pairing of RFID Tags: Current high-end RFID tags cannot establish a secure ad-hoc communication channel to another device, unless the latter is part of the same RFID infrastructure (i.e., an authorized reader). Establishing such a channel seems important as it would give tag owners the ability to manage their tags. Previously proposed secure device pairing solutions require an auxiliary communication channel to authenticate devices and establish a secure communication channel [21], [20]. Until recently, however, RFID tags lacked user interfaces and thus could not be paired with other devices. Novel display-equipped RFID tags open a new chapter in RFID security and give users more control over their tags. Using an NFC-capable personal device (such as a smart-phone), for instance, a user can change settings on a personal RFID tag.

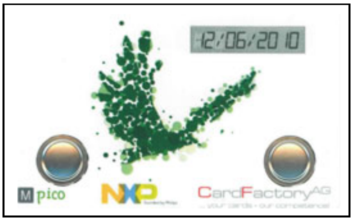


Fig. 1. NXP Display-Equipped RFID Tag (DERT) with two buttons

The gist of our approach is to take advantage of recently developed technology that allows high-end RFID tags to be equipped with a small passive display (see Figure 1 for a tag manufactured by NXP Semiconductors). We refer to such tags as **D**isplay-**E**quipped **R**FID **T**ags or DERTs. The only other publicly known application of DERTs are eID cards in Germany since November 2010 [3]. As we will show in the remainder of this paper, carefully designed user interaction with personal DERTs can yield solutions to the aforementioned problems. We present several simple techniques that require little or no change to already well-established RFID back-end infrastructures (e.g., the back-end processing systems of ePassports, payment instruments, etc.). Thereafter we conduct a thorough study to assess the usability of these techniques.

One of the key motivating factors for our work is the fact that DERTs are already being produced and are available on the market. Moreover, they cost only a few dollars (or euros) more than their display-less counterparts. We note that our work and usability studies are also to a small degree relevant to cards with displays and buttons that require physical contact with readers.

The rest of this paper is organized as follows: we summarize related work in Section 2, describe our technical approach in Section 3, present a comprehensive usability evaluation of the proposed techniques in Section 4, and conclude with a summary in Section 5.

2 Related Work

2.1 Secure User-to-Tag Authentication

User authentication is a fundamental problem that has received a great deal of attention in the security community, for several decades. Solutions range from simple modifications of the standard PIN/password entry techniques [33,14] to schemes that pose more complicated cognitive tasks to users [31,15].

The authentication of users to passive devices (such as RFID tags) is a very recent issue. In the first proposed solution by Czeckis *et al.* [13], users authenticate to an accelerometer-equipped RFID tag by moving or shaking it (or the wallet containing it) in a certain pattern. However, this method assumes that RFID tags are equipped with an accelerometer, and it requires users to memorize movement patterns. Also, it is prone to passive observer attacks. A similar technique called "PIN-Vibra" was suggested by Saxena *et al.* [30] for authenticating to an accelerometer-equipped RFID tag using a mobile phone. In it, a vibrating mobile phone is used to lock or unlock RFID tags. While the usability of PIN-Vibra seems promising, it has a some drawbacks: (1) high error rates – accelerometers on tags can not perfectly decode PINs encoded in phone vibrations, (2) the user's phone must be present and functional (e.g., not out of battery) whenever the tag has to be used, and (3) accelerometer-equipped RFID tags are relatively expensive and do not lend themselves well to other applications that would help amortize their cost.

The secure user-to-tag authentication solution described and tested in this paper is most similar to Abadi *et al.*'s [7] proposal for authentication on smartcards, where a displayed random number is modified by a user to match a PIN.

2.2 Transaction Verification

Current systems that address transaction verification and amount fraud utilize data mining (e.g., [12]), machine learning techniques (e.g., [8]), and out-of-band communication. Most banks verify transactions via alternate communication mediums such as email or telephone. A complete survey of modern fraud detection techniques for Card Present (a.k.a, off-line) and Card not Present (a.k.a, on-line) transactions is given by Kou *et al.* in [22]. In this paper, we present a simple solution that permits user-aided verification using DERTs and fully mitigates amount and currency fraud for Card Present transactions. To the best of our knowledge, this is the first work that offers a real solution and provides a comprehensive analysis of its usability.

2.3 Reader Revocation Checking

Three popular methods to verify the status of a public key certificate (PKC) are: Certificate Revocation Lists (CRLs) [18], Online Certificate Status Protocol (OCSP) [26] and Certificate Revocation System (CRS) [25,24]. CRLs are signed lists of revoked certificates periodically published by certification or revocation authorities (CAs or RAs). The usage of CRLs is problematic in RFID systems since they require the tag to have a clock in order to determine whether a given CRL is sufficiently recent, and since the communication overhead can be quite high if the number of revoked entities is large. OCSP is an online revocation checking method that reduces storage requirements for all parties involved, while providing timely revocation status information. Although well suited for large connected networks, it is a poor fit for RFID systems as it requires constant connectivity between readers and OCSP responders. Furthermore, the need for a two-round challenge-response protocol with OCSP responders may make it susceptible to network congestion and slow turnaround times. CRS offers implicit, efficient and compact proofs of certificate revocation. However, it is unworkable in the RFID context as it also requires verifiers (RFID tags) to have a clock.

Despite much prior work in RFID security and certificate revocation, coupled with the fact that the problem had been spotted by researchers [17,19,16], little has been done to address reader PKC revocation and expiration checking problems. Only very recently, Nithyanand *et al.* [28] proposed a method that entails user involvement and DERTs to determine PKC validity. We adopt and experiment with this solution. Although [28] includes a preliminary usability study using a mocked-up implementation on mobile phones, this paper presents a comprehensive analysis of the usability of the method tested using actual DERTs and realistic user tasks.

2.4 Secure Device Pairing

A number of device association/pairing methods have been proposed over the past few years. They use various out-of-band (OOB) channels in the process of establishing a secure connection, and as a result, exhibit different usability characteristics. Recent work in [21,20] and [23] surveys many pairing methods and reports on their usability. However, because of the nature of (very) basic displays that can be integrated into RFID tags, only visual text-based methods are appropriate for DERTs.

In this paper, we adopt the "Copy" method that was introduced by Uzun *et al.* [32], and evaluate its usability in the DERT setting. In the *copy* pairing technique, one device displays a randomly generated passkey, which the user types into the second device. The devices automatically run a password based authenticated key agreement protocol (e.g., [10]), which succeeds or fails depending on the user's ability to copy the passkey correctly between the devices and the presence of an active attack on the communication channel (e.g., man-in-the-middle or denial of service attacks).

3 Proposed Techniques

3.1 General Assumptions

All methods described below share the following general assumptions:

1. Tags are owned and operated by individuals (users/owners) who understand their roles in each context (users only need to know the actions they are required to perform, but not the reasons for performing them).
2. Tags are powerful enough to perform public key operations (at least signature verification). This is true for all our target applications.
3. Tags are equipped with an one-line alpha-numeric display (OLED or ePaper) capable of showing at least 8 characters. This is made possible by current DERT technology.
4. Tags can maintain simple counters or timers *while* powered by a reader.
5. Each tag has a programmable button.[1]

3.2 User-to-Tag Authentication

The authentication method described in Figure 2 is designed for DERTs but can be used on any wireless, interface-constrained device.

We make three additional assumptions:

1. Tags are capable of generating short random numbers (i.e., 4-6 decimal digits).
2. Users have access to a possibly *untrusted* keypad (or keyboard) with cursor keys. The keypad can be part of the reader, or be connected to it.
3. Tags always clear and reset their displays after authentication. Note that this is possible even in the case of malicious readers due to the presence of residual charges in a DERT.

[1] We used NXP tags with two buttons in our usability tests. One of the button actions can always be substituted with a timeout though.

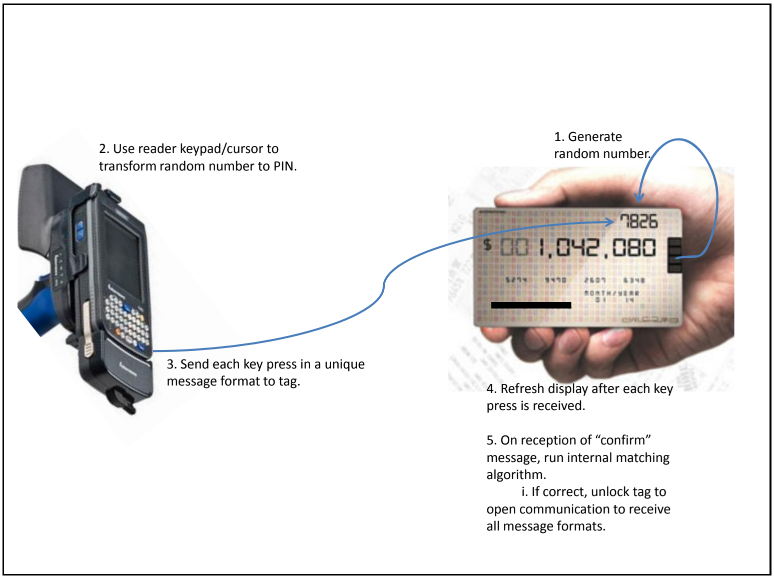

2. Use reader keypad/cursor to transform random number to PIN.
1. Generate random number.
7826
$ 00 1,042,080
3. Send each key press in a unique message format to tag.
4. Refresh display after each key press is received.
5. On reception of "confirm" message, run internal matching algorithm.
i. If correct, unlock tag to open communication to receive all message formats.

Fig. 2. Secure user-to-DERT authentication

The Protocol. In order to unlock a tag for a transaction (e.g., a credit card at a store, a cash card at an ATM, or an e-passport at a hotel), the user needs to be authenticated by proving knowledge of a secret, such as a PIN. The following method, which is a variant of the method proposed in [7] for battery powered smart-cards, allows user-to-tag authentication without requiring any buttons/keys on the tag. Moreover, the PIN is protected from potentially malicious (and certainly untrusted) readers.

1. Powered by the reader, DERT generates a one-time random number of the same length as the PIN. DERT proceeds to display this random number. Note that this $nonce$ is not known by the reader that powers the DERT.
2. User operates the cursor keys (↑, ↓, ←, →) on the reader keypad to basically *adjust* this random number on the DERT to his/her PIN. This is done digit by digit. For example, if the random number displayed by DERT is "5723" and the user's PIN is "296", the necessary sequence of key presses is: 1) 4 times ↓, →, 2) 5 times ↑, →, 3) 3 times ↓, →, 4) 3 times ↑, followed by *Confirm*. For each user key-press, the reader sends a corresponding message to the tag detailing the key-press, thereby prompting the tag to update its display.
3. Upon receipt of the *Confirm* message, DERT unlocks itself for a transaction if the PIN was entered correctly.

Since the reader is unaware of the nonce initially generated by the DERT, it is impossible (even with knowledge of the sequence of keys pressed by the user) to reconstruct the PIN used to unlock the DERT. Note that this method's security is based on several factors. The first is our assumption about the DERT's ability to generate cryptographically secure random numbers. The second security requirement is that the user *must* *alternate* ↑ and ↓ movements between digits. In other words, if only the ↓ key is used for small PIN digits (i.e., < 5) instead of sometimes going past "9" to reach it, or vice versa for large digits, then such a pattern may leak information about the PIN if the protocol is executed repeatedly with the same reader. If there is a concern about such leaks, they can be easily prevented by allowing only one of the ↑ or ↓ keys to be used when modifying the digits.

Shoulder-Surfing Resistant Variant: In a shoulder-surfing attack, an adversary somehow observes the user's actions to obtain critical information (e.g., the PIN entered into an ATM). Such attacks range from simply looking over the victim's shoulder to using a camera to observe him or her. They are simple to launch and effective in public areas where large crowds or long queues are likely to occur. By masking all digits except the one being modified, it is easy to make the above protocol shoulder-surfing resistant (It does not become *shoulder-surfing proof*, however).

We tested both flavors of this protocol and used '$\backslash$' as the masking character. Although '$*$' is more commonly used for this purpose, the prototype firmware on our test tags was not yet capable of displaying it.

3.3 Transaction Verification

Our approach to transaction amount verification is designed to work with any RFID-enabled payment instrument. Its primary goal is to provide simple, secure and usable transaction verification at a Point-of-Sale (PoS). The following additional assumption is necessary:

- The user has access to either a printed or a digital (e.g., displayed on the cash register) receipt for the transactions to be verified.

The Protocol (also see Figure 3)

1. DERT receives transaction details from the reader (seller/merchant).
2. DERT verifies that the details (e.g., issuing bank, account number, etc.) match their counterparts in the reader PKC. Protocol is aborted in case of a mismatch.
3. DERT extracts and displays user-verifiable data, i.e, the amount and optionally the currency code. It then enters a countdown stage that lasts for a predetermined period of time (e.g., 10 seconds).
4. User observes transaction information and, if the transaction amount and other details are deemed correct, presses the *Confirm* button on DERT before the timer runs out. At this point, DERT signs the time-stamped transaction statement and sends it to the reader. This signed statement is then sent to the payment gateway and eventually to the financial institution that issued the payment DERT.

   However, if the user decides that transaction details are incorrect, the timer runs out (or the user presses the reject button, if one is available) and DERT automatically aborts the protocol.

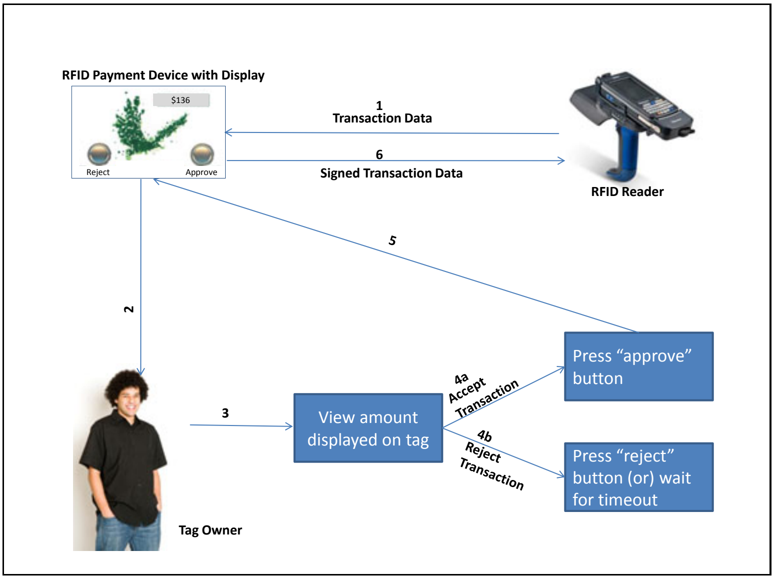


Fig. 3. DERT-enabled transaction verification

3.4 Reader Revocation Status Checking

Our approach for reader certificate expiration and revocation checking [28] is aimed at personal RFID tags – such as ePassports, e-licences or credit/debit cards – when used in places where trust is not implicit. For example, trust in readers might be implicit in international airports (immigration halls) or at official border crossings. Whereas, it is not implicit in many other locations, such as car rental agencies, hotels, flea markets or duty-free stores.

This approach entails the following additional assumptions:

- Tags are aware of the identity and public key of the system-wide trusted Certificate Authority (CA). In other words, all tags and readers are subsumed by a system-wide Public Key Infrastructure (PKI). An example of such a CA is the ICAO CVCA [2].
- The CA is assumed to be infallible: anything signed by the CA is guaranteed to be genuine and error-free.
- The CA periodically (at fixed intervals) issues an updated revocation structure, such as a CRL.
- All tags are aware of the periodicity of issuance of the revocation information and thus can determine expiration time of the revocation structure by simply consulting its issuance time-stamp.
- A tag can retain (in local non-volatile storage) the last valid time-stamp it has encountered.

Note that our usage of the term "time-stamp" is not restricted to time, i.e., hours and minutes. It is meant to express (at appropriate granularity) issuance and expiration of both certificates (PKCs) and revocation information.

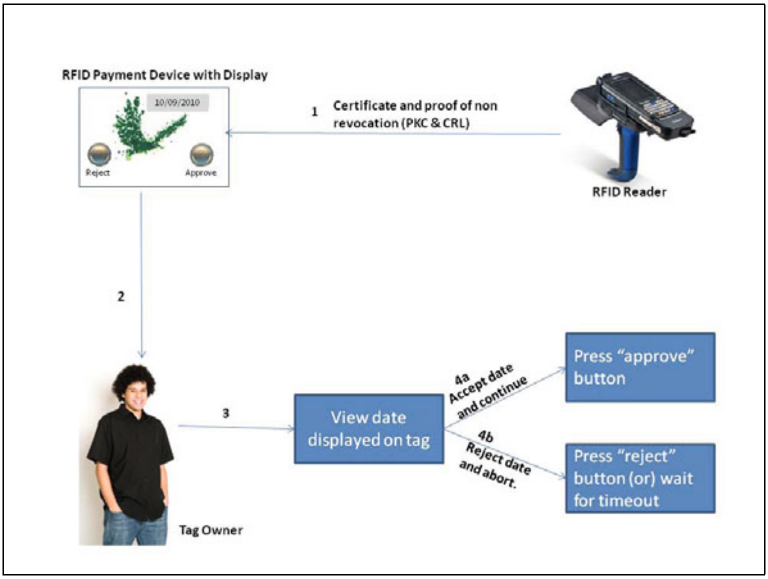

RFID Payment Device with Display
10/09/2010
Reject
Approve
1
Certificate and proof of non revocation (PKC & CRL)
RFID Reader
2
3
Tag Owner
View date displayed on tag
4a Accept date and continue
4b Reject date and abort.
Press "approve" button
Press "reject" button (or) wait for timeout

Fig. 4. Reader certificate expiration/revocation checking

The Protocol. Before providing any information to the reader, a tag has to validate the reader's certificate (PKC). The verification process is as follows (also illustrated in Figure 4):

1. Freshly powered-up DERT receives the Certificate Revocation List (CRL) and the reader's Public Key Certificate (PKC). Let CRL_{iss}, CRL_{exp}, PKC_{iss} and PKC_{exp} denote issuance and expiration times of CRL and PKC, respectively. The last encountered valid time-stamp kept by DERT is denoted as Tag_{Curr}.
2. If either CRL_{exp} or PKC_{exp} is smaller than Tag_{curr}, or $CRL_{iss} \geq PKC_{exp}$, DERT aborts.
3. DERT checks whether CRL includes the serial number of the reader certificate. If so, it aborts.
4. DERT checks the CA signatures of PKC and CRL. If either check fails, DERT aborts.
5. If CRL_{iss} or PKC_{iss} is more recent than the currently stored date, DERT updates it to the more recent of the two.
6. DERT displays the lesser of: CRL_{exp} and PKC_{exp}. It then enters a countdown stage of fixed duration (e.g., 10 seconds).
7. The user decides whether the displayed time-stamp is in the future. If so, the user presses the DERT button before the timer runs out, and communication with the reader continues. Otherwise, the user does nothing: the timer runs out and DERT automatically aborts the protocol.

NOTE: we use the term CRL above to denote a generic revocation structure.

3.5 Secure Device Pairing

Our protocol for bootstrapping a secure communication channel between DERTs and more powerful computing devices such as laptops or cell-phones (i.e., pairing) is based on the "Copy" pairing technique introduced in [32] and described in Section 2.

Additional Assumptions. This technique entails the following additional assumption:

- DERT can generate short random passcodes for the purpose of device pairing and can run secret based key agreement protocols, such as [10].

The protocol. The method operates as follows.

1. DERT generates and displays a sufficiently long decimal passcode (e.g., 6-9 digits).
2. The software interface on the other device prompts the user to enter this passcode.
3. Using the (presumably common) passcode, DERT and the second device run an authenticated key agreement protocol to establish a (stronger) common key and confirm its possession by both parties.

4 Usability Analysis

Since all proposed methods require varying degrees of user involvement, it is very important to assess their usability in order to gauge their eventual user acceptance in real-world deployment. To this end, we conducted a comprehensive usability study with prototype implementations. The goal of the study was to provide answers to the following concrete questions:

1. How do users rate the usability of proposed methods in each problem context?
2. Are users able to perform the required tasks with sufficiently low error rates?
3. Are users willing to perform these tasks on a regular basis?

4.1 Apparatus, Implementation and Setup

Our study was conducted using display-equipped RFID tags (DERTs) from NXP Semiconductors and an HID Omnikey 5321 desktop reader [4]. DERTs were equipped with an integrated 10-position alpha-numeric (ePaper) display unit and two buttons. All code was written in Java 2 Platform Standard Edition with the Java Smart Card I/O API [5].

All tests were conducted in a designated conference room at a university campus. Participants were introduced to the concept of personal RFID tags, with RFID-enabled credit cards and ePassports serving as our main motivating examples. A short presentation using the same set of slides (to ensure consistency) was made to each subject, explaining each usage scenario and subjects' task as potential users in each protocol. These tasks were re-explained before each protocol was tested. Participants were informed of the importance of maintaining natural behavior during the study and were requested not to ask questions during the testing process. However, they were allowed to talk to the test administrator before and after each protocol was tested. Participants were then presented with the DERTs used in the tests in order to familiarize them with the “hardware”. After completing a background questionnaire to collect demographic data, tests were conducted for each protocol described in Section 4.3, and task performance times and error rates were measured.

After testing each protocol, every participant completed a post-test survey. It included the System Usability Scale (SUS) questionnaire [11], a widely used and highly reliable 10-item 5-point Likert scale, and several other questions framed to gain insights into the potential acceptance of the proposed methods.

On average, each person took about 30 minutes to finish the entire series of tests. Everyone was allowed to take part in the study only once. Each participant was rewarded with either an open movie coupon or a $10 Starbucks gift card.

4.2 Subject Background

Our study was conducted over a period of 25 days, in two phases. It involved a total of 35 participants who were chosen on a first-come first-serve basis from the respondents to recruitment emails and flyers. The first 5 respondents were assigned to the pilot test (phase 1) subject pool. Data obtained from this pilot phase was used to make important decisions regarding the need for additional test cases in each protocol. Phase 1 was also important to verify the stability and the limits of our RFID hardware setup. Due to several changes made after the pilot tests in phase 1, data obtained in this phase was not comparable to the data gathered from the remaining 30 participants. Consequently, phase 1 data is not reflected in the results discussed in this paper.

Of the 30 subjects who took part in phase 2, 30% (9 subjects) were aged 18 to 24, 36.67% (11 subjects) 25 to 30, and 33.33% (10 subjects) 30 and over. Gender distribution was nearly even with 53.33% (16 subjects) males and 46.67% (14 subjects) females. The subject pool was extremely well-educated, with 86.67% (26 subjects) having a bachelors degree or higher. We attribute this to the specifics of the study venue, a university campus. 6.67% (2 subjects) reported a disability that impaired their visual perception.

4.3 Test Procedures and Results

User Authentication Variants. In tests of user-tag authentication, each subject was presented with an Automated Teller Machine (ATM) simulator and was asked to authenticate as the tag owner. While our protocol can be used to lock and unlock tags for any purpose, the ATM environment was used to aid the understanding of potential use cases.

After being informed about his/her role in the protocol, each subject was presented with a Logitech N305 wireless number pad [6] that had four highlighted cursor keys to aid in digit manipulation. Next, a subject was asked to complete four test cases (two for each variant). For all test cases, the same four digit PIN was used for the same subject. Furthermore, the initial random number generated by the tag always required a minimum of 13 key presses total for successful authentication. This was done in order to compare completion times between subjects more accurately. In this section, we present our results and attempt to provide insight into which protocol is better suited for the real world.

- **Completion Time and Error Rates:** Each variant had 60 test cases, and the average time to completion for both variants was well under a minute. The study yielded an average completion time of 38.469 seconds for the regular authentication protocol (UA), and 39.684 seconds for the shoulder-surfing resistant variant (UA-SSR). A paired t-test showed that this difference is not statistically significant. Unfortunately, looking at error rates does not give us better insight either: the study yielded low error rates of 6.67% and 3.33% for the UA and UA-SSR protocols, respectively.
- **SUS Scores and Usability Analysis:** The UA protocol was rated at 68.58 out of 100 on the SUS scale, while the UA-SSR protocol received a higher score of 72.58. The possible reasons for this are noted in the following discussion section.

  When asked if they would like to see the protocols implemented in the real world for the purpose of user authentication, 50% (15 subjects) indicated that they would like to see an implementation of UA, while 36.67% (11 subjects) were neutral). When asked the same question about UA-SSR, 60% (18 subjects) agreed that they would like to see it implemented, while 23.33% (7 subjects) were neutral. Finally, when asked if they preferred using UA-SSR over UA, 50% (15 subjects) picked UA-SSR while 20% (6 subjects) did not have a preference. The question received a score of 2.89 on the five point Likert scale.
- **Discussion:** An analysis of the completion times and error rates does not yield a clear winner between the UA and UA-SSR protocols. However, the SUS scores and user opinions indicate that UA-SSR is the preferred protocol for users. Post-test subject interviews lead us to believe that the UA-SSR was preferred because of the presence of the '*cursor*' that indicated which digit was currently being manipulated (recall, all digits which were not being manipulated were replaced by a '\'). This, however, was not present in the UA protocol, and as a result, subjects often lost track of which digit they were manipulating, causing some of them to become frustrated during the authentication process.

  Several subjects indicated concern with the usability of our protocols for visually challenged individuals. Current authentication and PIN-entry techniques allow individuals with visual impairments to perform their roles with reasonable ease through the use of Braille. In contrast, our protocols do not seem to be easily accessible for this user group, and may require special hardware such as personal radio frequency headphones. This is an important concern that we hope to address in future work.

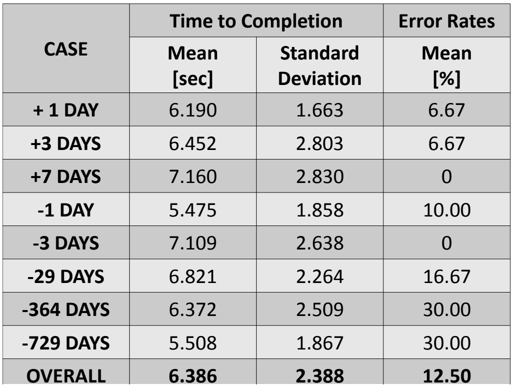
  We point out that while other solutions to the user-to-tag authentication problem such as [30] take significantly less time to complete (mean: 7.122 seconds), the error rates are prohibitively high at 78.75%.

Transaction Verification. While the transaction verification method can be used with any RFID payment/transaction instrument, we focused on the common case of RFID-enabled credit cards in a Point-of-Sale (PoS) environment. This was done not only to help subjects understand use cases more clearly, but also because we envision this case as the primary application domain for this protocol.

- **Test procedure:** After an explanation of their tasks and roles, each subject was presented with a vending machine simulator (with structure and products similar to the Best Buy airport vending machines [1]). Then, each subject was asked to make two separate sets of purchases (each set was a test case). On pressing the *checkout* button on the machine, a digital receipt appeared on the display monitor of the vending machine. Next, the total amount the machine intended to charge was displayed by the tag. Each subject was asked to check whether the two amounts matched. If they matched, the vending machine was deemed "honest". Otherwise, an amount mismatch indicated a malicious vendor attempting to overcharge the user. For each participant, one of the (randomly selected) test cases involved a malicious vending machine that attempted to over-charge by \$1, \$10 or \$100 (the amount was selected at random).
- **Completion Time and Error Rates:** For the 60 $(= 30 * 2)$ test cases, the study yielded an average completion time of 6.6 seconds, with a standard deviation of 3.0 seconds. Furthermore, all 30 subjects completed their tasks successfully and no errors were recorded in the process.
- **SUS Scores and User Opinion:** Subjects rated usability at 86 out of 100 on the System Usability Scale (SUS) [11]. This is far above the "industry average" of 70.1 reported in [9], and indicates excellent usability and acceptability. Also, a staggering 96.67% (29 subjects) stated that they would like to see the system implemented on their own personal tags. Only 1 subject opposed this idea. The average score on a 5-point Likert scale was 4.57, with a standard deviation of 0.64.
- **Discussion:** As the results indicate, our method is unlikely to cause errors. However, we note that this is possibly a consequence of our specific implementation. We anticipate that user errors are likely to arise quite often in real-world deployments if malicious vendors manipulate the placement of decimal points on the DERT (e.g., displaying \$344.1 instead of \$34.41). We were unable to test this attack in our study since the specific NXP prototype tags that we used are incapable of displaying decimal points. This fact in return prompts us to recommend an implementation such as ours when applicable, since it does not display the fractional part of a number (i.e., cents), thereby making it resistant to such attacks. Such an implementation would not be suitable though if micro-payments (less than a dollar) or attacks at the level of decimal fractions are expected.

Reader Revocation Status Checking. To help subjects understand the concept of personal RFID tags and the reader certificate expiration/revocation problem, the ePassport example was used throughout this test. Care was taken to prevent subjects from checking clocks, watches or cell phones for the current date, in order to upper-bound the error rate. After being informed of their role in the protocol, each subject was presented with our implementation and asked to execute the protocol eight times. Finally, opinions were solicited via the post-test questionnaire.

- **Test procedure:** Each subject was presented with eight test cases in a random order. These corresponded to DERT-displayed dates of: +/-1 day, +/-3 days, +7 days, -29 days, -364 days and -729 days from the actual test date ("+" and "-" indicate future and past dates, respectively). The choices of -29 days, -364 days and -729 days



CASE	Time to Completion		Error Rates
	Mean [sec]	Standard Deviation	Mean [%]
+ 1 DAY	6.190	1.663	6.67
+3 DAYS	6.452	2.803	6.67
+7 DAYS	7.160	2.830	0
-1 DAY	5.475	1.858	10.00
-3 DAYS	7.109	2.638	0
-29 DAYS	6.821	2.264	16.67
-364 DAYS	6.372	2.509	30.00
-729 DAYS	5.508	1.867	30.00
OVERALL	**6.386**	**2.388**	**12.50**

Fig. 5. Completion times and error rates for various test cases

were deliberate so as to make their "staleness" more obscure to the subjects. After a date was displayed on the DERT, each subject was asked to decide to: (1) accept the date by pressing the *OK* button, or (2) reject it by pressing the *CANCEL* button. A *safe default* timeout of 10 seconds was selected. If no subject input was provided within this time, the date was automatically rejected.

- **Completion Time and Error Rates:** For the 240 (=8*30) test cases, the study yielded an average completion time of 6.386 seconds with a standard deviation of 2.388 seconds (see Figure 4.3). This shows that subjects made quick decisions regarding the timeliness of displayed dates. Among the 240 test cases, the false negative rate (reject dates that are not stale) was quite low, at 4.44%. No one rejected a date that was seven days in future, and only 6.67% (2 subjects) of the sample rejected dates that were one and three days in the future.

  The false positive rate (stale date accepted) was considerably higher, namely 17.33% on average. When subjects were shown dates that were 1 and 3 days earlier, the error rates were only 10% and 0%, respectively. Surprisingly though, when subjects were shown dates that were 29, 364 and 729 days earlier, the error rates shot up to 16.67%, 30% and 30%. We will elaborate on possible reasons for this spike in the discussion below.
- **SUS Scores and User Opinion:** Subjects that tested our implementation rated its usability at 76 on the System Usability Scale (SUS) [11]. We note that this is almost identical to the score of 77 obtained in [28], where subjects rated it based on a mock-up implementation on a Nokia N95 cell phone. The overall SUS score obtained is appreciably above the "industry average" of 70.1 [9], and indicates good usability and acceptability characteristics.

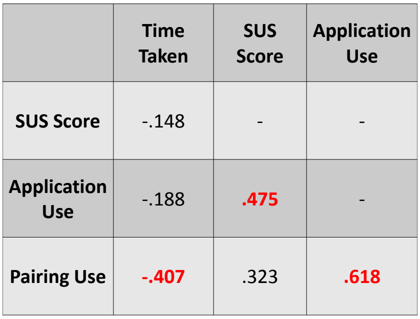
  Furthermore, 70% (21 subjects) stated that they would like this system on their own personal tags, while 23.33% (7 subjects) were neutral to the idea. The average score on a 5-point Likert scale was 3.78 with a standard deviation of 0.77.
- **Discussion:** As the results show, our method very rarely yields false negatives: users are capable of not mistaking valid (future) dates for past dates. Regarding false positives, however, the results are mixed. Stale days are, for the most part,

easily recognized as such. However, with stale years, error rates are quite high, at 30%. While we do not claim to know the exact reason(s) for this fact, some conjectures can be made. When confronted with a date, e.g., current dates on documents or expiration dates on perishable products, most people are used to first check day and month. They may not tend to pay as much attention to more blatant errors such as wrong year, perhaps because they consider it to be an unlikely event. We anticipate though that year mismatches will be quite rare in practice, since (as we mentioned earlier in the paper) tags can record the most recent *valid* date they encounter. Therefore, dates with stale year values will be mostly automatically detected and rejected by tags without the need for any user interaction. However, high user error rates in wrong year values can still pose a threat if a tag is not used for a year or longer.

Secure Device Pairing. We chose the "Copy" method described earlier for the device pairing tests. There were two primary reasons for this choice: our previous studies [32,27] had indicated low error rates, and the method is device-controlled and therefore resistant to rushed user behavior [29].

- **Test procedure:** First, each subject was briefed on the purpose of pairing personal RFID tags with personal devices (in this case, a laptop). Next, the subject's role in the protocol was described. Subjects were then asked to enter a random 5-digit number generated by the tag into the laptop. Upon correct number entry, they were notified of successful pairing via the tag and laptop displays, and a mock user interface depicting possible applications of the pairing was displayed on the laptop. Only a single test case was performed for each user.
- **Completion Time and Error Rates:** A total of 30 test cases were performed, yielding an average completion time of 23.904 seconds with a standard deviation of 8.272 seconds. Only 3.33% of the sample (1 subject) entered an incorrect number into the laptop that resulted in an error.
- **SUS Scores and Usability Analysis:** Before rating the pairing protocol on the System Usability Scale, subjects were clearly informed of the distinction between rating the pairing protocol and rating its applications. The SUS scale was only used to understand the usability of the former, and resulted in a score of 81.83%. This indicates very good usability and acceptability.

  Furthermore, 86.67% (26 subjects) indicated that they found the "Copy" method easy to use and that they wanted to use it more often for pairing. 83.33% (25 subjects) indicated that they were likely or very likely to use the applications that were now available as a result of the ability to pair their personal tags with other devices.
- **Discussion:** High SUS scores, low error rates and positive user feedback point to the usability of the "Copy" device pairing approach, and potential applications of tags paired with more sophisticated devices. An effective and usable pairing method should demonstrate high scores on all three measures. To better understand the correlations among four selected measures, we computed their cross correlations. Fig. 6 shows the Pearson correlation coefficients. Interestingly, there exist three medium to high correlations. These are between perceived ease of use of the pairing method and time to completion (medium: -.407), likelihood of using applications of



	Time Taken	SUS Score	Application Use
SUS Score	-.148	-	-
Application Use	-.188	**.475**	-
Pairing Use	**-.407**	.323	**.618**

Fig. 6. Pearson correlation coefficient matrix for tag-to-PC pairing

pairing and SUS score (medium: .475), and perceived ease of use of pairing method and likelihood of using applications of pairing (high: .618).

5 Conclusions

Recent advances in display technology and hardware integration have resulted in relatively inexpensive display-equipped RFID tags (DERTs). Their low cost coupled with achievable security properties make DERTs desirable and ready for real world applications.

In this paper, we made the case for using DERTs in several security-related contexts. In particular, we presented simple, intuitive solutions to several security problems with personal RFID tags. Our methods take advantage of the newly available user interface (display) for RFID tags and the presence of human owners. Preliminary usability studies suggest that target users find all our methods usable, and they are capable of performing their roles with reasonably low error rates. As more applications for DERTs are found, we believe that they will soon be in mass production and methods proposed in this paper will become applicable to a wide range of personal RFID tags.

Acknowledgements. The authors are grateful to NXP Semiconductors, especially to Thomas Suwald and Arne Reuter, for providing us with the display-equipped tags used in our studies. This work is supported in part by NSF Cybertrust grant #0831526.

References

1. Bestbuy To Put Gizmo Vending Machines In Airports, http://www.pcworld.com/article/149684/best_buy_to_put_gizmo_vending_machines_in_airports.html
2. BSI: Country Verifying Certificate Authority. https://www.bsi.bund.de/cln_174/DE/Themen/ElektronischeAusweise/CVCAePass/CVCAePass_node.html.
3. BSI: The New ID-Card, https://www.bsi.bund.de/cln_174/ContentBSI/Themen/Elekausweise/Personalausweis/ePA_Start.html.

4. Hid Omnikey 5321 Cl Usb Reader, `http://www.hidglobal.com/documents/OK5321_cl_ds_en.pdf`
5. Java Smart Card I/O, `http://java.sun.com/javase/6/docs/jre/api/security/smartcardio/spec/`
6. Logitech Wireless N305, `http://www.logitech.com/en-us/keyboards/keyboard/devices/6355`
7. Abadi, M., Burrows, C., Kaufman, C., Lampson, B.: Authentication and delegation with smart-cards. Science of Computer Programming 21(2), 93–113 (1993)
8. Aleskerov, E., Freisleben, B., Rao, B.: Cardwatch: A Neural Network Based Database Mining System For Credit Card Fraud Detection. In: Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr), March 23-25, pp. 220–226 (1997)
9. Bangor, A., Kortum, P., Miller, J.: An Empirical Evaluation Of The System Usability Scale. Int. J. Hum. Comput. Interaction 24(6), 574–594 (2008)
10. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
11. Brooke, J.: SUS: A "Quick And Dirty" Usability Scale. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A., McClelland, A.L. (eds.) Usability Evaluation in Industry. Taylor and Francis, London (1996)
12. Chan, P.K., Fan, W., Prodromidis, A.L., Stolfo, S.J.: Distributed Data Mining In Credit Card Fraud Detection. IEEE Intelligent Systems 14(6), 67–74 (1999)
13. Czeskis, A., Koscher, K., Smith, J.R., Kohno, T.: RFIDs And Secret Handshakes: Defending Against Ghost-And-Leech Attacks And Unauthorized Reads With Context-Aware Communications. In: CCS 2008: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 479–490. ACM, New York (2008)
14. Evans Jr., A., Kantrowitz, W., Weiss, E.: A User Authentication Scheme Not Requiring Secrecy In The Computer. Commun. ACM 17(8), 437–442 (1974)
15. Forget, A., Chiasson, S., Biddle, R.: Shoulder-Surfing Resistance With Eye-Gaze Entry In Cued-Recall Graphical Passwords. In: CHI 2010: Proceedings of the 28th International Conference on Human Factors in Computing Systems, pp. 1107–1110. ACM, New York (2010)
16. Heydt-Benjamin, T.S., Bailey, D.V., Fu, K., Juels, A., O'Hare, T.: Vulnerabilities in first-generation RFID-enabled credit cards. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 2–14. Springer, Heidelberg (2007)
17. Hoepman, J.-H., Hubbers, E., Jacobs, B., Oostdijk, M., Schreur, R.W.: Crossing borders: Security and privacy issues of the european e-passport. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S.-i. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 152–167. Springer, Heidelberg (2006)
18. Housley, R., Ford, W., Polk, W., Solo, D.: Rfc 5280: Internet X.509 Public Key Infrastructure Certificate and CRL profile (May 2008)
19. Juels, A., Molnar, D., Wagner, D.: Security And Privacy Issues In E-Passports. In: International Conference on Security and Privacy for Emerging Areas in Communications Networks, pp. 74–88 (2005)
20. Kainda, R., Flechais, I., Roscoe, A.W.: Usability And Security Of Out-Of-Band Channels In Secure Device Pairing Protocols. In: SOUPS: Symposium on Usable Privacy and Security (2009)
21. Kobsa, A., Sonawalla, R., Tsudik, G., Uzun, E., Wang, Y.: Serial Hook-Ups: A Comparative Usability Study Of Secure Device Pairing Methods. In: SOUPS: Symposium on Usable Privacy and Security (2009)
22. Kou, Y., Lu, C.-T., Sirwongwattana, S., Huang, Y.-P.: Survey Of Fraud Detection Techniques. In: 2004 IEEE International Conference on Networking, Sensing and Control, vol. 2, pp. 749–754 (2004)

23. Kumar, A., Saxena, N., Tsudik, G., Uzun, E.: Caveat Emptor: A Comparative Study of Secure Device Pairing Methods. In: IEEE International Conference on Pervasive Computing and Communications, PerCom (2009)
24. Micali, S.: Efficient Certificate Revocation. Technical Memo MIT/LCS/TM-542b, Massachusetts Institute of Technology (1996)
25. Micali, S.: Certificate Revocation System. United States Patent 5,666,416 (September 1997)
26. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: Internet Public Key Infrastructure Online Certificate Status Protocol- Ocsp. RFC 2560 (1999), http://tools.ietf.org/html/rfc2560
27. Nithyanand, R., Saxena, N., Tsudik, G., Uzun, E.: Groupthink: Usability Of Secure Group Association For Wireless Devices. In: 12th ACM International Conference on Ubiquitous Computing, Ubicomp 2010 (2010)
28. Nithyanand, R., Tsudik, G., Uzun, E.: Readers Behaving Badly. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 19–36. Springer, Heidelberg (2010)
29. Saxena, N., Uddin, M. B.: Secure pairing of "Interface-constrained" devices resistant against rushing user behavior. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 34–52. Springer, Heidelberg (2009)
30. Saxena, N., Uddin, M.B., Voris, J.: Treat 'em Like Other Devices: User Authentication of Multiple Personal RFID Tags. In: SOUPS 2009: Proceedings of the 5th Symposium on Usable Privacy and Security, p. 1. ACM, New (2009)
31. Perković, T., Čagalj, M., Saxena, N.: Shoulder-Surfing Safe Login in a Partially Observable Attacker Model. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 351–358. Springer, Heidelberg (2010)
32. Uzun, E., Karvonen, K., Asokan, N.: Usability analysis of secure pairing methods. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 307–324. Springer, Heidelberg (2007)
33. Wilkes, M.V.: Time Sharing Computer Systems. Elsevier Science Inc., New York (1975)

Forcing Johnny to Login Safely
Long-Term User Study of Forcing and Training Login Mechanisms

Amir Herzberg and Ronen Margulies

Dept. of Computer Science, Bar Ilan University
{herzbea,margolr}@cs.biu.ac.il

Abstract. We present the results of the first long-term user study of site-based login mechanisms which force and train users to login safely. We found that interactive site-identifying images received 70% detection rates, which is *significantly better* than passive indicators' results [15, 8, 12]. We also found that login bookmarks, when used together with 'non-working' links, doubled the prevention rates of reaching spoofed login pages in the first place. Combining these mechanism provides *effective prevention and detection* of phishing attacks, and when several images are displayed in the login page, the best detection rates (82%) and overall resistance rates (93%) are achieved. We also introduce the notion of *negative training functions*, which train users not to take dangerous actions by experiencing failure when taking them.

1 Introduction

Phishing, i.e., password theft via fake websites, is an extremely worrying, wide spread and worldwide phenomenon. With billions of dollars lost and dozens of percents increase in the amount of attacks over the years [1, 19, 20], it appears that there is still a need to improve the defenses against phishing.

Psychology can provide important insights that can help improve anti-phishing defenses, by understanding what makes users so susceptible to phishing. In [11], Karlof et al. described how humans tend to develop automatic responses to situations repeating themselves. In familiar situations, the human brain makes us respond *mindlessly* with the action that is usually most appropriate; for such responses, the psychologist Robert Cialdini coined the term *click whirr* responses [4]. Cialdini says that these responses are like pre-recorded tapes in our heads, and when we encounter a familiar situation we automatically "click the play button" (which makes a whirr sound, hence click whirr). Karlof et al. explained that users developed a click whirr response to login forms, and will *automatically* submit their credentials to a login form residing on an interface they have seen before. Our main focus is on influencing users' responses to spoofed emails and web pages, and their decision-making process.

In addition, as was clearly seen on our user study (section 5), two other click whirr responses are shared by most users. The first is to follow email links from

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 452–471, 2011.

a familiar sender, as clicking a link is a natural thing to do when meeting one (as was also noticed by Karlof et al.). The second is to put trust in a site's home page that looks familiar (even if not protected by SSL), and move to the site's login page when wanting to login (by clicking an "Enter Your Account" button/link). The three mentioned click whirr responses make the Internet a fertile ground for phishing attacks.

1.1 Current Mechanisms: Passive Indicators

Early web browsers include three main indicators to help users identify the websites they visit: the address bar (indicating the site's URL), the *https* prefix and the padlock/key image (both indicating SSL usage). Several experiments [5, 9] showed that users often enter their passwords *without validating them.* This is not surprising: the indicators are not very visible, and there is no mechanism forcing or training users to inspect them, so it is easier to just skip those checks.

Several proposals and implementations for enhanced indicators, most involving change to the browser, and few that require only support by the website, were introduced. Those indicators display warnings [21], a user-custom image/text to help the user identify the site [9, 17, 3], SSL Certificate information [9, 16], and emphasis of the domain name and protocol in the URL bar.

Both the 'classical' and 'enhanced' indicators are *passive*, i.e., they are *only displayed* to the user and *require no action* in a regular login. Several experiments measuring users' ability to detect fake sites using different (enhanced) indicators, resulted in mostly disappointing results [8, 15, 12].

1.2 Interactive Custom Indicators

We found that the use of *interactive custom indicators* can significantly improve the ability of users to avoid phishing. Interactive custom indicators force the user to interact with her customized (personal) indicator (image/text) in order to login. An example is Passpet [18] – a firefox extension which acts both as a password manager and an interactive custom indicator.

One important advantage of interactive custom indicators is that they can be implemented by changes to only the website, without requiring any change on the client side; this is significant as changes in the client side are often complex for users, requiring expensive support, and require support of multiple machines and browsers. With a site-based interactive custom indicator, the login page can hide the password text field until the user clicks her custom indicator.

The interactive nature of these indicators, which *conditions* users to find and click their custom image/text, makes users more alert to the indicator's absence on a spoofed login page. There was almost no study of the effectiveness of interactive indicators in detecting spoofed login pages [1].

[1] An exception are the encouraging results of the preliminary phase of this research (see in Dvorkin's thesis [6]): ∼20% – ∼40% detection rates for passive indicators, ∼85% for interactive.

1.3 Secure Login Using a Bookmark

The initial stage of a phishing attack is to get the user to enter a spoofed login page. A common scenario for the initial stage is to send a spoofed email containing a link to a spoofed login page. Users might follow email links, which could be risky since emails are easily spoofed. In addition, most sites' home pages are not protected by SSL and can be spoofed by a MITM; users might put their trust in a spoofed page looking similar to their target site's home page, and follow its "Enter Your Account" button which leads to the (spoofed) site's login page. Search engine results are also not protected by SSL and can be spoofed by a MITM, thus leading the user to a spoofed site.

A good habit for accessing high-value sites is to create a browser bookmark ('favorite') for a sensitive site's (https) login page, and *always surf to the site's login page via that bookmark*. Adida has presented BeamAuth [2], a *two-factor authentication* mechanism based on a login bookmark. In this mechanism, users receive a special login-bookmark from the website, containing a secret token, which identifies them to the site. To ensure that users will *always* login via their bookmark, the login page looks for the secret token and prohibits the login if a valid token is not supplied (an error message is displayed on the login page, which trains the user not to enter the login page in any way but the bookmark). A login bookmark ensures reaching the correct URL, and by containing an *https* prefix, ensures a secure channel is established.

Apart from initial and non-reliable results of this research [6], there was no study of the effectiveness of login bookmarks in preventing users from reaching spoofed sites. There was also no study of other mechanisms that aim to prevent users from following email links or entering the site's login page via its (non SSL protected) home page.

1.4 Challenges and Requirements for User Studies

User studies should try to emulate users' real-life activities. Most previous studies in the field [15, 8, 12] were short term (few hours) lab studies. Such studies experience problems in making participants act as in real life: if they know the true intention of the study, they will be more cautious than in real-life, and if a false intention is introduced they will be less cautious; even if a sense of risk is added (for . using their own bank accounts), the study's environment, which is not in the users' regular environment, might influence their behavior [14].

It is important to complement short-term lab studies with long-term real-life studies, where participants use an online system, with a different purpose than security, regularly for several months, and login from anywhere they want. This kind of study is closer to real-life, and even if the study's purpose is introduced, users' motivation to detect attacks is not expected to be higher than usual due to the constant usage and other purposes of the system. To the best of our knowledge, no long-term user study which examined users' response to emails and the detection rates of spoofed pages was previously conducted.

Table 1. Detection rates and overall resistance rates for a classic phishing attack. Cells are merged when results were combined for higher confidence or when it does not make sense to split (e.g. 'non-working' links don't affect the detection rates, only the prevention rates).

mechanisms	detection rate	resistance rate
none	19.61±4.95	40.22±10.24
bookmark only	42.56±5.61	49.84±14.77
bookmark+'non working' links		81.08±12.88
image only	59.84±6.24	76.12±8.44
bookmark+image	72.71±6.31	
bookmark+'non working' links+image		80±10.03
bookmark+4 images	81.94±5.17	93.24±7.8
bookmark+'non working' links+4 images		

1.5 Our Contributions

We have conducted an extensive long-term user study of real-life web and email activities, which included different kinds of simulated phishing attacks. We examined the effectiveness of different phishing defense mechanisms, including a login bookmark, interactive custom images and their combination.

We also tested mechanism sites can use to prevent users from reaching spoofed login pages – intentionally including 'non-working' links/buttons in the site's home page and email announcements. From the results we concluded that users need to *experience failure* in order to avoid dangerous actions, and introduced the notion of *negative training functions.*

We found significant differences between the mechanisms' detection rates and overall resistance rates (see table 1). From all the results of our study we derived a set of conclusions and guidelines that (high-value) sites can use. We present an important conclusion derived from table 1 in advance:

Conclusion 1. *By combining a login bookmark with 'non-working' email links and an interactive custom image, and displaying multiple images in the login page, the detection rates and overall resistance rates are higher than any other mechanism previously tested in a real-life experiment (82% and 93% respectively).*

Detailed discussion of the table and the study's results and conclusion is given in section 5. In our study we also measured the effectiveness of browsers' SSL certificate warnings and conducted a usability survey.

Another contribution is WAPP (Web Application Phishing-Protection), a site-based anti-phishing solution we designed and implemented, which combines a login bookmark with multiple interactive images to constitute a *conditioned-safe* login ceremony (see section 2.1).

1.6 Paper Organization

Section 2 describes the principles for effective anti-phishing mechanisms, which was used as a basis for WAPP (section 2.4) and for our user study (section 3). In section 4 we present a comprehensive threat analysis and users' expected behavior for simulated attacks, and section 5 presents the results and conclusions from our user study. Finally, section 6 presents the results of our usability survey.

2 Principles for Effective Anti-Phishing Mechanisms

2.1 Conditioned-Safe Ceremonies

Karlof et al. [11] introduced the notion of a *conditioned-safe ceremony*, which is a ceremony that "deliberately conditions users to reflexively act in ways that protect them from attacks", i.e., forces users to take actions that are safe.

Forcing functions, which were also mentioned by Karlof et al., are behavior-shaping constraints, used in the human reliability field aiming to prevent human errors. Forcing functions usually work by *preventing a user from progressing in her task* until she performs a specific action whose omission results in failure. Because users must take this action during every instance of the task, the forcing function trains them to *always* perform this action, and after a short experience will become a *click whirr response.*

To defeat conditioned-safe ceremonies, attackers will try to make users perform a dangerous action instead of the forcing function, thus bypassing its protection. For example, convincing users to follow a link to a spoofed login page instead of clicking the login bookmark which leads to the correct login page. Since such actions are indeed possible (as our study's results show), we introduce the notion of *negative training functions.*

Unlike forcing functions, negative training functions are *not* a part of the ceremony, and train users to *never* perform dangerous actions by making them *experience failure* when performing those actions. Negative training functions can come together with forcing functions to better train users of what should and what should not be done.

2.2 Design Goals for a Conditioned-Safe Login Ceremony

Since humans tend to make routine actions mindlessly, and in particular skip any voluntary action during a login ceremony, we should not fight this tendency. A conditioned-safe login ceremony should consist of *several forcing and negative training functions* – at least one function against each click whirr response that puts users in danger. In particular, there should be forcing and training functions against:

1. Automatic following of links.
2. Automatic submission of credentials.
3. Automatic entrance to a site's login page by clicking an "Enter Your Account" button in its home page (which could be spoofed).

2.3 Mechanisms of Interest

On our user study we focused on mechanisms which are either forcing or negative training functions, since we believed they will be the most effective mechanisms against phishing. In addition, such mechanisms were never tested before, apart from initial encouraging results by Dvorin [6]. The forcing functions mechanisms we used were:

1. A login bookmark, which forces users to login via their bookmark only, since a login attempt not via the bookmark results in failure.
2. An interactive custom image, which forces users to find and click their custom image in order to submit their password and login.

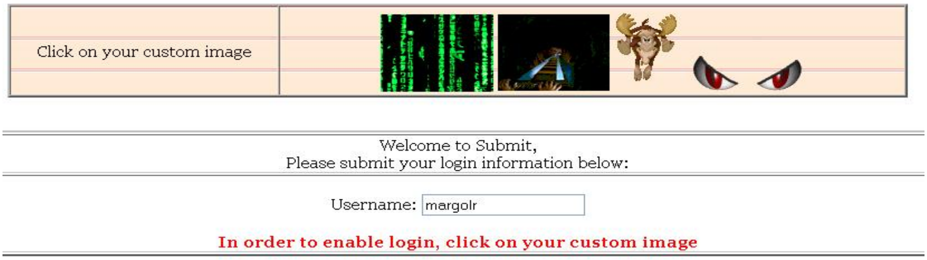
Since the bookmark and image clicks are necessary for each login, they provide *constant training*; the more a user is trained, lower are the chances for her to omit the forcing functions. The interactive custom image defends against automatic submission of credentials and the login bookmark defends against reaching a spoofed login page (for example by following email links or entering a site's login page via a non-SSL protected home page). The combination of the two mechanisms provides both *prevention and detection* of spoofed login pages, thus achieves *defense-in-depth* and guarantees that omitting only one of them will not suffice. Therefore, a combined method is also worth testing.

To login to a site implementing the combined method, the user has to first click her bookmark which leads her to the login page and sends the secret token to the site for initial identification. The site then recognizes the user and sends her custom image to be displayed in the login page. After the user clicks her custom image she will be able to submit her password. If her username, password and secret token match, the site allows her to login.

The login ceremony of the combined method is simpler and faster than typical login ceremonies, as it requires only two mouse clicks (for the bookmark and interactive image), and saves the need for typing the site's URL (or looking for it on a search engine) and moving from the site's home page to its login page. In addition it saves the need for typing the username, as it can be automatically filled-in (see [2]). The ceremony is even faster with browser auto-completion.

A variation of interactive custom images is to choose (and click) the custom image from a small set of random images on the login page. This variation makes the user even more aware of her custom image (as seen on our study), and improves the detection rates. With only one image displayed, the user's click whirr response could cause her to immediately click on any image displayed to her, especially if she forgot her image. Making her choose the correct image, reduces the chance for an immediate mindless click. In addition, with several images being used the site can notice when a user has forgotten her image (after several mistaken clicks), and can refresh her memory. Another idea is to present animated images which attracts the human eye more than static images and could increase memorability.

In addition to the login bookmark, we also tested two negative training functions aiming to prevent users from reaching spoofed login pages:



Click on your custom image

Welcome to Submit,
Please submit your login information below:

Username: margolr

In order to enable login, click on your custom image

Fig. 1. The login page requires the user to click her correct custom image

1. Intentionally including "non working" links to the login page in the site's email announcements; when clicked, the user reaches the login page which displays an error telling her to login only via her bookmark. This experience trains the user to never follow links.
2. Intentionally including a "non working" account-entrance button in the site's home page. When clicked, the user reaches the login page and displayed with a similar error page. This experience trains the user to never enter the site's home page in the first place when wanting to login.

2.4 WAPP

We designed and implemented WAPP (Web Application Phishing Protection), a server side solution which implements all the above mentioned mechanisms. We refer to a site implementing those mechanisms as a WAPP-protected site. A demo of WAPP is available at `http://submit2.cs.biu.ac.il/WAPP/`.

3 Long-Term User Study

3.1 Study's Framework System

For our long-term study we used an online exercise submission system called 'submit' (`submit.cs.biu.ac.il`), which is used by most courses at the computer science department of Bar-Ilan University. With the submit system, students submit their exercises and receive emails announcing new grades. Due to its wide usage, most users logged-in to the system dozens and even hundreds of times throughout the study. We have used the system for two years (four semesters), among a population of ~400 students, and present the results of the first three semesters (initial analysis of the results of the fourth semester correlate with the earlier results). In addition to its primary usage as an exercise submission system, we simulated several phishing attacks, and collected the attacks' results.

3.2 Introducing the Experiment

When users experience a sense of risk or are aware of the security purpose of the system, they become more concerned about security. Yet, its long-term usage and the fact that its primary usage isn't security, but rather exercise submission, should not cause more concern than user's real life concern for high-value sites. We had two variations of our experiment:

First Experiment – Weak Motivation. In the beginning of the study we announced an up to 5 points bonus in one of Herzberg's courses of their choice for correctly detecting attacks to provide the users with an incentive to cooperate. After analyzing the results of the first two semesters and users' answers to an online survey we applied at the end of the second semester, we found that 26% of the students did not cooperate with our experiment and did not try to detect attacks. We removed the results of those users (see appendix B) and concluded that further motivation for cooperation and a higher sense of risk are needed.

Second Experiment – Extra Motivation. In the third semester we introduced our study in a more informative way: on the first login, each user, including users from previous semesters, was introduced with an instructions page which shortly described: a) what phishing is and the extent of phishing attacks; b) who we are and what are our goals; and c) the experiment details. We asked the students to cooperate and promised our gratitude. We also announced the 5 point bonus and told users they will lose bonus points upon classification mistakes. Users that used the system in the previous year knew that the bonus points were indeed granted, and that they depend on the correct classification rates.

3.3 Users' Login Methods

Upon registration, each user was randomly assigned a login method from the following five methods:

image only an interactive custom image only
bookmark only a login bookmark only
bookmark+image a login bookmark combined with an interactive custom image
bookmark+4 images like the latter, where the login page displays 4 images
none no site-based indicator assigned, used as a control group

Users could only login using the method assigned to them upon registration. In normal usage of the system, `non-bookmark` users (`none` and `image only`) have reached the login page, which was running over https, via the system's home page, which was running over http, by providing their username. `Bookmark` users (`bookmark only`, `bookmark+image` and `bookmark+4 images`) that entered the login page the same way received an error message (except when they were attacked) telling them to login via their bookmark, and could not login. This simulated our second negative training function.

3.4 Users' Email Methods

Each user was also assigned an email method, which determined how she will get her new grade announcement emails from the system. We used three types of emails – 1) emails that contain a link to the system's login page telling the users they have to login to view their grade and submission details; 2) emails that contain the grade and submission details within the mail body and contain no link; 3) emails like the latter containing no link and also containing a warning at the end of the email body, saying that the system never includes links in its emails since clicking links in emails is dangerous. We refer to these email methods as `link`, `no link`, and `warning` respectively.

Users always got the same type of email except when the system had sent a spoofed email. `Bookmark` users from the link group received 'non-working' links regularly (except when they were attacked), as the links directed them to the system's login page where they were shown an error message. This simulated our first negative training function.

3.5 Attacks

When users tried to enter the system's login page via its home page or via their bookmark, there was a low probability for them to be randomly directed to one of the system's spoofed login pages. We uses low attack probabilities to prevent increased awareness due to frequent attacks. Spoofed emails were also sent to users with rather low probabilities; these emails contained similar content to the system's genuine emails sent to link users, apart from the fact that the links contained the URL of a spoofed page. See the full version of our paper [10] for a detailed description of the attacks and logging of the results.

4 Threat Analysis

In this section we attempt to analyze all realistic phishing attacks against a site implementing the mentioned forcing and negative training functions mechanisms (which we refer to as a WAPP-protected site). We hypothesize users' expected behavior in the different attack scenarios, and describe how we simulated those attacks in our user study.

Classic Phishing Attack. In a classic phishing attack a spoofed email containing a link to a spoofed login page is sent to the user. WAPP defends against this attack by training the user not to follow the link and always login via her bookmark. If the user makes an error of omission and follows the link, her interactive custom image won't be displayed and the user will most likely understand she reached a spoofed page and resist the attack. We executed this kind of attack in our user study and measured whether `bookmark` users and in particular users with 'non-working' links, are less likely to follow links, and if they did, whether the interactive custom image is effective in detecting the spoofed page.

Malicious Bookmark Replacement. An attacker might trick the user into replacing her WAPP bookmark, using for example, a spoofed email that mimics the site's registration email, or overriding the bookmark by creating a bookmark with the same name when the user visits the attacker's site.

Even if the bookmark has been replaced, the secret token is not revealed and the user will most likely identify she reached a spoofed site since her custom image does not appear, thus not provide her password. This way she could suspect the new installed bookmark, delete it, and replace it with the original bookmark she received from the site. We simulated the second part of the attack by redirecting `bookmark` users to a spoofed login page after clicking their bookmark, as if it was previously replaced, and measured the detection rates.

Pharming Attack. A MITM attacker who hijacks a DNS entry will direct the user's traffic for the legitimate website towards the attacker's machine. Since the bookmark link start with https, the user's browser will try to establish an SSL connection with the attacker's machine and notice an invalid certificate. When encounter an invalid certificate, modern browsers don't display the site's content, and display active certificate warning pages, with alarming text and colors, instead. Users are required to manually add an exception or approve in order to forward to the site.

Egelman et al. [7] found high percentage (79%) of users resisting phishing attacks with active browser warnings, which makes sense as it is an interactive forcing function which prevents users from progressing with the login. For a user entering the spoofed site despite the warnings, the attacker can gain the user's secret token and custom image, and will most likely gain the user's password as well. In our user study we simulated a pharming attack by using a spoofed login page in the same domain as the submit system, which used an invalid certificate. We measured the percent of users entering the spoofed page despite the browser warnings. For a user that did enter, though a MITM could present the user with her real custom image, we did not do that in our study, as no added value could be achieved from such an attack. Instead, we did not display the image and measured the detection rates as in the other spoofed pages. We wanted to find out if the detection rates are better when a preventive forcing function is combined with a detective one.

Spoofing the Home Page. Since most sites don't apply SSL at their home page (mostly for performance reasons), the site's home page could be spoofed. A MITM can change the site's home page or the results returned by a search engine in case the user looked for the site's URL. Another scenario is that an attacker sends a spoofed email with a link to a spoofed home page instead of a spoofed login page. After reaching the spoofed home page, the user might put trust in the site and forward to the (spoofed) login page by clicking an "Enter Your Account" button/link.

WAPP users would most likely not enter the site's home page in the first place, since by previously doing so they have experienced a login failure due to the 'non-working' account entrance button. Even if the user performs an error of omission, enters the spoofed home page and forwards to the spoofed login page, she won't see her custom image and will most likely notice the attack. We have simulated this attack for `non-bookmark` users (which always logged-in from the home page) by directing them to spoofed pages. `Bookmark` users trying to enter the login page via the system's home page mostly reached the genuine login page and received an error message, but some of their attempts also led them to a spoofed login page.

An attacker can try additional attack scenarios (which are discussed in appendix A) as a preliminary phase to a phishing attack, in order to get a hold of the user's secret token. We did not execute these attacks due to previous studies' results, complication, and ethical reasons. Our study's results empirically proved that WAPP is well protected against all kinds of realistic phishing attacks. Only two-phased attacks that try to gain the user's secret token prior to a phishing attack could effectively defeat WAPP. These complicated attacks require more effort and resources from phishers, and are likely to significantly reduce phishers' motivation to attack users of WAPP-protected sites in the first place.

5 Study Results and Conclusions

We already presented table 1, which described the detection rates and overall resistance rates of the different mechanisms against a classic phishing attack, in the introduction of this paper. We found that by combining all mechanisms the best detection and prevention rates (and hence overall resistance rates) are achieved. In this section we deal with detection and prevention rates and with the different attacks individually, and present the conclusions from the study's results.

5.1 Detection Rates Summary

In this subsection we focus on the detection rates of spoofed pages that have been entered, i.e not including invalid certificate pages which users were transferred to but not entered nor emails with spoofed links which were not clicked. Figure 2 shows a summary of the detection rates for the different defense mechanisms. Results are divided to the two variations of our study – 'weak' stands for students' weak cooperation and sense of risk in the first experiment and 'strong' stands for students' stronger cooperation and sense of risk in the second experiment. 'Combined' stands for a wighted average of the results of all three semesters, where we gave the 'strong' variation a weight of 0.7, since we believe it better suits real-life and had shorter evaluation time. The results show that:

Conclusion 2. *There is a* significant gap *between the detection rates of the different methods ranging from 20% for* `none` *users (in the combined results), up to 42% for* `bookmark only`*, 60% for* `image only`*, 72% for* `bookmark+image` *and 82% for* `bookmark+4 images`*. In the strong variation of our study, this method achieved even better and outstanding detection rates of 87%.*

In particular, `image` users (`image only`, `bookmark+image` and `bookmark+4 images`) detected *more than twice* the percent of attacks detected by `non-image` users (`none` and `bookmark`). Therefore:

Conclusion 3. *The interactive custom image is a highly effective forcing function against automatic submission of credentials.*

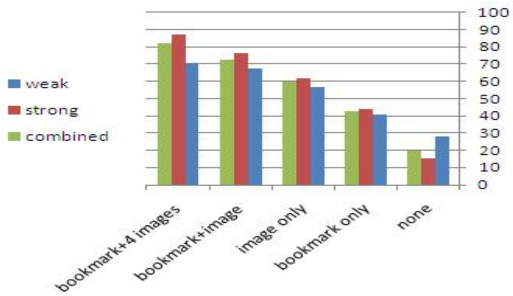


Fig. 2. Detection rates found for the different defense mechanisms

Another observation is that `bookmark only` users received better detection rates than `none` users, and `bookmark+image(s)` users received better detection rates than `bookmark only` users. We can conclude that:

Conclusion 4. *The login bookmark increases the detection rates, and its advantage is not limited to prevention.*

Finally, we noticed that the more effective a mechanism is, larger the gap in detection rates for that mechanism between the two versions of the study. In addition, the strong version of our study shows larger significance in the detection rates between the different mechanisms than in the weak version (see appendix C for further details). This implies that whether users of non-effective detection mechanisms (bookmark only and none) were cooperative did not affect their detection rates. On the contrary, when users of effective detection mechanisms were cooperative, their detection rates significantly increased. Therefore, we conclude that the strong version of our study shows the true potential of the mechanisms and better suits user's real-life behavior for sensitive sites.

5.2 Users' Response to Emails

In this subsection we deal with users' response to emails sent, both spoofed and non-spoofed. Table 2 shows the spoofed links following rates for `bookmark` and `non-bookmark` users, w.r.t. the different email methods. First we can see that there is no significant difference between `no link` and `warning` users, i.e:

Conclusion 5. *Warnings against following links in legitimate emails don't contribute in preventing users from following links in spoofed emails.*

Our results for email warnings correlate with the results achieved by Karlof et al.'s study [11]. Now, let's focus on `non-bookmark` users. The legitimate emails of `non-bookmark` users which normally receive no links (and receive their grades within the email body) look entirely different than the spoofed emails (not containing the grade and asking them to follow a link to watch it). Despite the difference, the percent of links followed by those users is as high as `non-bookmark link` users, which their legitimate and spoofed emails are similar (both contain working links). From this we conclude that:

Table 2. Links following rates for `bookmark` and `non-bookmark` users

method	email method	followed	sent	following rate
`non-bookmark`	`no link`	50	69	72.46 ± 8.84
`non-bookmark`	`warning`	68	100	68 ± 7.67
`non-bookmark`	`‘working’ link`	95	139	68.34 ± 6.49
`bookmark`	`no link`	53	81	65.43 ± 8.69
`bookmark`	`warning`	74	97	76.29 ± 7.1
`bookmark`	`‘non-working’ link`	36	116	31.03 ± 7.06

Conclusion 6. *Users don't distinguish between spoofed and non-spoofed emails, even if the email's structure is non-familiar.*

This can be explained by the fact that clicking a link is a click whirr response and the natural thing to do, and users follow email links regularly. Users might also put too much trust in the emails' ‘From’ header, which can be spoofed easily, and/or think that the system has changed its email announcements' structure.

Finally, and most important, the results show that `bookmark` users that normally receive no links follow a similar percent of spoofed links as `non-bookmark` users. Only `bookmark` users that normally receive (‘non-working’) links follow *less than half* the percent of spoofed links followed by `non-bookmark` users. We found similar significant difference for non-spoofed emails. Therefore:

Conclusion 7. *The login bookmark is only effective against automatic following of links when users receive “non working” links in genuine emails and experience failure in reaching the login page by following a link.*

From this conclusion we derive that:

Conclusion 8. *Sites should intentionally include “non working” links in their email announcements to (constantly) train their users not to follow email links*[2].

5.3 Spoofed Home Page Attacks Summary

In this subsection we focus on the spoofed home page attack (which lead the users to a spoofed login page). For all methods, we noticed that:

Conclusion 9. *Detection rates are lower when users enter spoofed login pages from the system's home page, in comparison to reaching them via email links or bookmark clicks*[3].

A possible explanation is that:

[2] Even if the site does not send email announcements at all, it is advised to send announcements from time to time just for the training's sake.

[3] 17% vs. 26% for `none` users, 18% vs. 46% for `bookmark only`, 58% vs. 63% for `image only`, 57% vs. 74% for `image+bookmark` and 74% vs. 83% for `bookmark+4 images`.

Table 3. Entrance rates (amount entered from amount transferred) and detection rates (amount noticed from amount entered) for invalid certificate spoofed pages

method	entered	transferred	entrance rate	noticed	detection rate
none	14	31	45.16 ± 14.71	1	7.14 ± 11.32
bookmark only	7	46	15.22 ± 8.71	4	57.14 ± 30.77
image only	7	23	30.43 ± 15.78	5	71.43 ± 28.08
bookmark+image	4	33	12.12 ± 9.34	4	100
bookmark+4 images	15	34	44.12 ± 14	13	86.67 ± 14.44
total	47	167	28.14 ± 5.72	27	57.45 ± 11.86

Conjecture 1. Users put high trust in the home page of a familiar-looking site, even if it does not provide an SSL certificate.

With reduced detection rates in this attack, prevention gets higher importance. The vast majority of all `bookmark` users tried to enter the site's login page via its home page, despite their bookmark. To prevent users from doing so, the site can choose not to include an account-entrance button in its home page at all, or to include a 'non-working' button which leads the user to an error page.

We only tested the second option, and a closer look at the attacks log showed that in spite the rather high attack probability, only two `bookmark` users were attacked more than once. Therefore:

Conclusion 10. *Combining the login bookmark with a "non working" account-entrance button in the site's home page achieves effective prevention.*

Both options experience the user with failure when wanting to enter the login page via the home page, and train the user not to enter the home page in the first place when wanting to login. Yet, when a user is triggered to enter a spoofed page looking similar to her target site's home page (for example by following a link), which includes an account-entrance button, she might be tempted to click it, as she did not experience failure in this specific action. Since the vast majority of `bookmark` users did click the account-entrance button on our study, it is important to experience failure with the button click itself. Therefore:

Conclusion 11. *Sites should intentionally include a "non working" account-entrance button/link in their home page.*

5.4 Effectiveness of Active Browser Warnings

In this subsection we examine how well modern browsers' active warnings prevent users from entering spoofed pages with invalid certificates, and for the users that did enter those pages, what are the detection rates. Table 3 shows the percent of users from each method entering the spoofed site, and the detection rates for those who entered.

From the table it seems that the active browser warnings prevented 72% of the users from entering the spoofed page, which correlate with the results achieved by Egelman et al.'s study [7]. Therefore:

Conclusion 12. *Active browser warnings are an effective forcing function against spoofed sites with invalid certificates (and in particular pharming attacks).*

It Is also noticeable that `image` users have better detection rates when entered the invalid certificate page. In addition, as seen before, `bookmark+image` and `bookmark+4 images` users had better detection rates than `image only` users. Therefore, we assume the following generalization:

Conjecture 2. Combining a preventing forcing function with a detecting forcing function increases the detection rates.

5.5 False Negatives

We've seen that throughout the study only one eighth of all users have falsely reported a spoofed page, mostly once or twice within a few minutes. Since a reasonable amount of false negatives is accepted and better than false positives (better safe than sorry), we conclude that:

Conclusion 13. *All the tested mechanisms does not confuse users to falsely classify the legitimate site as spoofed.*

6 Usability Survey

Since users are forced to set the bookmark (or an authentication cookie as alternative) on each computer or browser they use, this process should be secure and usable. If the bookmark setup involves manually surfing to the website and providing some identifying information, users are susceptible to phishing attacks each time they set the bookmark. To avoid this, bookmark setup has to be *via a secondary channel.* Many of today's websites send a registration email containing a verification link which the user has to click on to complete her registration. This process can also be used to create the bookmark, and we used it on our study.

Keeping the registration email around enables a bookmark setup on other computers. If the user loses the registration email, she can request it to be sent again (to the same email address). A second email address and an SMS can also be used for fallback bookmark recovery. On emergencies where the user has no option to recover her bookmark link, social authentication techniques [13], which were originally suggested for password recovery, can be used for bookmark recovery.

Though bookmark setup is not as immediate as the login ceremony, we believe most users usually access high-value sites only from few computers. We have tested this assumption and the overall usability of the mechanisms in our usability survey, which included the results of 136 `bookmark` and/or `image(s)` users.

The results confirmed our assumption, as users enter high-value sites from only 1.75 computers in average; medium-values sites such as social networks or webmail are entered from 3.26 computers on average.
In the survey's introduction we mentioned the high prevention and detection rates achieved by the login bookmarks and interactive custom images and asked users to say if the would want to login with those mechanisms[4] to medium-value and high-value sites. The results showed that most users (72% ± 7.42) want to use a login bookmark to access high-value sites, and half of them (51% ± 8.31) also want to use it on medium-value sites. In addition, only 36% of the users who did not want to use the bookmark picked the bookmark setup as the reason why they wouln't want to use this method. We can conclude that:

Conclusion 14. *When considering their security benefits, users are willing to use login bookmarks and other mechanisms that require registration of the computer via a secondary channel.*

We also found that 64% of the users are willing to use interactive custom images on high-value sites and 41% of them also want to use them for medium-value sites. The results are encouraging, and we expect them to be much higher due to two main reasons that biased users' answers in the survey and are described in appenidx D.

7 Conclusions

We have conducted a realistic long-term user study which tested the effectiveness of different defense mechanisms that use forcing and negative training functions, against different phishing attacks. The study's results showed that forcing and negative training functions are very effective in both prevention and detection due to their constant trainings. A combined method, which uses a login bookmark with interactive custom images, displays several images in the login page, and intentionally includes "non working" links in the site's email announcements and a "non working" account-entrance button in its home page, received outstanding prevention and detection rates.

Acknowledgments. The authors would like to thank Ben Adida for his feedback and helpful suggestions and Alex Dvorkin for the initial results of this study. This work was supported by Israeli Science Foundation grant ISF1014/07.

References

[1] Aaron, G., Rasmussen, R.: Global Phishing Survey: Trends and Domain Name Use in 2H2009. Anti-Phishing Working Group (May 2010), `http://www.antiphishing.org/reports/APWG_GlobalPhishingSurvey_2H2009.pdf`

[4] Bookmark only and `image only` users were only asked a yes/no question about their method and `bookmark+image(s)` users were given four options – bookmark only, image only, both or none.

[2] Adida, B.: Beamauth: two-factor web authentication with a bookmark. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 48–57. ACM, New York (2007)

[3] Sitekey Bank of America, http://www.bankofamerica.com/privacy/index.cfm?template=sitekey

[4] Cialdini, R.: Influence: Science and Practice, 5th edn. Allyn and Bacon, Boston (2008)

[5] Dhamija, R., Tygar, J.D.: Why phishing works. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 581–590. ACM Press, New York (2006)

[6] Dvorkin, A.: Evaluation of the Tools for User Protection against Web Site and Electronic Mail Based Attacks. Master's thesis, Bar-Ilan University (December 2008)

[7] Egelman, S., Cranor, L.F., Hong, J.: You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In: CHI 2008: Proceeding of the Twenty-sixth Annual SIGCHI Conference on Human Factors in Computing Systems, pp. 1065–1074. ACM, New York (2008)

[8] Herzberg, A.: Why Johnny can't surf (safely)? Attacks and defenses for web users. Computers & Security (2008)

[9] Herzberg, A., Jbara, A.: Security and identification indicators for browsers against spoofing and phishing attacks. ACM Trans. Internet Techn. 8(4) (2008)

[10] Herzberg, A., Margulies, R.: Long-term user study of forcing and training login mechanisms against phishing. Tech. rep., Bar Ilan University (March 2011), http://submit2.cs.biu.ac.il/WAPP/WAPP_primary.pdf

[11] Karlof, C., Tygar, J.D., Wagner, D.: Conditioned-safe ceremonies and a user study of an application to web authentication. In: SOUPS 2009: Proceedings of the 5th Symposium on Usable Privacy and Security (2009)

[12] Schechter, S., Dhamija, R., Ozment, A., Fischer, I.: The emperor's new security indicators. In: SP 2007: Proceedings of the 2007 IEEE Symposium on Security and Privacy, pp. 51–65. IEEE Computer Society, Washington, DC, USA (2007)

[13] Schechter, S., Egelman, S., Reeder, R.W.: It's not what you know, but who you know: a social approach to last-resort authentication. In: CHI 2009: Proceedings of the 27th International Conference on Human Factors in Computing Systems, pp. 1983–1992. ACM, New York (2009)

[14] Sotirakopoulos, A., Hawkey, K., Beznosov, K.: "i did it because i trusted you": Challenges with the study environment biasing participant behaviours. In: SOUPS User Workshop (2010)

[15] Wu, M., Miller, R.C., Garfinkel, S.L.: Do security toolbars actually prevent phishing attacks? In: CHI 2006: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 601–610. ACM, New York (2006)

[16] Better website identification and extended validation certificates in ie7 and other browsers (November 2005), published in Microsoft Developer Network's IEBlog http://blogs.msdn.com/b/ie/archive/2005/11/21/495507.aspx

[17] Yahoo: What is a sign-in seal?, http://security.yahoo.com/article.html?aid=2006102507

[18] Yee, K.-P., Sitaker, K.: Passpet: convenient password management and phishing protection. In: SOUPS 2006: Proceedings of the Second Symposium on Usable Privacy and Security, pp. 32–43. ACM, New York (2006)

[19] Gartner survey shows phishing attacks escalated in 2007 more than $3 billion lost to these attacks (2007), http://www.gartner.com/it/page.jsp?id=565125

[20] Gartner says number of phishing attacks on u.s. consumers increased 40 percent in 2008 (2008), `http://www.gartner.com/it/page.jsp?id=565125`

[21] Mcafee siteadvisor (2009), `http://www.siteadvisor.com/`

A Additional Attack Scenarios

1. ***Tricking the User to Give Away the Secret Token.*** An attacker could send a spoofed email on behalf of the target site's admin announcing some sort of technical problem and asking the user to copy&send the bookmark URL to the attacker, or to forward the registration email/SMS to the attacker. Even if this attack works, and the attacker finds the user's custom image, a second stage of the attack is still necessary. The attacker has to succeed in a phishing attack convincing the user to follow a link in another spoofed email. For this phase, the custom image won't help, but the constant trainings of the bookmark and non-working links should assist in preventing the user from following the spoofed link.

   Karlof et al. [11] included a similar attack in their long-term, real-life study. Each time a user had to register a new computer (to receive a cookie), the website sent her an email with a URL containing a single-use secret token. After the link click, the user reached a page which set a persistent authentication cookie, and deleted the token from its data base, to prevent further usage. Karlof et al. executed two kinds of attacks – asking the users to copy&paste the link to a spoofed site or forward the registration email to the attacker. In both attacks users got a message announcing "technical problems". The researchers stated that email-based registration is a conditioned-safe ceremony, which makes users click on the link they see rather than making one of the actions they have been asked to do by the attacker, which are of course more complicated to perform than a link click. Results have shown ∼40% success for those attacks.
2. ***Attacking the Email Account.*** If a registration email is used and the attacker breaks into a user's email account, the WAPP token is compromised, and the attacker will gain the user's custom image. Like the previous attack, the attacker needs to perform a phishing attack hoping the user will make an error of omission and follow a spoofed link sent to her.
3. ***Temporarily Using the User's Computer.*** An attacker might temporarily gain access to the user's computer and immediately gain the user's secret token and custom image. Fortunately, the attacker will not be able to immediately login as the user, as he still needs her password. This reduces to the last two attacks, requiring the attacker to phish the user hoping she will make an error of omission and follow a spoofed link.
4. ***Malware on the user's computer.*** An attacker who injects untrusted code into the user's computer can completely control the system, read the browser's bookmark content, and keylog the user's password. WAPP is completely vulnerable to this kind of attack. On the other hand, WAPP can assist in protection against malware, since attackers might phish downloads

sites in purpose of convincing users to download and install malware. Even though downloads sites usually don't maintain users' information and does not require a login, they could also integrate WAPP to prevent phishing attacks and allow only users who clicked the bookmark and their custom image to reach the downloads page.

B Removing Outliers

Prior to analyzing the results, we had to remove the results of users that did not cooperate with our experiment. Our methodology in finding outliers was to find the amount of attacks and amount of detections for each user. Then, for each method, we looked at the average amount of attacks and detections for all users from that method. By filtering users that had fewer detections than the average of their method and more attacks, than non-cooperating users were easy to find. For instance, if `bookmark+4 images` users had 4.48 detections of 6.77 attacks in average, than we filtered users with 4 hits or less which were attacked 7 times or more. For all methods, our filtering technique had left only users that were *noticeably non-cooperating*, for example having 0 detected attacks of 7 attacks or 1 of 23 and so on, without any 4 of 7 or other values which are just a bit lower than the average.

With the mentioned filtering technique we found 108/411 (26.27%) non-cooperative users in the first two semesters, and 75/402 (18.65%) in the third semester, 39 of whom (52%) were also non-cooperative in the previous semesters. In addition, several more non-cooperative users were hand-picked and removed. Theses included also users with many hits from many attacks. For instance, one user had 47 hits of 47 attacks; by closely looking at his attack-log entries, we saw that he have reached a spoofed page and did not enter his password. The same page was visited over and over, with exactly one hour between each visit, suggesting that the user had left his browser open and the browser is being refreshed constantly every hour.

To support our assumptions that the filtered users were indeed non cooperative, we have sent them an email asking them if they did not try to find spoofed pages (telling them that it is OK if they did not). 68% non-cooperating users of all three semesters had confirmed by email, none rejected, and the others did not respond.

C Different Versions of Our Study – Conclusions

Let's examine the difference between the two versions of our study: weak vs. strong cooperation and sense of risk. Assuming that users of all methods felt similar will to cooperate and similar sense of risk in each version of the study, one would expect to find a similar increase in detection rates for each method between the two versions, and similar gaps in detection rates between the methods in each version (for instance, a 5% increase in detection rates for all methods, keeping the gaps between different methods the same for the weak and strong versions).

Table 4. Increase in detection rates between the two version of the study

method	weak	strong	increase
`none`	27.88	15	-46.21
`bookmark only`	40.5	43.65	7.8
`image only`	56.54	61.9	9.48
`bookmark+image`	67.6	76.07	12.53
`bookmark+4 images`	70.7	87.07	23.16

Yet, the results turned out to be different than expected; it is noticed (see table 4) that the more effective a method is, larger the gap in detection rates for that method between the two versions of the study and in the strong version of our study there was a larger gap in the detection rates between the different methods. In particular, there was only a minor increase in detection rates for `bookmark only` users and a noticeable *decrease* for `none` users, suggesting that whether these users cooperated and felt a sense of risk or not *had no affect on the results*. On the other hand, for `bookmark+4 images` users the increase is the largest (23%), suggesting that lack of cooperation and sense of risk highly affected their results. From this we conclude that lack of cooperation and sense of risk inserts noise to the statistical data, and the true potential of each method's effectiveness is evidenced in the strong version of the study.

D Interactive Images Usability

There were two reasons that biased the survey answers of `image` users. First, we performed an additional experiment before the survey, where users were asked to remember few custom images instead of just one, and one of the custom images was randomly chosen and displayed on the login page on each login, along three non-custom images. This caused approximately half of the users that did not wish to use the interactive custom image to complain about the method, due to mistaken clicks which increased the authentication times. Most mentioned that a single custom image would be better. Due to the bias caused by the multiple-images experiment, we believe many of the users would want to use a single custom image.

Second, most of the other users that did not wish to use the interactive custom image stated that they don't need its protection, or because they did not understand its purpose (many thought of it as a user-to-site identification instead of the opposite). Those users did not say it was a bother to use the image, so we believe most of them won't be unhappy if it was applied on a site they use.

Towards a Mechanism for Incentivating Privacy

Piero A. Bonatti, Marco Faella, Clemente Galdi, and Luigi Sauro

Università di Napoli "Federico II"

Abstract. The economic value of rich user profiles is an incentive for providers to collect more personal (and sensitive) information than the minimum amount needed for deploying services effectively and securely. With a game-theoretic approach, we show that provider competition can reduce such information requests. The key is a suitable mechanism, roughly reminiscent of a Vickrey auction subject to integrity constraints. We show that our mechanism induces rational providers to ask exactly for the user information strictly necessary to deliver their service effectively and securely. In this framework, maximal attribute disclosures become more difficult to achieve.

1 Introduction

Web sites frequently ask their users for personal, sensitive information before granting access to their full functionalities. For example, some of the most popular web sites and services for e-commerce and social networking collect user name, birth date, gender, detailed address, credit card information, and—in some cases—even sex preferences, and political and religious views. Some of these fields can be easily aggregated to form *quasi identifiers* [7,19,16], that is, combinations of attributes such that the probability of their matching a single person is high. (i.e., the values of such attributes uniquely identify an individual with high probability). Releasing fake data is generally not an appropriate or sufficient privacy-preserving measure, as the correctness of some fields may be essential for correct functionality (e.g. credit card information for a purchase and home address for parcel delivery). The ongoing deployment of digital IDs and other cryptographically verifiable documents will further exacerbate this problem.

The above information requests are not parsimonious, in general. Rich user profiles have a significant economic value that constitutes an incentive for increasing the amount of user information collected (and for its disclosure to third parties). Therefore, providers are encouraged to ask for more information than the minimum required to deploy a given service effectively and securely.

Competition may contrast this trend. Indeed, a significant number of users have expressed concern over privacy during online interactions (e.g. user complaints have already influenced Facebook's privacy policy and services), and analysts say that privacy may become a factor of competition [5,17,9].

In this paper we take these analyses seriously and move the first steps towards mechanisms that may increase privacy by exploiting competition between providers. More precisely, such mechanisms should encourage providers to be *truthful*, that is, to ask nothing more than the minimum information sets necessary for correct and secure

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 472–488, 2011.

service deployment. Moreover, such approaches should supply users with *all* of the alternative ways of fulfilling the provider's policy, so that users can choose the alternative that they prefer from a privacy viewpoint.

Setting up a widely applicable mechanism of this kind is, of course, a very complex task. Open questions include at least the following: How should users and providers interact? (E.g. direct interaction vs. mediation by trusted third parties; single interactions vs. prolonged negotiations.) What is the interplay of information disclosure and nonfunctional service properties such as cost and quality of service? In particular, can providers compensate for more invasive information requests with such nonfunctional properties? Can the truthful mechanisms developed by economists be adapted to the scenarios described so far?

A complete answer to all of these questions lies beyond the scope of this paper (and any single paper of standard size). As a first step, here we focus on the last of the above questions, which concerns an essential prerequisite for the mechanisms of our interest and, more generally, for all the transactions where "payments" consist in user information disclosures. In these application scenarios, the payment means is naturally discrete and partially ordered, either in quantitative terms (e.g., according to set inclusion), or in qualitative terms (e.g., based on sensitivity). On the contrary, all standard mechanisms developed in microeconomics are based on totally ordered payment domains, e.g., money. Therefore, reconstructing such mechanisms in order to fulfill the requirements of our scenarios is a problem of general interest—and requires nontrivial changes in the underlying mathematics, as discussed below.

The mechanism studied in this paper is analogous to an auction: The user is the auctioneer; the providers "buy" the user's preference and "pay" for it by reducing the amount of (economically valuable) user information requested. The auction mechanism should be *truthful*, that is, information requested by rational selfish providers should match what is *actually* needed to deploy the service correctly and securely.

Perhaps, the most famous truthful auction mechanism is the second-price auction introduced by Vickrey [20]. It is a "one-shot" mechanism (bidders submit their offers in parallel, then the auctioneer evaluates them); the best offer "wins", and the price payed is the second best offer. As we anticipated, some of the main technical differences between Vickrey's mechanism and ours concern the range of bids and utility functions, that here is discrete and partially ordered rather than continuous and totally ordered. Of course, in a partial ordering it is not immediately clear how to generalize the notion of second price.

Even if a user may use a same service multiple times, a one-shot mechanism without memory of previous decisions may be appropriate in many cases. In particular, consider any transaction with no monetary costs, that is, where the only "cost" for the user consist of the personal information released. Usually, after such information has been disclosed and the user's profile created, subsequent service usage requires no further disclosures, so (from a privacy perspective) after the first information release the service can be used for free. Consequently, there is no need for further auctions until services change.

The paper is organized as follows: Section 2 introduces the formal framework and describes its formal properties. Related work is discussed in Section 3. Section 4 concludes the paper with a final discussion of the results and some interesting directions for further work. Proofs can be found in the appendix.

2 Formal Framework

The formal framework is relative to an arbitrary but fixed (implicit) service of interest to the user. The set of agents A is identified with an initial segment of the natural numbers: $A = \{0, 1, 2, \dots, N\}$. The user is represented by 0 and the providers by $1 \leq i \leq N$. We assume that the services deployed by the providers are all equivalent from the user's perspective. The set of information items that can be requested and released before service access is $C = \{c_1, \dots, c_z\}$. By analogy with trust negotiation frameworks [2,21,22], information items will be sometimes called *credentials*. The powerset of C is denoted by $\mathscr{P}(C)$.

Credential sets may have different sensitivity. The sensitivity order is modelled with a strict partial order $\prec$. When a credential set r_2 is more sensitive than a credential set r_1, we write $r_1 \prec r_2$. Let $r_1 \preceq r_2$ iff either $r_1 \prec r_2$ or $r_1 = r_2$. We assume that $r_1 \subseteq r_2$ implies $r_1 \preceq r_2$ (intuitively, by enlarging information sets, their sensitivity can only increase).

Several concepts, including policies, will be based on *thresholds* over $\mathscr{P}(C)$, that is, sets of sets of credentials $\theta \subseteq \mathscr{P}(C)$ such that $\theta \neq \emptyset$ and for all distinct $r, r' \in \theta$, $r \nsubseteq r'$. Θ denotes the set of all thresholds over $\mathscr{P}(C)$.

The user's policy is a threshold $pol_0 \in \Theta$. Intuitively, pol_0 represents the maximal sets of information items that the user is willing to disclose to access the service. Formally, a request $r \subseteq C$ is *admissible* iff $\exists r' \in pol_0 : r \subseteq r'$; we denote with $adm(pol_0)$ the set of admissible requests.

Example 1. Suppose $pol_0 = \{\{login, passw\}, \{card\_num,\ exp\_date\}\}$. This policy means that the user is willing to disclose either her login-password pair or her credit card number and expiration date. A request for the credit card number alone is admissible, too, as $\{card\_num\}$ is a subset of the second element of pol_0 and hence $\{card\_num\} \in adm(pol_0)$. On the contrary, $\{login, card\_num\}$ is not admissible, because it is not contained in any element of pol_0. □

Symmetrically, each provider i has a policy $pol_i \in \Theta$ that encodes the minimal (alternative) sets of information items that suffice to deliver the service securely and effectively. Formally, a credential set $r \subseteq C$ *fulfills* pol_i (and grants access to the service) iff $\exists r' \in pol_i : r \supseteq r'$; in the following $ful(pol_i)$ is the set of all $r \subseteq C$ that fulfill pol_i. The *policy profile* is the vector $\mathbf{pol} = \langle pol_0, \dots, pol_N \rangle$.

A provider i may decide to ask users for more information than what is prescribed by pol_i; the actual information request is called a *strategy* and corresponds to what is traditionally called policy in standard access control frameworks (because it determines which conditions must be fulfilled to access the service). Formally, a *strategy profile* for $\mathbf{pol}$ is any vector $\mathbf{req} = \langle req_1, \dots, req_N \rangle$ such that (i) $req_i \in \Theta$, and (ii) $req_i \subseteq ful(pol_i)$.

Each strategy req_i is the information request that provider i submits to the user. Each credential set $r \in req_i$ is an alternative way of fulfilling i's request, that is, the user must release a set of credentials $r' \in ful(req_i)$ in order to access the service deployed by i. The requests of provider i are required (by the second condition above) to fulfill the minimal requirements imposed by pol_i. Therefore the user's response r' is guaranteed to satisfy pol_i, too. Note that req_i might omit some ways of accessing the service. That is, for some $r \in pol_i$ there may be no $r' \in req_i$ such that $r' \supseteq r$. In this way, a provider may force the user to disclose credentials that are of greater interest for the provider.

Example 2. With reference to Example 1, a provider that can technically support both account-based and pay-per-use access would have the policy: $pol_1 = \{\{login, passw\}, \{card\_num, exp\_date\}\}$. The two members of pol_1 represent the two minimal information sets that need to be collected in order to grant the service. The request of provider 1 may in general be different. For instance, if $req_1 = \{\{card\_num, exp\_date, birth\_date\}\}$, the provider is trying to (i) force the user to disclose her credit card information rather than her account information, and (ii) obtain the user's birth date, which is not strictly necessary to service delivery. Note that the only element of req_1 is a superset of the second element of pol_1, therefore the request fulfills pol_1. However, it is not admissible w.r.t. pol_0. Now suppose that $req_1 = \{\{login, passw\}, \{card\_num, exp\_date, birth\_date\}\}$. In this case, due to the additional alternative $\{login, passw\}$, req_1 is admissible for pol_0 and fulfills pol_1. The user can release the set $\{login, passw\}$ and access the service. □

A strategy req_i is *truthful* iff $req_i = pol_i$, that is, the provider's requests match the actual minimal requirements for secure and effective service delivery.

Finally, each agent i is associated to a preference relation $\preceq_i$. We assume that $\preceq_0$ is $\preceq$ (that is, the user's goal is minimizing the sensitivity of disclosed information), while for all providers i, $\preceq_i$ can be either $\preceq$ or $\subseteq$. In the former case, i's goal is maximizing the sensitivity of the information acquired from users,[1] while in the latter case i's goal is maximizing the amount of such information. Let $\vec{\preceq}$ be the vector $\langle \preceq_0, \ldots, \preceq_N \rangle$.

Now a *(full) profile* is a triple $\pi = \langle \mathbf{pol}, \mathbf{req}, \vec{\preceq} \rangle$ where **pol** is a policy profile and **req** a strategy profile for **pol**. The set of all full profiles will be denoted by Π.

2.1 Selection and Response Mechanism

We need preliminary definitions. The set of optimal admissible requests in a profile π is:

$$opt(\pi) = \min_{\preceq} \left(\bigcup_{j=1}^{N} req_j \cap adm(pol_0) \right),$$

where $\min_{\preceq}(X)$ denotes the set of minimal elements of X according to $\preceq$, that is, $\min_{\preceq}(X) = \{r \in X \mid \forall r' \in X, r' \nprec r\}$.

The user prefers those providers that make minimal information requests. Formally, let the provider i be a *candidate winner* in π iff $opt(\pi) \cap req_i \neq \emptyset$. The set of candidate winners is denoted by $cw(\pi)$.

[1] The rationale is that information value is often correlated with sensitivity. For simplicity, in this first paper we assume a shared (objective) measure of sensitivity – e.g. based on statistics about the identification power of attribute aggregates, cf. quasi-identifiers [7] – so that $\preceq$ may be regarded as common knowledge. Generalizations are discussed in Sec. 4.

Each candidate winner i is associated to a set of possible *responses*, $res(\pi, i)$. Possible responses are credential sets that must satisfy both the user's policy and the provider's request, that is, for all $r \in res(\pi, i)$, $r \in adm(pol_0) \cap ful(req_i)$. Different specific definitions of $res(\pi, i)$ yield different properties in terms of robustness (i.e., lack of unnecessary transaction failures) and amount of information released. Before discussing the alternatives, let us fix the decision making process (provider selection and response selection), consisting of two steps:

1. choose a provider $i \in cw(\pi)$
2. choose a response $r \in res(\pi, i)$.

If $res(\pi, i) = \emptyset$, then the transaction fails. To simplify the discussion, let us assume that the above choices are made at random with uniform probability (different distributions can be adopted, though).

So far, the framework is similar to an auction with some extra constraints posed by policies. In Vickrey's auctions, truthfulness is achieved by setting the price to the second best offer. In this framework, a direct counterpart of this idea consists in defining $res(\pi, i)$ as the set of best requests made by all the providers $j \neq i$. In order to formalize this idea we need a few more auxiliary definitions. First, let $opt_{-i}(\pi)$ denote the best admissible requests of the providers $j \neq i$:

$$opt_{-i}(\pi) = \min_{\leq}\left(\bigcup_{1\leq j\leq N}^{j\neq i} req_j \cap adm(pol_0)\right).$$

Then, let $res_0(\pi, i)$ be the set of all the best admissible requests of the providers $j \neq i$ that satisfy i's request, that is: $res_0(\pi, i) = opt_{-i}(\pi) \cap ful(req_i)$.

By setting $res(\pi, i) = res_0(\pi, i)$ one obtains a mechanism that easily leads to failures, because the requests of the winners might not fulfill each other.

Example 3. Suppose that the user's policy permits the simultaneous disclosure of her credit card number (*card_num*), its security code (*sec_code*), user name (*name*), and birth date (*birth_date*), i.e., $pol_0 = \{\{card\_num, sec\_code, name, birth\_date\}\}$. Let the requests of providers 1 and 2 be $req_1 = \{\{card\_ num, name, sec\_code\}\}$ and $req_2 = \{\{card\_num, name, birth\_date\}\}$, respectively. If req_1 and req_2 are not comparable with respect to $\prec$, then both requests are $\leq$-minimal and admissible, so the set of candidate winners is $cw(\pi) = \{1, 2\}$. However, the request of provider 1 does not fulfill the request of provider 2 and viceversa. Then $res_0(\pi, 1) = res_0(\pi, 2) = \emptyset$ and the transaction fails no matter which of the two providers is chosen. □

This problem can be mitigated by adding to the pool of replies the largest admissible requests, that is, the members of pol_0. Let

$$res_1(\pi, i) = \min_{\leq}(opt_{-i}(\pi) \cup pol_0) \cap ful(req_i),$$
$$res_2(\pi, i) = (opt_{-i}(\pi) \cup pol_0) \cap ful(req_i).$$

The first definition (res_1) still leads to a failures; for instance, in Example 3, it is equivalent to res_0 because $\min_{\leq}(opt_{-i}(\pi) \cup pol_0) = opt_{-i}(\pi)$, for $i = 1, 2$. The second definition

(res_2) does not lead to any failure in Example 3; however, the user has to disclose all releasable credentials (*card_num*, *sec_code*, *name*, *birth_date*). In general, $res_2(\pi, i)$ contains (at least) all the elements of pol_0 that cover some request of i, therefore some maximal disclosable set of credentials can always be released with probability greater than 0. Then we move over to a more parsimonious definition (in terms of maximal disclosures). The idea consists in "interpolating" intermediate requests between the optimal requests of all $j \neq i$. Such interpolation constitutes a "vault" above req_i, from which possible responses can be selected. Formally, let

$$vault(\pi, i) = \max_{\subseteq}\{r \subseteq C \mid r \in adm(pol_0) \wedge \forall r' \in opt_{-i}(\pi).\ r' \not\preceq r\}\,.$$

In Example 3, assuming for simplicity that $\preceq$ equals $\subseteq$, the vaults are

$$\begin{aligned} vault(\pi, 1) = vault(\pi, 2) = \\ &\{\,\{card\_num, name, sec\_code\}, \{sec\_code, name, birth\_date\}, \\ &\{card\_num, sec\_code, birth\_date\}, \{card\_num, name, birth\_date\}\,\}\,. \end{aligned}$$

They contain the providers' requests, as well as elements that do not fulfill them. Therefore responses should consist of the vault elements that cover some optimal request of i:

$$res(\pi, i) = vault(\pi, i) \cap ful(opt(\pi) \cap req_i)\,.$$

Compare this definition with the standard second price approach: There, the winner pays the minimum price that is not worse (i.e., smaller) than any other offer; analogously, in our framework, the winner gets a maximal response that is not worse (i.e., more sensitive) than any other offer, and satisfies both the user's policy and the winner's request.

We are going to study in depth the framework based on this definition. Before starting its formal analysis, note that in the case of Example 3, $res(\pi, i) = req_i$ ($i = 1, 2$), that is, the selected provider receives nothing more than what it asked for (as opposed to what happens with res_2). In general, however, the user may have to release more than what the winning provider asks for. In general, this is the price to pay for truthfulness. At the end of this section we will characterize a wide class of scenarios in which no unnecessary information is disclosed.

The first formal property of the definition based on vaults concerns its robustness: Unlike res_0 and res_1, it introduces no unnecessary failures. In other words, whenever some request is admissible (equivalently, $cw(\pi) \neq \emptyset$), all candidate winners can be given a response:

Theorem 1. *If there exist a provider j and a request $r \in req_j$ such that $r \in adm(pol_0)$, then for all $i \in cw(\pi)$, $res(\pi, i) \neq \emptyset$.*

It is easy to verify that a similar property holds for res_2. So the second formal property of interest concerns a comparison of the two robust strategies *res* and res_2 with respect to the amount of credentials potentially released. It can be shown that *res* is generally more parsimonious than res_2:

Theorem 2. *For all $r \in res(\pi, i)$ there exists $r' \in res_2(\pi, i)$ such that $r \subseteq r'$.*

Next we investigate the effectiveness of the provider selection mechanism in reducing the amount of information disclosed in the worst case. A first question related to this issue is: Under which circumstances can a maximal releasable set of credentials (i.e., a member of pol_0) be disclosed?

Theorem 3. *Let $r \in pol_0$, $r \in res(\pi, i)$ if and only if there exists $x \in (opt(\pi) \cap req_i)$ such that $x \subseteq r$ and for all the other providers $j \neq i$ and for all $r' \in req_j$, it holds $r' \not< r$.*

Note that if $r' \subseteq r$, then $r' \not< r$ implies $r' = r$. Then Theorem 3 says that r can be released to i if either there is no competition within the option r (i.e., the other providers' requests are not compatible with r), or the competitors ask exactly for r. Consequently, it appears that competition makes maximal disclosures more difficult to achieve systematically, at least in the absence of detailed information about the user's policy.

Another interesting, related problem is characterizing the circumstances under which the user may have to release all disclosable credentials at once (as it may happen in the formal trust negotiation frameworks studied in the literature).

Corollary 1. *Assume that i makes an admissible request ($req_i \cap adm(pol_0) \neq \emptyset$). Then $\bigcup_{r \in pol_0} r$ can be disclosed to provider i iff the following conditions hold:* (i) $pol_0 = \{r\}$, *and* (ii) $r \in req_j$ *for all providers $j \neq i$ such that $req_j \cap adm(pol_0) \neq \emptyset$.*

According to the first condition in the above corollary, if $|pol_0| > 1$, then it is impossible to release $\bigcup_{r \in pol_0} r$. The reason is clear: whenever $|pol_0| > 1$, the user's policy encodes some integrity constraints that forbid the disclosure of arbitrary unions of disclosable credentials. For example, if $pol_0 = \{\{birthday\}, \{address\}\}$ then both birth date and address can be separately disclosed, but their union is considered too sensitive to be released. Now suppose that $|pol_0| = 1$. Condition 2 says that either i has no competitors (no other provider j makes any admissible request), or i's competitors all ask for r (which is unlikely in practice unless pol_0 is public). This shows how competition helps in reducing complete credential disclosures.

Example 4. Suppose that a user is willing to execute payments either by using her credit card or by bank transfer. In the first case she permits the simultaneous disclosure of her credit card number (*card_num*) and its security code (*sec_code*). For the latter payment form, she is willing to provide the unique ID (*id*) associated to the bank transfer and her own bank account information (*acc_info*). Formally $pol_0 = \{\{card\_num, sec\_code\}, \{id, acc\_info\}\}$. The user can select among three providers for executing a given payment whose requests are, respectively: $req_1 = \{\{card\_num\}\}$, $req_2 = \{\{card\_num\}, \{id\}\}$, $req_3 = \{\{card\_num, sec\_code\}\}$. In such context, there is clear competition among all servers since everyone allows credit card payments. Server 1 and 2 only require *card_num* while server 3 requires both *card_num* and *sec_code*. So, in a parsimonious selection, the user would prefer server 1 or 2. For the second payment method there is no competition at all. Indeed only server 2 allows bank transfers and requires the unique transaction identifier in order to accept the payment. Let π be a profile describing such a scenario. The set of optimal admissible requests is $opt(\pi) = \min_{\subseteq}(\bigcup_{j=1}^{N} req_j \cap adm(pol_0)) = \{\{card\_num\}, \{id\}\}$. Then the set of candidate winners is $cw(\pi) = \{1, 2\}$. Let us focus on server 1. The set of optimal admissible requests made by all providers except 1 is $opt_{-1}(\pi) = \{\{card\_num\}, \{id\}\}$. Then the vault for provider 1—i.e., the maximal subsets of credentials that are admissible for the client and do not cover optimal requests made by other servers—is $vault(\pi, 1) = \{\{card\_num\}, \{sec\_code\}, \{id\}, \{acc\_info\}\}$. Note that the vault contains no elements of pol_0, because provider 1 competes with provider 2 (whose request $\{card\_num\}$ expunges $\{card\_num, sec\_code\}$ from the vault). Finally, the set of possible responses for server 1 are the members of $vault(\pi, 1)$ that satisfy the server's requests, i.e., $res(\pi, 1) = \{\{card\_num\}\}$. Similarly, for provider 2 we have $opt_{-2}(\pi) = \{\{card\_num\}\}$, $vault(\pi, 2) = \{\{card\_num\}, \{sec\_code\}, \{id, acc\_info\}\}$, and $res(\pi, 2) = \{\{card\_num\}, \{id, acc\_info\}\}$.

Thus, when different servers compete within a specific element of pol_0, (here, $\{card\_num, sec\_code\}$), such element is not entirely disclosed to any server. On the other hand, if exactly one server makes a request compatible with some element in $r \in pol_0$, i.e., if there is no competition within r ($r = \{id, acc\_info\}$ in the example), then r is fully disclosed with nonzero probability. □

2.2 Rational Strategies

We are left to characterize the strategies adopted by ideally rational providers. We consider two kinds of providers: Providers of the first kind are mainly interested in attracting new customers, i.e. their primary goal is maximizing the probability of being selected (or probability of winning) $pw(\pi, i)$, where $pw(\pi, i) = 1/|cw(\pi)|$ if $i \in cw(\pi)$, and $pw(\pi, i) = 0$ otherwise. As a secondary goal, these providers prefer those strategies that better meet their preference $\preceq_i$. Providers of the second kind invert the above priorities. A new player in a given application domain is likely to be a player of the first kind. Similarly, providers whose main income is based on advertisement are likely to be providers of the first kind. On the contrary, when the utility of service usage is dominated by the value of user profiles, providers should be expected to be agents of the second kind.

In order to formalize the perfect strategies for these providers we need some auxiliary notions. First, one needs to compare different responses w.r.t. provider preferences. For this purpose, $\preceq_i$ should be extended from credential sets to *sets* of credential sets (i.e., the range of *res*).

Definition 1. *For all $\rho, \rho' \subset \mathscr{P}(C)$, let $\rho \preceq_i^* \rho'$ iff*

1. *for all $r \in \rho$, there exists $r' \in \rho'$ such that $r \preceq_i r'$, and*
2. *for all $r' \in \rho'$ and $r \in \rho$, $r' \nprec_i r$.*

In other words ρ' is preferable to ρ if for all possible responses in ρ there exists an equally preferred or better response in ρ' (according to i's preferences) and none of the responses in ρ' is less preferable than any response in ρ.

Next, we need a handy way of replacing the strategy of an agent: For all strategy profiles **req** and all providers $1 \le i \le N$, let

$$\mathbf{req}[i \leftarrow req'] = \langle req_1, \ldots, req_{i-1}, req', req_{i+1}, \ldots, req_N \rangle ,$$

and for all profiles $\pi \in \Pi$, let

$$\pi[i \leftarrow req'] = \langle \mathbf{pol}, \mathbf{req}[i \leftarrow req'], \vec{\preceq} \rangle .$$

Finally, let $pol \in \Theta$ be an arbitrary but fixed policy. The set of profiles where $pol_i = pol$ is denoted by Π^i_{pol}. A *strategy for pol* is any $req \in \Theta$ such that for all $r \in req$, $r \in ful(pol)$. Now the optimal strategies for the two kinds of agents can be formalized.

Definition 2. *A strategy req for pol is a* dominant attraction strategy for *i* with respect to *pol iff for all* $\pi \in \Pi^i_{pol}$,

1. $pw(\pi, i) \leq pw(\pi[i \leftarrow req], i)$*, and*
2. *if* $pw(\pi, i) = pw(\pi[i \leftarrow req], i)$ *then* $res(\pi, i) \preceq^*_i res(\pi[i \leftarrow req], i)$.

Definition 3. *A strategy req for pol is a* dominant investigation strategy for *i* with respect to *pol iff for all* $\pi \in \Pi^i_{pol}$,

1. $res(\pi, i) \preceq^*_i res(\pi[i \leftarrow req], i)$*, and*
2. *if* $res(\pi[i \leftarrow req], i) = res(\pi, i)$ *then* $pw(\pi, i) \leq pw(\pi[i \leftarrow req], i)$.

A few explanations are in order here. The universal quantification over Π^i_{pol}, whose only invariant is $pol_i = pol$, means that strategy *req* is optimal w.r.t. all the other possible strategies req_i that might be adopted by *i*, and this holds in all possible contexts (i.e., no matter what the policies and strategies of the other agents are). Our mechanism yields the desired result: the truthful strategy $req = pol$ is the best strategy a provider can adopt, under both priorities.

Theorem 4. *For all pol* $\in \Theta$ *and all providers i, the unique dominant attraction strategy for i w.r.t. pol is pol itself.*

Theorem 5. *For all pol* $\in \Theta$ *and all providers i, the unique dominant investigation strategy for i w.r.t. pol is pol itself.*

In game-theoretic terms, the previous two theorems prove that being truthful is a *dominant strategy equilibrium* (DSE) [13], i.e., no matter what the other agents' policies and strategies are, being truthful is always the best response. Every DSE is in particular a Nash equilibrium.

One may wonder whether gaining information on the behavior of the other agents could allow a provider to increase either its winning probability or the amount of credentials received from the client. The answer is negative, regardless of the extra information available to the provider. Indeed, any gained information corresponds to restricting the set of possible profiles Π^i_{pol} in the definition of dominant attraction (resp., investigation) strategy (Definition 2 and 3, respectively). Clearly, by applying any such restriction, the set of dominant attraction (resp., investigation) strategies may only increase. However, since any pair of dominant strategies dominate each other, it is straightforward to see from Definition 2 and 3 that all dominant strategies give rise to the same probability of winning and the same response from the client.

The presence of rational (and hence truthful) providers may induce minimal disclosures in a framework that, in general, releases to providers more information than what they ask for. For simplicity, we analyze this issue in scenarios where all providers have the same policy. In practice, this assumption is naturally satisfied when provider policies are determined by the same technological constraints—for example, all providers supporting VISA credit card payments must provide VISA's servers with the same information for credit card validation (as in Example 6 below).

Theorem 6. *If all providers have the same policy and there are two truthful providers i and j, then $res(\pi, i) = res(\pi, j) = req_i \cap adm(pol_0) \subseteq pol_i$.*

In informal terms, the above theorem ensures that under the uniform policy hypothesis, rational servers are given only elements of their policy, that is, some of the minimal possible credential sets that grant access to the service. Consider the following scenario, for example. It is inspired by real flight reservation portals. Kayak and Momondo ask for no information; tickets are purchased directly from airline companies. On the contrary, eDreams asks for a rich user profile that is then used to make a request to airline companies. Note that eDream user profiles comprise attributes that are not mandatory for airline companies. The following is a formalization of this scenario:

Example 5. Assume that $pol_1 = pol_2 = pol_3 = \{\emptyset\}$, $req_1 = req_2 = \{\emptyset\}$, and $req_3 = \{\{name, address, phone\_num, email\}\}$. Note that providers 1 and 2 are truthful and provider 3 is not. Clearly, only providers 1 and 2 are candidate winners. The response is $\{\emptyset\}$, that is, the user releases no information. Note that the same result is obtained when $pol_3 \neq \{\emptyset\}$ (e.g., $pol_3 = req_3$); thus, in future work, it may be interesting to relax the hypothesis of Theorem 6. □

Moreover, if all policies are the same and there is at least one rational (truthful) server, then the other servers cannot receive more information than what the policy requires.

Theorem 7. *If all providers have the same policy and provider i is truthful, then for all $j \neq i$, $res(\pi, j) \subseteq pol_j$.*

Example 6. Consider an e-commerce application with the same credentials as in Example 3. Assume that all vendors use the same underlying financial institution that requests the credit card number and either the owner's name or the credit card security code, so the providers all share the same policy $pol_i = \{\{card\_num, name\}, \{card\_num, sec\_code\}\}$. Suppose that server i is truthful, while j has strategy $req_j = \{\{card\_num, name, birth\_date\}, \{card\_num, sec\_code\}\}$, i.e., in addition to the credit card number and owner's name, j requests also her birth date. The policy of the client is $pol_0 = \{\{card\_num, name, sec\_code, birth\_date\}\}$, i.e., all credentials are simultaneously releasable. Now j is a candidate winner (as it makes the request $\{card\_num, sec\_code\}$). However, even if j is selected to deliver the service, in accordance with Theorem 7 it will *not* receive the user's birth date, even if it is releasable by the client. Indeed, we have $opt_{-j}(\pi) = \{\{card\_num, name\}, \{card\_num, sec\_code\}\}$. This means that $vault(\pi, j)$ consists of $\{card\_num, name$, $\{card\_num, sec\_code\}$, $\{card\_num, birth\_date\}$ and $\{name, sec\_code, birth\_date\}$. Finally, we have that $res(\pi, j) = \{\{card\_num, sec\_code\}\}$. □

3 Related Work

To the best of our knowledge, the approach introduced in the above sections has no analogue in the literature. Standard access control frameworks place no constraints on policies and set up no mechanisms for reducing the extension and sensitivity of user profiles. In the trust negotiation area, privacy is mainly pursued by having disclosed information match as precisely as possible the provider's request, see for example [18,14,10]. Another work aims at preventing providers from inferring which *types* of credentials are owned by the user when a transaction fails [21]. A major difference between our approach and theirs is that those works do not attempt to influence the providers' requests. There, providers' policies do not represent minimal functional and security requirements (unlike our pol_is); they should rather be considered as part of the specification of req_i. Not only can trust negotiation policies ask for unnecessary information; some interoperable negotiation strategies may further inflate requests by forcing the user agent to disclose all releasable credentials in the attempt to keep the negotiation alive [22,2]. Therefore, the issue of minimizing the difference between req_i and pol_i is not tackled (and there is no precise counterpart of pol_i).

In [8], privacy is tackled in the context of auctions (including second-price auctions). Their purpose is the opposite of ours, namely, minimizing the amount of information that *bidders* have to disclose in order to let the auctioneer compute the optimal outcomes. Payments are fully traditional: continuous and totally ordered. Similarly, another economically inspired model [11] proposes both an estimate of the value of private information and a fair compensation for such information release that may induce *users* to release richer and correct information about their personal preferences.

Concerning the many auction models introduced in the past, the main technical difference is that utility functions and bids always range over a totally ordered domain such as the set of real numbers. Then truthfulness can be obtained with a straightforward implementation of the second-price idea, without resorting to more complex notions such as vaults.

4 Discussion and Perspectives

4.1 Current Achievements

We provided a first formal evidence that the potential competition between equivalent applications may enhance privacy and reduce the amount of personal information requested and collected by application providers. As a starting point, we focussed on scenarios where the competing services are equivalent with respect to functionalities and quality, and the only cost for the user consists of the personal information released. Flight reservation applications provide concrete examples of such scenarios, cf. Example 5 and the preceding paragraph. We argued that one-shot mechanisms (like second-price auctions) are appropriate for such scenarios; indeed, after using the service a first time, subsequent usage is "free" from the point of view of information disclosure; therefore, re-using the same service is always an optimal choice for the user until either demand or offer change.

We proved that a suitable one-shot mechanism regulating provider selection and information disclosure induces truthful behavior in selfish rational providers, resulting in minimal information requests and in a wider range of choices for the user. It is important to note that essentially these results still hold even if agents know each other's policies and strategies.

When functional and security requirements are the same for all providers (e.g., because such requirements are determined by exogenous technological constraints), two rational (and hence truthful) providers suffice to minimize the amount of user information disclosed to *any* provider—and interestingly, if only one provider is rational, then it is the only provider that may receive a non-minimal response. In some cases, the hypothesis that all policies are the same can be relaxed, as shown in Example 5. Another set of results shows that extracting all releasable credentials (or any maximal disclosable set of credentials) from a user is a difficult task, whose systematic achievement, in practice, requires some knowledge of the user's policy.

Competition may be exploited to address a weakness of automated disclosure techniques, such as OpenID profile sharing and automated trust negotiation. These approaches require a policy to decide when a user profile can be automatically transferred to a new web service, or when an information request is reasonable in a given context. Formalizing a policy that decides on behalf of the user whether a provider is collecting a reasonable set of user attributes is a very difficult task. A mechanism inducing a spontaneous moderation of provider requests may turn out to be less expensive and/or more reliable.

Concerning trust negotiation, it is known that interoperable negotiation strategies are vulnerable to attacks that force a peer to release progressively all disclosable information (cf. [22,2]). It seems that the current one-shot mechanism can address this problem: First, the user agent negotiates with all the providers' agents using a method that does not actually disclose credentials until the end of the negotiation (as in the protocol introduced in [21]). At the end of the negotiations, before really sending credentials, the user can choose among the successful negotiations those with minimal information disclosure, and compute the response with our mechanism, thereby inducing rational providers to reduce information requests.

4.2 Possible Variations to the Current Framework

Several aspects of the framework can be modified without affecting our results. For example, it is not hard to see that different probability distribution can be used in choosing winner and response. As an example of possible applications, skewed distributions over candidate winners may address additional user preferences over providers. Similarly, the preference relations $\preceq_i^*$ employed to compare responses and define dominant strategies can be modified in various ways, in order to model providers with different risk attitudes. For instance, an optimistic (resp. pessimistic) provider may consider only the $\preceq_i$-maximal (resp. $\preceq_i$-minimal) elements of $res(\pi, i)$ (the current definition considers all of $res(\pi, i)$). Our results hold for all such agents.

4.3 Generalizing Preferences

The current theoretical framework should be extended and complemented along several directions. In general, providers compete also on properties such as cost and quality of service. They can easily be modelled by extending credential sets to richer sets, including items that represent the additional quantities of interest. This affects some of the assumptions we adopt in this paper.

For example, in the extended framework, preference relations rank aggregates of credentials, money, and quality of service, thereby reflecting a tradeoff between privacy-related costs and other costs and benefits; consequently, user preferences have a more "personal" nature and it would not be reasonable to make the simplifying assumption that $\preceq$ is based on an objective, shared sensitivity measure. As a consequence, provider preferences could not be restricted to $\subseteq$ and $\preceq$, and the effects of this generalization should be formally analyzed. In this context, users may publish their preference relation to get personalized offers; it should be verified whether a rational user should be truthful, and whether preference disclosure may constitute a privacy violation in itself.

As preference specifications become more sophisticated, the need may arise for usability-enhancing techniques. For instance, a coarse-grained preference relation in a pure information disclosure scenario (where service re-use has no additional "costs") may induce the user to always select the same provider (lock-in effect). The articulated approach to monitoring, refining, and learning preferences introduced in [15] for access control policies can perhaps be adapted to our framework.

4.4 Repeated Auctions

Another consequence of modelling features such as money and quality is that each call to a same service may have additional costs; then it makes sense to repeat the service selection process and abandon the one-shot approach. We conjecture that, in general, a sequence of independent selections may lead to globally suboptimal disclosures (as any greedy strategy). A formal analysis of iterative selections is an important step in our agenda, that may start from the literature on repeated (or sequential) procurement auctions, e.g. [12,1]. It is also interesting to evaluate "globalized" selections over bundles of services as an attempt to optimize information disclosure for a set of commonly used services. It is known that this optimization problem is tractable in some cases [4].

4.5 Deployment

Last but not least, the need is felt for an articulated analysis of deployment solutions; for example, providers may want to make sure that the service selection mechanism is carried out correctly, i.e. users are not cheating and actually disclose an element of the vault. Some application contexts may admit trusted intermediary services. Portals similar to Kayak, Momondo etc. are interesting candidates to fill in this role, that may create new business opportunities and models. In the absence of trusted third parties (i.e., the user is the auctioneer), auctions can be implemented using Secure Multiparty Computation. The instantiation of general MPC constructions can be inefficient; it is, however, possible to design less complex specialized MPC protocols implementing the described variation of second price auctions, cf. [3]. Alternatively, one can resort to ad hoc protocols where credential requests are eventually revealed to all providers (*commitments* and *blind signature* [6] may be employed for this purpose).

Acknowledgments. The authors would like to thank Alessandro Bonatti for helpful preliminary discussions, and Adam J. Lee and the anonymous referees for their detailed and stimulating feedback.

References

1. Bae, J., Beigman, E., Berry, R.A., Honig, M.L., Vohra, R.V.: Sequential bandwidth and power auctions for distributed spectrum sharing. IEEE Journal on Selected Areas in Communications 26(7), 1193–1203 (2008)
2. Baselice, S., Bonatti, P., Faella, M.: On interoperable trust negotiation strategies. In: IEEE POLICY 2007, pp. 39–50. IEEE Computer Society, Los Alamitos (2007)
3. Bogetoft, P., Damgård, I., Jakobsen, T.P., Nielsen, K., Pagter, J., Toft, T.: A practical implementation of secure auctions based on multiparty integer computation. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006. LNCS, vol. 4107, pp. 142–147. Springer, Heidelberg (2006)
4. Bonatti, P.A., Festa, P.: On optimal service selection. In: Ellis, A., Hagino, T. (eds.) Proc. of the 14th Int. Conf. on World Wide Web, WWW 2005, pp. 530–538. ACM, New York (2005)
5. Broache, A.: Competition is good for search privacy, report says. CNET News (August 8, 2007), `http://news.cnet.com/Competition-is-good-for-search-privacy,-report-says/2100-1029_3-6201468.html`
6. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology - Crypto 1982, pp. 199–203. Springer, Heidelberg (1983)
7. Dalenius, T.: Finding a needle in a haystack - or identifying anonymous census records. Journal of Official Statistics 2(3), 329–336 (1986)
8. Feigenbaum, J., Jaggard, A.D., Schapira, M.: Approximate privacy: foundations and quantification (extended abstract). In: Parkes, D.C., Dellarocas, C., Tennenholtz, M. (eds.) ACM Conference on Electronic Commerce, pp. 167–178. ACM, New York (2010)
9. Gray, E.: FTC to boost competition in privacy protection. Global Competition Review (September 23, 2010)
10. He, Y., Zhu, M., Zheng, C.: An efficient and minimum sensitivity cost negotiation strategy in automated trust negotiation. In: Int. Conf. Comp. Sci. and Soft. Eng.,, vol. 3, pp. 182–185 (2008)
11. Kleinberg, J., Papadimitriou, C.H., Raghavan, P.: On the value of private information. In: TARK 2001: Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge, pp. 249–257. Morgan Kaufmann, San Francisco (2001)
12. Luton, R., McAfee, P.R.: Sequential procurement auctions. Journal of Public Economics 31(2), 181–195 (1986)
13. Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
14. Paci, F., Bauer, D., Bertino, E., Blough, D.M., Squicciarini, A.C.: Minimal credential disclosure in trust negotiations. In: Bertino, E., Takahashi, K. (eds.) Digital Identity Management, pp. 89–96. ACM, New York (2008)
15. Sadeh, N.M., Hong, J.I., Cranor, L.F., Fette, I., Kelley, P.G., Prabaker, M.K., Rao, J.: Understanding and capturing people's privacy policies in a mobile social networking application. Personal and Ubiquitous Computing 13(6), 401–412 (2009)
16. Samarati, P.: Protecting respondents' identities in microdata release. IEEE Transactions on Knowledge and Data Engineering 13(6), 1010–1027 (2001)
17. Schwartz, A., Cooper, A.: Search privacy practices: A work in progress. Center for Democracy and Technology report (August 2007)
18. Squicciarini, A.C., Bertino, E., Ferrari, E., Paci, F., Thuraisingham, B.M.: PP-trust-X: A system for privacy preserving trust negotiations. ACM Trans. Inf. Syst. Secur. 10(3) (2007)
19. Sweeney, L.: Guaranteeing anonymity when sharing medical data, the Datafly system. Journal of the American Medical Informatics Association (1997)
20. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. Journal of Finance 16, 8–37 (1961)

21. Winsborough, W.H., Li, N.: Protecting sensitive attributes in automated trust negotiation. In: WPES, pp. 41–51. ACM, New York (2002)

22. Yu, T., Winslett, M., Seamons, K.E.: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM Trans. Inf. Syst. Secur. 6(1), 1–42 (2003)

Proofs

Theorem 2. *For all $r \in res(\pi, i)$ there exists $r' \in res_2(\pi, i)$ such that $r \subseteq r'$.*

Proof. Assume $r \in res(\pi, i)$. By definition $r \in vault(\pi, i)$ and $r \in ful(opt(\pi) \cap req_i)$; therefore, $r \in adm(pol_0)$ and $r \in ful(req_i)$. From $r \in adm(pol_0)$, it follows that there exists $r' \in pol_0$ such that $r \subseteq r'$. Then from $r \in ful(req_i)$, derive $r' \in ful(req_i)$, and hence $r' \in pol_0 \cap ful(req_i)$. The theorem follows immediately from the definition of $res_2(\pi, i)$. □

Theorem 3. *Let $r \in pol_0$, $r \in res(\pi, i)$ if and only if there exists $x \in (opt(\pi) \cap req_i)$ such that $x \subseteq r$ and for all the other providers $j \neq i$ and for all $r' \in req_j$, it holds $r' \nprec r$.*

Proof. (If) Note that since for all $r' \in req_j$, with $j \neq i$, $r' \nprec r$, *a fortiori* for all $r' \in opt_{-i}(\pi)$, $r' \nprec r$. Then, the thesis follows by applying the definition of $vault(\pi, i)$ and $res(\pi, i)$.

(Only if) By definition, if $r \in res(\pi, i)$, then $r \in vault(\pi, i)$ and there exists a request x such that $x \subseteq r$ and $x \in (opt(\pi) \cap req_i)$. Assume now that for some provider $j \neq i$ and some request $r' \in req_j$, $r' \prec r$. On one hand, there exists in $opt_{-i}(\pi)$ a request r'' such that $r'' \preceq r'$ and hence $r'' \prec r$. On the other hand, since $r \in vault(\pi, i)$, for all $r' \in opt_{-i}(\pi)$, $r' \nprec r$ *(absurdum).* □

Corollary 1. *Assume that i makes an admissible request ($req_i \cap adm(pol_0) \neq \emptyset$). Then $\bigcup_{r \in pol_0} r$ can be disclosed to provider i iff the following conditions hold:* (i) $pol_0 = \{r\}$, *and* (ii) $r \in req_j$ *for all providers $j \neq i$ such that $req_j \cap adm(pol_0) \neq \emptyset$.*

Proof. Since the client releases exactly one element in pol_0, the only way in which the client could release $\bigcup_{r \in pol_0} r$ is that pol_0 contains exactly one set r. Then, the proof follows easily from Theorem 3. □

In the following results, we need two auxiliary relations: for all $\theta, \theta' \in \Theta$ let $\theta \subseteqq \theta'$ iff $\forall r' \in \theta' \; \exists r \in \theta : r \subseteq r'$. Note that for all providers i, it holds $pol_i \subseteqq req_i$. Similarly, let $\theta \leqq \theta'$ iff $\forall r' \in \theta' \; \exists r \in \theta : r \preceq r'$.

Lemma 1. *For all $r \in req_i \cap opt(\pi)$ there exists $r' \subseteq r$ such that $r' \in pol_i \cap opt(\pi[i \leftarrow pol_i])$.*

Proof. Let $r \in req_i \cap opt(\pi)$. Since $pol_i \subseteqq req_i$, there exists $r' \in pol_i$ s.t. $r' \subseteq r$; moreover $r \in opt(\pi)$ implies $r \in adm(pol_0)$ and then $r' \in adm(pol_0)$. Assume per adsurdum that there exists $r'' \in opt(\pi[i \leftarrow pol_i])$ s.t. $r'' \prec r'$. Clearly, r'' cannot belong to pol_i because pol_i is by definition a threshold; therefore there exists a provider $j \neq i$ such that $r'' \in req_j \cap adm(pol_0)$, but in this case, as req_j is the same in both π and $\pi[i \leftarrow pol_i]$, we would have $r'' \prec r$ and hence $r \notin opt(\pi)$ against the hypothesis. Therefore, $r' \in opt(\pi[i \leftarrow pol_i])$. □

Corollary 2. *For all* $\pi \in \Pi$ *and providers i, we have* $opt(\pi[i \leftarrow pol_i]) \leqq opt(\pi)$.

Corollary 3. *For all* $\pi \in \Pi$, *if* $i \in cw(\pi)$ *then* $i \in cw(\pi[i \leftarrow pol_i])$.

Lemma 2. *For all* $\pi \in \Pi$, *if* $i \in cw(\pi)$ *then* $cw(\pi[i \leftarrow pol_i]) \subseteq cw(\pi)$.

Proof. It suffices to show that for all servers $j \neq i$, $j \in cw(\pi[i \leftarrow pol_i])$ implies $j \in cw(\pi)$. Let r be in $req_j \cap opt(\pi[i \leftarrow pol_i])$. By definition, $r \in adm(pol_0)$. Furthermore, because of *opt* minimality, for all $r' \in opt(\pi[i \leftarrow pol_i])$, we have $r' \nprec r$. Since $opt(\pi[i \leftarrow pol_i]) \leqq opt(\pi)$ (Corollary 2), for all $r'' \in opt(\pi)$, $r'' \nprec r$. Therefore, $r \in req_j \cap opt(\pi)$. □

Lemma 3. *For all* $\pi \in \Pi$ *and providers i, it holds that* $res(\pi, i) \subseteq res(\pi[i \leftarrow pol_i], i)$.

Proof. $r \in res(\pi, i)$ implies that $r \in vault(\pi, i)$ and there exists $r' \in opt(\pi) \cap req_i$ s.t. $r' \subseteq r$. Note that $vault(\pi, i)$ does not depend on the request of i, therefore $vault(\pi, i) = vault(\pi[i \leftarrow pol_i], i)$. Moreover, due to Lemma 1, there exists r'' s.t. $r'' \subseteq r'$ and $r'' \in opt(\pi[i \leftarrow pol_i]) \cap pol_i$. Therefore, $r \in vault(\pi[i \leftarrow pol_i], i) \cap ful(opt(\pi[i \leftarrow pol_i]) \cap pol_i)$, i.e. $r \in res(\pi[i \leftarrow pol_i], i)$. □

Lemma 4. *For all* $\pi \in \Pi$ *and providers i it holds that* $res(\pi, i) \preceq_i res(\pi[i \leftarrow pol_i], i)$.

Proof. If $res(\pi, i) \neq res(\pi[i \leftarrow pol_i], i)$, we need to show that both conditions in Definition 1 are met. By Lemma 3, $res(\pi, i) \subseteq res(\pi[i \leftarrow pol_i], i)$. Condition 1 follows by reflexivity of relation $\preceq_i \in \{\preceq, \subseteq\}$. Indeed, for every $r \in res(\pi, i)$, there exists $r' = r \in res(\pi[i \leftarrow pol_i], i)$ such that $r \preceq_i r'$.

As for condition 2, we observe that $res(\pi[i \leftarrow pol_i], i)$ is a set of maximal elements w.r.t. $\preceq$. This means that every pair of elements in $res(\pi[i \leftarrow pol_i], i)$ are incomparable w.r.t. $\preceq$, and hence w.r.t. $\subseteq$. In particular, for every $r' \in res(\pi[i \leftarrow pol_i], i)$ and $r \in res(\pi, i) \subseteq res(\pi[i \leftarrow pol_i], i)$, it holds that $r' \nprec r$ and hence $r' \not\subset r$. □

Lemma 5. *If* $opt(\pi) \cap req_i \neq \emptyset$, *then* $res(\pi, i) \neq \emptyset$.

Proof. Let $V = \{r \subseteq C \mid r \in adm(pol_0) \wedge \forall r' \in opt_{-i}(\pi).\ r' \nprec r\}$, so that $vault(\pi, i) = \max_{\subseteq} V$. Let $r \in opt(\pi) \cap req_i$, we prove that $V \neq \emptyset$. Since $r \in opt(\pi)$, we have $r \in adm(pol_0)$. Let $r' \in opt_{-i}(\pi)$; if by contradiction $r' \prec r$, we would have $r \notin opt(\pi)$. Hence, $r \in V$ and $vault(\pi, i) \neq \emptyset$. □

Theorem 4. *For all* $pol \in \Theta$ *and all providers i, the unique dominant attraction strategy for i w.r.t. pol is pol itself.*

Proof. First we prove that $pol_i = pol$ is a dominant attraction strategy (membership), then that for all strategies $req_i \neq pol_i$ there exists a full profile $\pi \in \Pi^i_{pol}$ such that $pw(\pi[i \leftarrow req_i], i) < pw(\pi, i)$, therefore req_i is not a dominant attraction strategy (uniqueness).

(Membership). Let π be a full profile. By the (contrapositive of) Corollary 3 and Lemma 2, it is straighforward to see that $pw(\pi, i) \leq pw(\pi[i \leftarrow pol_i], i)$. From Lemma 4, $res(\pi, i) \preceq_i res(\pi[i \leftarrow pol_i], i)$ always holds, therefore in particular when $pw(\pi, i) = pw(\pi[i \leftarrow pol_i], i)$.

(Uniqueness). Consider $req_i \neq pol_i$, for some $r \in req_i$ and $r' \in pol_i$, $r' \subset r$. Choose $\pi \in \Pi^i_{pol}$ such that *(i)* $pol_0 = \{r'\}$; *(ii)* for two providers $i \neq j$, the requests of i and j in π are $\{r'\}$; *(iii)* for all the other providers $k \neq i, j$, it holds $r' \notin req_k$. Clearly, as $r' \subset r$, $r \notin adm(pol_0)$ and since req_i is a threshold, for the other $r''(\neq r) \in req_i$, $r'' \notin adm(pol_0)$. This implies that $opt(\pi[i \leftarrow req_i]) \cap req_i = \emptyset$ and hence $pw(\pi[i \leftarrow req_i], i) = 0$. On the contrary, due to *(iii)*, $\{r'\} = opt(\pi)$ and hence $pw(\pi, i) > 0$. □

Theorem 5. *For all pol* $\in \Theta$ *and all providers i, the unique dominant investigation strategy for i w.r.t. pol is pol itself.*

Proof. (Membership) By analogy with Theorem 4, membership is a straightforward consequence of Corollary 3 and Lemma 4 and 3.

(Uniqueness). Assume a request req_i for the server i such that $req_i \neq pol_i$ and choose a full profile π as in Theorem 4. Since $opt(\pi[i \leftarrow req_i]) \cap req_i = \emptyset$, then $res(\pi[i \leftarrow req_i], i) = \emptyset$, whereas from $i \in cw(\pi)$ and Lemma 5, $res(\pi, i) \neq \emptyset$. Therefore, $res(\pi[i \leftarrow req_i], i) \prec_i res(\pi, i)$. □

Lemma 6. *For all* $\pi \in \Pi$ *and providers i, we have* $opt_{-i}(\pi) \subseteq vault(\pi, i)$.

Proof. Let $V = \{r \subseteq C \mid r \in adm(pol_0) \wedge \forall r' \in opt_{-i}(\pi).\ r' \nprec r\}$. It holds $vault(\pi, i) = \max_{\subseteq} V$. Let $r \in opt_{-i}(\pi)$. By definition of $opt_{-i}(\pi)$, we have $r \in adm(pol_0)$ and $r' \nprec r$ for all $r' \in opt_{-i}(\pi)$. Hence, $r \in V$.

It remains to prove that r is a maximal element of V. By contradiction, assume that $r' \in V$ is such that $r \subset r'$, which implies $r \prec r'$. Since $r \in opt_{-i}(\pi)$, this leads to the contradiction that $r' \notin V$, and we obtain the thesis. □

Theorem 6. *If all providers have the same policy and there are two truthful providers i and j, then* $res(\pi, i) = res(\pi, j) = req_i \cap adm(pol_0) \subseteq pol_i$.

Proof. First, notice that $opt(\pi) = opt_{-i}(\pi) = req_j \cap adm(pol_0) = req_i \cap adm(pol_0)$. Then,

$$\begin{aligned} res(\pi, i) &= vault(\pi, i) \cap ful(opt(\pi) \cap req_i) \\ &= vault(\pi, i) \cap ful(req_i \cap adm(pol_0)) \\ &= vault(\pi, i) \cap ful(opt_{-i}(\pi)). \end{aligned}$$

Now, if an element r of $vault(\pi, i)$ is included in an element r' of $opt_{-i}(\pi)$, it must be $r = r'$, because $r' \in vault(\pi, i)$ (by Lemma 6) and $vault(\pi, i)$ is a threshold. Therefore, we have $vault(\pi, i) \cap ful(opt_{-i}(\pi)) = opt_{-i}(\pi)$, and the thesis. □

Theorem 7. *If all providers have the same policy and provider i is truthful, then for all* $j \neq i$ *it holds* $res(\pi, j) \subseteq pol_j$.

Proof. Let $j \neq i$. Since server i is truthful, it holds $opt_{-j}(\pi) = opt(\pi) = pol_i \cap adm(pol_0)$. By definition, for all $r \in res(\pi, j)$ there exists $r' \in req_j \cap opt(\pi)$ such that $r' \subseteq r$. We prove that in this case it also holds $r' = r$. Assume by contradiction that r' is strictly contained in r. We have that $r \in vault(\pi, j)$ and hence $r' \notin vault(\pi, j)$, because $vault(\pi, i)$ is a threshold. By Lemma 6, from $r' \in opt(\pi) = opt_{-j}(\pi)$ it follows $r' \in vault(\pi, j)$, which is a contradiction. Hence, $r \in opt(\pi) \subseteq pol_i = pol_j$, and the thesis. □

Investigating the OpenPGP Web of Trust

Alexander Ulrich[1], Ralph Holz[2], Peter Hauck[1], and Georg Carle[2]

[1] Diskrete Mathematik
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
{ulricha,hauck}@informatik.uni-tuebingen.de
[2] Network Architectures and Services
Fakultät für Informatik
Technische Universität München
{holz,carle}@net.in.tum.de

Abstract. We present results of a thorough analysis of the OpenPGP Web of Trust. We conducted our analysis on a recent data set with a focus on determining properties like usefulness and robustness. To this end, we analyzed graph topology, identified the strongly connected components and derived properties like verifiability of keys, signature chain lengths and redundant signature paths for nodes. Contrary to earlier works, our analysis revealed the Web of Trust to be only similar to a scale-free network, with different properties regarding the hub structure and its influence on overall connectivity. We also analyzed the community structure of the Web of Trust and mapped it to social relationships. Finally, we present statistics which cryptographic algorithms are in use and give recommendations.

Keywords: Web of Trust, OpenPGP, GnuPG, PGP, Community Structure.

1 Introduction

Pretty Good Privacy (PGP) and the GNU Privacy Guard (GnuPG) are implementations of OpenPGP (RFC 4880 [1]). Instead of a hierarchical trust architecture with Certification Authorities as in X.509, OpenPGP employs a certification model where any entity can certify another entity. This results in a so-called Web of Trust (WoT).

In this paper, we describe the results of a thorough investigation of the Web of Trust as established by OpenPGP users. We employed graph analysis to find answers to security-related issues in the WoT. Our contributions are the following. First, we analyzed the OpenPGP WoT's graph components and identified its macro structures. We will see that this is a prerequisite for more detailed analyses as there is a single most important component. Second, we analyzed the 'usefulness' of the WoT for its users. We investigated properties like the length of certification chains, redundant paths, the Small World effect in the WoT and mutual signatures. Third, we determined how robust the WoT is to

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 489–507, 2011.

changes like the random or targeted removal of keys, which can be the result of key expiration, revocation or even attack. Fourth, as the WoT shows properties of a social network, we used State-of-the-Art algorithms to detect community structures and map them to social relations. Finally, we analyzed which cryptographic algorithms are in use and whether this is problematic or not.

The remainder of this work is organized as follows. The following section provides background to OpenPGP. Section 3 describes our methodology. Section 4 presents our results concerning the WoT's usefulness and robustness. The results of our analysis of the community structure are presented in Section 5. Section 6 presents statistics about key properties. Section 7 puts this work into the context of previous, related publications and highlights the differences from our work.

2 Background

Essentially, the WoT is a user-centric and self-organized form of PKI. A user in OpenPGP is identified by a *user ID*, a data structure that contains a user name and e-mail address. Every user ID is associated with a public/private key pair (either DSA/ElGamal or RSA). Users 'issue certificates' to each other by signing another key (i. e. user ID and public key) with their private keys. The exact mechanism of creation of the WoT is not fully known, but it is commonly agreed that personally established contact between users plays a major role, particularly organized events like Keysigning Parties at conferences and meetings. OpenPGP keys are frequently uploaded to a network of key servers. These use the Synchronizing Keyservers (SKS) protocol for synchronization. A snapshot contains a complete history of the network: keys cannot be deleted from an SKS server and timestamps of key creation, signature creation, expiration dates and revocation dates are stored.

The advantages and disadvantages of different PKI structures and trust models have been discussed, among others, by Perlman [2] and Maurer [3]. In contrast to the hierarchical X.509, which is said to suffer from insufficient Certification Authority (CA) practices and insufficient control over intermediate CAs [4], the situation is different in OpenPGP. Firstly, certificates are not verified by following a certification chain from some Root CA (with the chain already known in advance[1]), but by finding a certification path from the own key to the key that is to be verified as belonging to some entity. Secondly, OpenPGP uses a trust metric to allow users to assess the trust in a key-entity binding. There are two notions of 'trust': 'Introducer trustworthiness' refers to how much another user is trusted to apply care when verifying an identity. This value is determined and stored locally for every *locally* known user ID. 'Public-key trustworthiness' is the degree to which a user claims to be sure of a key-entity binding. This value is stored as part of a signature. Before using someone else's public key, users must determine the key-entity binding and assess whether it's likely to be correct. Different trust metrics can be applied here. GnuPG, for example, uses

[1] E. g., most HTTPs servers are configured to send the full chain in the SSL/TLS handshake.

a default setting that focuses on introducer trustworthiness: this must either be 'full' for all keys on the certification path, or there must at least be three redundant certification paths to the key in question. Also, a certification path must not be longer than 5 keys. Trust in OpenPGP thus relies on social relations for identify verification; ideally a WoT should model real-world relationships. CAs are not forbidden in OpenPGP – they are merely a special kind of user. While very flexible, this trust model is very demanding on the user. OpenPGP's model can thus be viewed to be more focused on the local 'environment' of a user – it is infeasible for a user to determine introducer trust for everyone in the WoT. A user can only make reasonable assessments about keys to which paths are short, and lead over social contacts. This also helps with the 'Which John Smith?' problem: looking for the key of a certain 'John Smith' is much easier if it is known that John Smith should share some of one's own contacts.

The open nature of the WoT could lead one to speculate whether large-scale attacks on the WoT are possible, where a malicious entity certifies a large number of keys to trick others. However, this attack is much more difficult than it seems. Assume Alice wants to verify a fake key for the identity Bob, which has only been signed by a number of false identities signed by Mallory. Alice must establish a certification path to the 'fake Bob key' using the faked signed keys. These faked keys would *only* be used in a path search if Alice has manually and explicitly set a trust value for Mallory *and* the false identities. As setting introducer trust is a manual operation, it is unlikely that Alice would assign the needed trust to unknown and 'strange' entities. For this reason, we view it as an unlikely attack.

Also note that multiple keys per user is not uncommon and not evidence of such an attack: one might for example wish to use different keys for business and private matters, or for different levels of security. Multiple (non-revoked, non-expired) keys for one person/entity occur quite commonly in the key database without any evidence for malicious behavior.

From these considerations, we can thus derive several important properties a 'good' WoT must exhibit. It must allow to find certification paths between many keys, otherwise it is not useful. The length of paths is essential: short paths reduce the number of entities on the path that a user has to trust and thus increase a user's chances of accurate assessment of key authenticity. Giving and receiving many signatures is important, too: it increases the chances of several redundant paths between nodes, which is beneficial for GnuPG's trust metrics. It also means that removal of a key has little impact on reachability, which increases the WoT's robustness. Finally, a good WoT should model social relations and social clustering well: where 'communities' of users exist, chances of being able to accurately assess trustworthiness of users within the same community increase.

3 Methodology

In this section, we describe how we extracted the graph topology and summarize the metrics we used in the graph analysis.

Table 1. Our data set

Total number of keys	2,725,504
Total number of signatures	1,145,337
Number of expired keys	417,163
Number of revoked keys	100,071
Number of valid keys with incoming or outgoing signatures	325,410
Number of valid signatures for the latter set of keys	816,785

3.1 Graph Extraction and Analysis

We modified the SKS software to download a snapshot of the key database as of December 2009.

Table 1 shows properties of our data set after our extraction. The data set contains about 2.7 million keys and 1.1 million signatures. Of these, about 400,000 keys were expired, another 100,000 revoked. About a further 52,000 keys were found to be in a defective binary format. *The actual WoT*, which consists only of valid keys that have actually been used for signing or have been signed, is made up of *325,410 keys* with *816,785 valid signatures between them.* Consequently, the majority of keys in the data set is not verifiable (no signature chains lead to them) and does not belong to the WoT. Note that the data set contains only keys from key servers. We cannot know the number of unpublished keys.

When representing the WoT as a graph, we represented keys as nodes and signatures as directed edges. This was a deliberate choice. An alternative would have been to map keys to individual persons. However, such a mapping is not easy to define due to changes of e-mail addresses, spelling of names and the use of pseudonyms. Ultimately, it is keys that sign other keys, and we thus chose to analyze a key-based graph.

3.2 Terms and Graph Metrics

We briefly describe terms and metrics that we use in our analysis of the WoT. For precise definitions, we refer the reader to the Appendix.

Strongly Connected Components (SCCs). A strongly connected component is a maximally connected sub-graph of a directed graph where there is at least one directed path between every node pair u, v. Note that the paths from u to v and v to u may incorporate different nodes.

Distances, Eccentricity, Radius and Diameter. The *distance* between two nodes is the length of the shortest path between them. The *characteristic distance of the graph* is the average over all distances in the graph. *Eccentricity* is a node property that indicates the distance to the node farthest away from this node in the graph. *Graph radius* is defined as the *minimum over all eccentricities* and the *diameter* is defined as the *maximum over all eccentricities*.

Neighborhoods. A *node v's neighborhood* is the set of all nodes for which the distance from v is at most a certain value.

Clustering Coefficient. The *clustering coefficient* is a measure of transitivity in a graph. It indicates the probability that two neighbors of a node are themselves neighbors, i.e. have an edge between them.

Correlation of Node Degrees. Pastor-Satorras et al. [5] defined a measure for the correlation of node degrees in a function *knn*. It determines whether nodes with similar degrees have edges between them. The *assortativity coefficient* [6] is a similar measure. It measures how many nodes with high degree are connected mainly to other nodes with high degree.

4 Results

We present the results of our analysis.

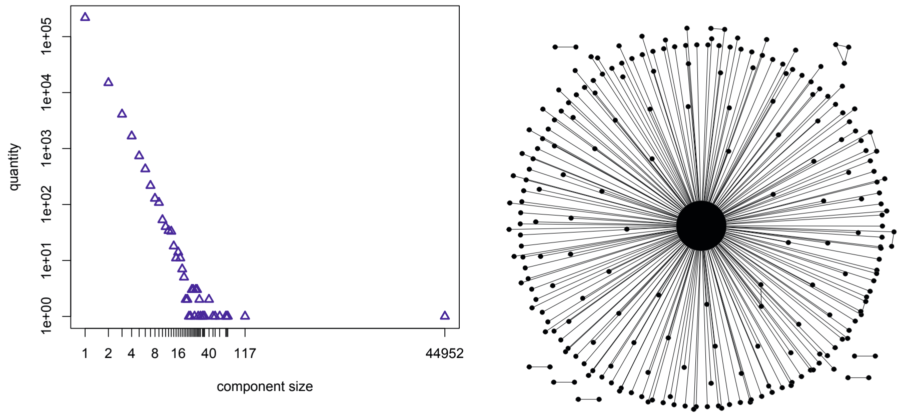
4.1 Macro Structure: Strongly Connected Components (SCCs)

Within SCCs, there is at least one signature chain between every key pair. SCCs are thus important for participants of the WoT: mutual verification of key authenticity is only possible for participants within the same SCC. An optimally meshed WoT should be one giant SCC.

We computed the SCCs of the graph, and found 240,283 SCCs in the WoT. However, more than 100,000 of these consisted of a single node and about 10,000 SCCs consist of node pairs. The largest SCC (LSCC) consists of about 45,000 nodes. The remaining SCCs mostly have a size between 10 and 100 nodes. Figure 1 (a) shows the distribution. The SCCs can be arranged in a star formation around the LSCC in the middle (Figure 1 (b)).

Many SCCs have *uni-directional* edges to the LSCC, but extremely few have edges between each other. Out of all smaller SCCs, about 18,000 nodes show a uni-directional edge into the LSCC, making it (in principle) possible for such a key to verify keys from the LSCC. In the other direction, 92,000 keys outside the LSCC are reachable from a key within the LSCC. We found three interesting hubs in the LSCC and one regional particularity. The German publisher Heise, CACert and, until recently, German DFN-Verein operate or have operated CAs to sign keys. Together, they have signed about 4,200 keys in the LSCC. The Heise CA alone has, in total, signed 23,813 keys – yet of these only 2,578 are in the LSCC.

This SCC structure gravely impacts the usability of the WoT. First of all, the large number of smaller SCCs means that even among those users who have made the effort to upload their keys to a key server, most do not participate actively in the WoT. Otherwise, their SCCs would already have merged with the LSCC (one mutual signature is enough). This is also emphasized by the following comparison. The ratio of edges:nodes in the LSCC is 9.85; the same ratio for the total WoT is 2.51. Signature activity in the LSCC must thus be



(a) (b)

Fig. 1. (a) Size distribution of SCCs. (b) Plot of SCCs down to a size of 8.

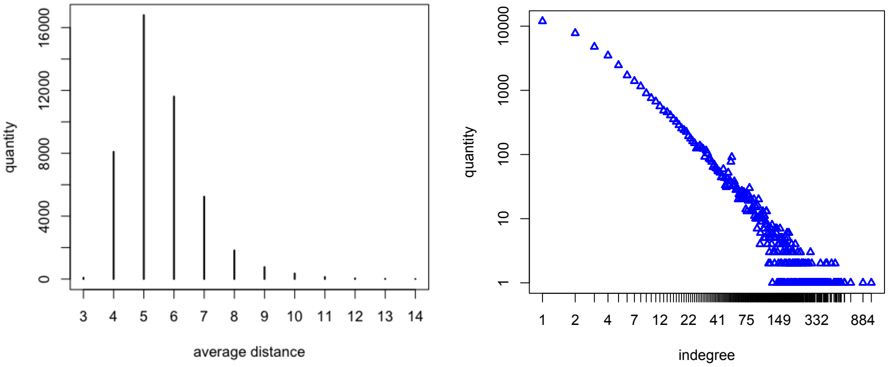
much higher than in the rest of the WoT. However, strong user activity is very desirable to achieve a better meshing in the WoT.

Second, a high percentage of participants in one of the smaller SCCs are unable to verify most keys in the WoT. The LSCC is really a structure of paramount importance: the keys in the LSCC constitute only 14% of the keys in the WoT, but only the owners of these keys can really profit in a significant way from the WoT. They can build signature chains to all keys in the LSCC plus to twice as many keys outside of the LSCC. Thus, a recommendation for new participants would be to obtain a signature from a member of the LSCC as early as possible to make their key verifiable. A good choice is also to get a (mutual) signature of one of the CAs in the LSCC. With such a signature, paths can be built to all keys in the LSCC, plus to a large number of keys outside the LSCC that are only reachable via the CA. This emphasizes that a WoT can benefit from CAs.

The remainder of our analysis focuses on the LSCC as the most relevant component for participants.

4.2 Usefulness in the LSCC

'Usefulness' is a term that is difficult to express formally. It can be defined in several dimensions. An obvious one is how many keys are verifiable from a given key, and how many paths to other keys can be found from the given key. The higher these numbers are, the more useful the WoT is from the perspective of this key. Recall that introducer trustworthiness is not stored in the signatures: the following discussion thus relates to *upper bounds*.



(a) (b)

Fig. 2. Distribution of (a) average distances, (b) indegree in LSCC

Distances. We first analyzed distances between keys in the graph. The *average distances between nodes* in the LSCC (see Figure 2(a)) range between 4–7, which is at best just below GnuPG's limit (path length 5), but exceeds it at worst. The *eccentricity* in the LSCC is much higher: it is almost exclusively between 26–31. To determine the implications of this for usefulness, we identified how many keys are reachable from a given key within a certain distance.

We computed the set of verifiable keys as the nodes in a *h-neighborhood* for $h = 1, .., 5$ (see Definition 5 in the Appendix). Figure 3 shows the CDF of *h-neighborhoods*. For the 2-neighborhood, we see a steep incline, from which we can conclude that this neighborhood must be relatively small for all nodes. The size of the neighborhoods grows considerably for increasing h. For $h = 3$, the third quartile is about 3,300. For $h = 4$ and $h = 5$, it becomes 16,300 and 30,500, respectively.

Our findings indicate that signature chains within GnuPG's restrictions are sufficient to make a very large fraction of the keys in the LSCC verifiable. This is a good result for usefulness and shows that the LSCC is quite well meshed. However, for $h = 5$, the maximum number of reachable keys we found was 40,100. This means that, on average for all keys, there will be almost 5,000 keys (a tenth of the LSCC) to which no path at all can be found within GnuPG's restrictions.

Small World Effect and Social Links. The size of 5-neighborhoods shows that paths are frequently very short. A possible explanation for this is a Small World effect, which – following [7] – can be informally understood to be the phenomenon that the average path length between any two nodes is significantly shorter than could be expected by judging from graph radius and diameter. A high clustering coefficient is often viewed as indicative. We investigated this in the LSCC. As there does not seem to be a universally accepted definition of the clustering coefficient for directed graphs, we reduced the directed graph to an undirected one (omitting the direction of edges and merging duplicates). The

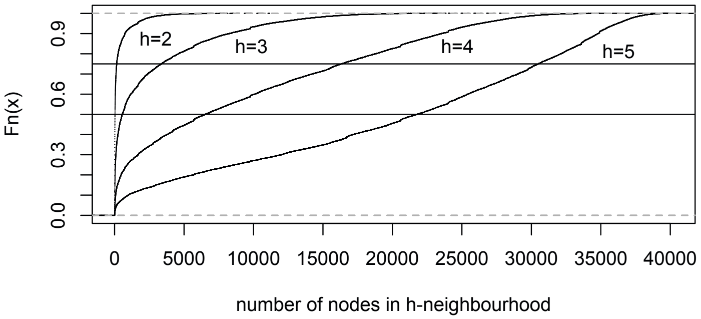


Fig. 3. CDF of reachable nodes due to h-neighbourhoods

clustering coefficient we computed is $C = 0.46$. This indicates that, *on average*, roughly half of all neighbors of a node have edges between them. The value is of the same order as described in [7] for social networks with strong clustering. The characteristic distance in the LSCC is 6.07, while the diameter of the graph is 36 and the radius 16. Our finding is that the LSCC does indeed show a Small World effect. This indicates social clustering. Together with the short paths, this would make trust assessments easier for users. We explore the social nature of the WoT further in Section 5.

Node Degrees. Recall that GnuPG's trust metrics view redundant and distinct signature chains as beneficial for a key's trustworthiness. A high node indegree thus means that the corresponding key is more likely to be verifiable by other keys. A high outdegree increases the likeliness to find redundant signature chains to other keys. We computed the average indegree (and outdegree) in the LSCC as 9.29. However, as can be seen in Figure 2(b), the distribution of indegrees in the LSCC is skewed. The vast majority of nodes have a low indegree (i. e., 1 or 2). The result for the outdegrees is very similar: as can be seen in Figure 4(a), there is a positive correlation between indegree and outdegree of a node. The plot for outdegrees is indeed so similar to the one for indegrees that we omitted it here. About a third of nodes in the LSCC have an outdegree of < 3. Together, these results mean that the WoT's usefulness has an important restriction: many nodes need to rely on single certification paths with 'full' introducer trust and cannot make use of redundant paths.

Mutual Signatures (Reciprocity of Edges). If many WoT participants cross-signed each other, this would be a great improvement in overall verifiability of keys. We computed the reciprocity of edges, i. e. the fraction of uni-directional edges to which there exists a uni-directional edge in the other direction. The LSCC has a reciprocity value of 0.51. This shows that there is room for improvement: the LSCC would profit much if more mutual signatures were given, which would of course also strengthen indegree and outdegree and shorten distances.

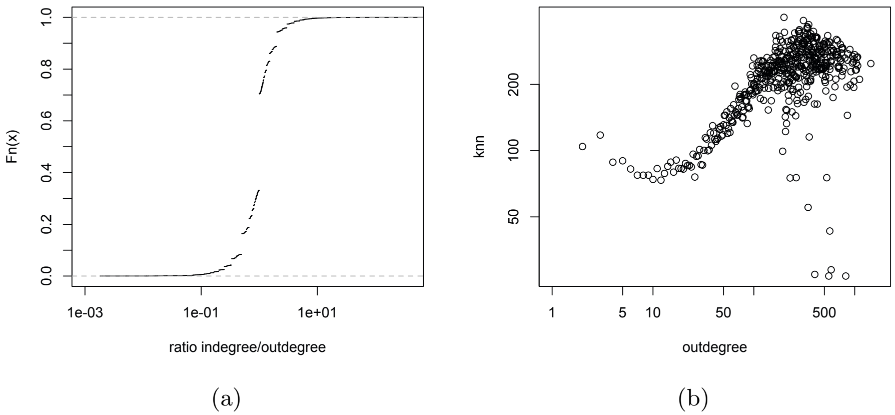


Fig. 4. (a) CDF of ratio indegree:outdegree in LSCC. (b) Correlation of node degrees according to Definition 8 (see Appendix): average outdegree (*knn*) of neighbors of nodes with degree *k*

4.3 Robustness of the LSCC

The robustness of the LSCC is also an interesting topic: how is the LSCC connected internally, and hence how sensitive is it to removal of keys? In the context of OpenPGP, the random removal of a node can be the result of an event like key expiration or revocation, which invalidates paths leading over the key in question. These events can and do occur in practice. Targeted removal of a key, however, is very hard to accomplish as SKS never deletes keys and stays synchronized. An attacker would need an unlikely high amount of control over the SKS network to make a key disappear.

Scale-Free Property. Scale-freeness in a graph means that the node degrees follow a Power Law. Connectivity-wise, scale-free graphs are said to be robust against random removal of nodes, and vulnerable against the targeted removal of hubs (which leads to partitioning). This is usually explained by the hubs being the nodes that are primarily responsible for maintaining overall connectivity [8]. We thus first investigated to which extent the WoT shows this property.

The double-log scale in Figure 2(a) could lead one to the conclusion that the distribution of node degrees follows a Power Law. However, Clauset et al. argued in [9] that this is not indicative and methods like linear regression can easily be inaccurate in determining a Power Law distribution. We followed the authors' suggestion instead and used the Maximum Likelihood method to derive Power Law coefficients and verified the quality of our fitting with a Kolmogorov-Smirnov test. [9] gives a threshold of 0.1 to safely conclude a Power Law distribution. Our values for indegrees and outdegrees were 0.012 and 0.011, respectively. As this is off by a factor of 10, our conclusion is that a Power Law distribution

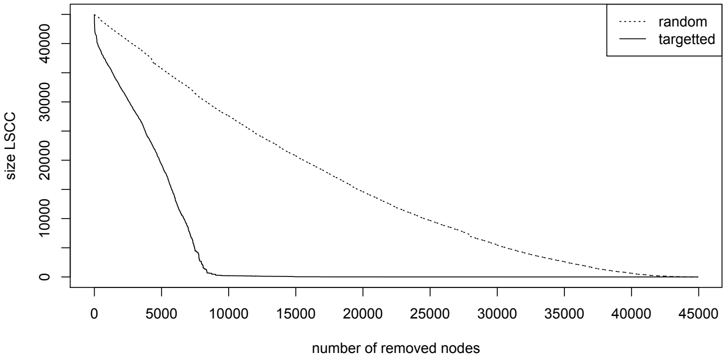


Fig. 5. Removing nodes at random and in a targeted fashion and recomputing the size of the LSCC

is not plausible. Consequently, the graph cannot be scale-free in the strict sense of the definition. This finding is contradictory to earlier works by Boguna et al. [10] and Capkun et al. [11].

The question is yet whether the graph is still similar to a scale-free one. Apart from high variability of node degrees, a set of high-degree nodes that act as inter-connected hubs are characteristic for scale-free graphs [8,12]. The positive correlation between the degree of nodes and the average degree of their neighbors (Figure 4(b)) suggests that nodes with high outdegrees do indeed connect to other such nodes with high probability. To bolster our finding, we computed the assortativity coefficient (see Appendix A.4) and obtained a value of 0.113. This is similar to what has been computed for other social networks with a hub structure [7]. Our conclusion is thus that the graph is similar to a scale-free one and exhibits a hub structure, but is not scale-free in the strict sense.

Random Removal of Nodes. Based on this finding, we investigated how the LSCC reacts to random removal of nodes. We removed nodes and recomputed the size of the remaining LSCC as an indication of loss in overall connectivity. For random removal, we picked the nodes from a uniform distribution. Figure 5 shows our results. The graph is very robust against the random removal of nodes: we must remove 14,000 nodes to cut the LSCC's size by half. To reduce it to a quarter, we must remove more than half the nodes (25,000).

The conclusion here is that events like key expiration or revocation do not greatly influence the robustness, and consequently the usability, of the WoT.

Targeted Removal of Nodes and CAs. For targeted removal, we chose nodes with highest degrees first. The graph was more robust than expected. When we removed all nodes with a degree of more than 160 (240 nodes), the size of the LSCC was still 40,000. Only when we proceeded to remove all nodes with a degree of more than 18 ($\sim$ 5,000 nodes), the LSCC was half its size. Removing 2,500 more nodes, we finally cut the LSCC down to about 1/9 of its original size. This means that nodes with lower degrees (< 18) play a significant role in overall connectivity (although the decay of the LSCC is quite pronounced after they are also removed). The rather slow decay stands in contrast to the rapid decay upon removal of the best-connected nodes that is commonly observed in scale-free networks. Targeted removal of keys does not affect the WoT greatly, and is not an efficient attack. The hub structure is not the single reason for highly meshed connectivity in the WoT.

We decided to strengthen the attack by removing the keys of the three CAs. Our finding was similar: the LSCC split into one LSCC of size 42,455 and 1,058 very small SCCs. This means that the CAs, although beneficial in making keys verifiable, are not responsible for holding the LSCC together. The characteristic distance of the new LSCC remained almost unchanged (6.25); radius, diameter and eccentricity remained the same. This means that path properties did not change, either. Our conclusion here is that attempting to selectively remove keys from keyservers, even shutting down CAs, would not change the WoT's properties significantly. It is very robust in this respect.

5 Community Structure of the Web of Trust

We know from Section 4 that the WoT shows the small-world property, which hints at social clustering. Newman and Park also noted that a high degree of clustering is typical for social networks [13]. Fortunato [14] calls such subsets of nodes 'communities' if the nodes have high intra-connectivity in their subset, but the subset as such shows a much lower connectivity to nodes outside. Social clustering can make the WoT more powerful: it is more likely that members of a cluster know each other at least to some extent and can thus better assess the trustworthiness of particular keys.

Community Detection. We analyzed the WoT with State-of-the-Art algorithms for community detection to determine whether a pronounced community structure exists and can be mapped to 'real-world' relationships. Also, we attempted to find whether signing events like Key Signing Parties can be identified in the graph. Unfortunately, algorithms for community detection are often defined for undirected graphs. Also, signatures store little information that helps with identifying social links and events in time. We decided to use DNS domains in user IDs and timestamps of signature creation as a basis. As an algorithm for a directed graph, we chose the one by Rosval et al. [15]. For undirected graphs, we chose the algorithms by Blondel et al. [16] and COPRA [17], based on suggestions in [18]. COPRA allows overlapping communities, but is non-deterministic. We ran it 10 times and computed differences. As a measure for the quality of a dissection, we used *Modularity* [19], which relates the amount of intra-cluster edges of a graph with communities to the expected value for a graph without communities. Note that the definition for overlapping communities is different, so the values for COPRA and BL cannot be compared directly.

Table 2. Dissection of the LSCC into communities: algorithms BL and COPRA

Method	Modularity	Communities found (size > 3)
BL ($l = 2$)	0.70	936
BL ($l = 5$)	0.71	186
COPRA ($v = 1$)	(0.78)	1,421
COPRA ($v = 3$)	(0.79)	1,354

Only the algorithms by Blondel et al. and COPRA yielded useful results. The algorithm by Rosval et al. computed a dissection into 2,869 communities, almost all of them without any intra-cluster edges. We considered these results unreliable and ignored them in our subsequent evaluation.

Blondel et al. and COPRA. Table 2 shows the results of dissections with Blondel et al. (BL) and COPRA for communities of size > 3. Both BL and COPRA are configurable: BL can be repeated in iterative phases and COPRA requires a (user-chosen) parameter v to reflect the degree of overlapping. For BL, phase 2 yielded the best results (plausible number of communities, high modularity). For COPRA, values of v up to 3 were found best. We know from [20] that modularity values > 0.3 indicate a significant community structure. Depending on the algorithm and chosen parameters, between 94% (COPRA) and 99% (BL) of nodes in the LSCC belonged to such a community.

BL and COPRA agree on the same orders of magnitude with respect to the number of communities and nodes therein. The high modularity values and the general shape of community distributions by size (see Figure 6) are also similar. Most communities are very small, but a significant number of large or very large communities exist. Similarities, however, end here. COPRA indicates one extremely large community of 19,000-21,000 members. BL finds more communities of medium size (100-500) and mid-large size (500-5,000). To further investigate this, we analyzed how communities are connected. COPRA found that most small communities are clustered around the largest community and mostly only link to this community. BL found several large communities to which the smaller communities connect.

Mapping to Domain Names and Keysigning Parties. We analyzed how the community dissections mapped to top-level and second-level domains (TLDs and SLDs) in the user IDs. We say a community is *dominated* by a domain if at least 80% of its nodes belong to that domain. We say a community can be *assigned* to a domain if at least 40% of its nodes belong to it.

Table 3 shows the results. For both BL and COPRA, we found that a large percentage of communities are dominated by a top-level domain: between 47% and 58%. Only if a community was not dominated, we checked if it could at least be assigned. A further 38%–47% could be said to be assignable to a TLD.

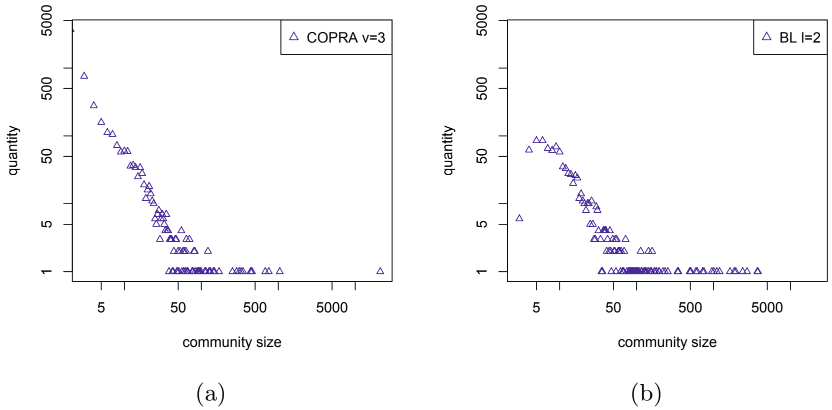


Fig. 6. Distribution of communities by size

This result did not change much when we disregarded generic TLDs (`.com` etc.): with COPRA, 38% of communities were dominated by a country's TLD and a further 23% were still assignable. Results for BL were similar. Together, assigned and dominated communities make up by far the largest part of communities found (98% for BL-2 and COPRA, $v = 1$). However, the picture changes for second-level domains. With COPRA, only about 13% of communities are dominated by an SLD and only a further 30% of communities can be assigned to an SLD.

Keysigning Parties are events where one can expect signatures to be uploaded to key servers within a short time frame. Table 3 shows the percentage of nodes in the communities where signatures were created within a month. We find poor results for BL, but much better ones for COPRA. In about 40% of communities, the signatures were created within 30 days of each other.

Conclusion with Respect to Community Detection. Concerning community detection, it is difficult to reach compelling conclusions. We provide ours as a basis of discussion. Both algorithms agreed that a large number of smaller communities exist. Given the huge number of TLDs and SLDs and given that the WoT graph spans more than a decade, the results seem statistically significant enough to conclude that the community structure does indeed capture some 'social' properties of the WoT. However, grouping by TLD is a blunt measure, and the mappings to SLDs were by far not as compelling. Our tentative conclusion is that the signing process in the WoT is indeed supported, to a traceable extent, by real-world social links. The social nature of the WoT is not a myth. At least where certification paths are short, the community structure should make it easier for users to assess the trustworthiness of a key. Beyond this result, however, community detection is yet too imprecise to offer more succinct conclusions.

Table 3. Community structure with respect to membership in top-level domains (TLD) and second-level domains (SLD)

Method	dominated by TLD	assignable to TLD	dominated by SLD	assignable to SLD	signatures within 30d
BL ($l = 2$)	499 (53%)	417 (45%)	41 (4%)	254 (27%)	115 (12%)
BL ($l = 5$)	85 (47%)	85 (47%)	15 (8%)	38 (21%)	26 (14%)
COPRA-1	824 (58%)	564 (40%)	178 (13%)	429 (30%)	572 (40%)
COPRA-3	792 (58%)	525 (38%)	187 (14%)	425 (31%)	555 (41%)

6 Cryptographic Algorithms

Table 4 presents results on the use of hash and public key algorithms in the WoT. Several algorithms encountered raise security concerns: MD5 can probably be said to be an unwise choice today [21]. SHA-1, although much safer, is also scheduled for phase-out [22]. RSA keys of 768 bits have been factored [23] and a length of more than 1,024 bits is recommended since 2010 [24].

Table 4. Occurrences of (a) hash algorithms, (b) public key algorithms

(a)

Algorithm	Occurrences
SHA1	398,849
MD5	41,700
SHA256	5,031
SHA512	2,472
SHA224	532
RIPE-MD/160	122
Signatures total	446,325

(b)

Algorithm	Occurrences
DSA-1024	36,555
RSA-1024	3,903
RSA-2048	2,408
RSA-4096	1,198
RSA-768	257
RSA-512	203
RSA-3072	96
Keys total	44,952

Especially the comparatively high number of RSA keys with a key length of $\leq$ 1,024 bits is somewhat problematic. We investigated these keys and found that a substantial number of them appears well-connected, based on their in- and outdegrees. It seems reasonable to assume that quite a few users trust these keys as introducers, thus enabling their use in certificate chains. Although not a threat yet and possibly also not for the next few years, it opens up attack opportunities if factorization of 1,024 bits keys should become feasible [23].

7 Related Work

The OpenPGP WoT has been the subject of investigation before, albeit at other stages of its development and with a focus that was less on security-relevant properties. Capkun et al. [11] analyzed several structural aspects of the WoT of

2001. They did not investigate aspects like communities but presented a model to create similar graphs. They found a small characteristic distance and a high clustering coefficient. The authors claimed to have found a Power Law distribution for node degrees. Our own findings are that a Power Law distribution is not plausible. However, the graph is similar to a scale-free one, although its hub structure is not solely responsible for robustness. Note however that the rigid methods in [9] were generally not as widely in use then, and the graph from 2001 contained 4 times fewer nodes. Boguna et al. [10] also analyzed a PGP graph from 2001. They converted the graph to an undirected one and analyzed node degrees and clustering coefficient. They also claimed a Power Law for node degrees and determined a clustering coefficient on the same order as the one we found. The authors also applied an (older) algorithm for community detection. They claimed the community distribution follows a Power Law, too. All of the above have in common that they used significantly older data sets, and the focus was less on security issues like usefulness and robustness. Furthermore, our community dissection was conducted with more recent algorithms, with the aim of mapping communities to real-world groups. The OpenPGP community has also contributed some effort in analyzing the WoT's structure. The *wotsap* project [25] creates snapshots of the signatures in the WoT. However, it only considers the LSCC and does not store other key properties. We also found the data set to be incomplete (10% of keys missing) due to a bug. Penning [26] used the wotsap data set to determine aspects like distances, node distribution and robustness based on node removal.

8 Discussion and Conclusion

We have presented several results relating to security aspects of the OpenPGP Web of Trust. We found that only keys in the Largest SCC (LSCC) can really profit from the WoT. This severely limits the reach of the WoT to a fraction of its users: only about 45,000 keys out of 2 million can use the WoT without restrictions. A large fraction of keys in the smaller SCCs can make very little use of the WoT or none at all. However, for users with keys in the LSCC, the situation is much better. We found their certification chains to be relatively short. There is also a pronounced Small World effect. We followed this up with an investigation of the community structure of the WoT. While algorithms for community detection can capture the social groups of the WoT on a very coarse level only, the graph does exhibit a very strong community structure. Another positive aspect is that about 40,000 of 45,000 keys are reachable within GnuPG's restrictions (5 hops), and several thousand even via 3 hops or less. This is positive for the WoT as it can aid users in making better trust assessments regarding other keys that are close and in the optimal case also in the same community. The CAs we found help greatly in making keys verifiable. This is a viable option for users. Random removal of keys (e. g., due to expiration or revocation) is not a problem for the robustness of the WoT. The WoT is also very robust against targeted attacks; CAs are not fundamentally relevant for robustness.

However, we found that low indegrees and outdegrees are far too common. This reduces the number of redundant paths between keys, which means that many users would need to have ‘full’ introducer trust in known entities. Mutually cross-signing more often would help here.

In essence, our conclusion is that the WoT is likely to be quite an effective PKI structure *within smaller node neighborhoods, and particularly for those users that frequently sign other keys and are active in the WoT*. The cryptographic algorithms that are in use can be generally considered to be still secure. However, keys that have issued MD5-based signatures should be replaced and signatures renewed. Also, a stronger move towards key lengths of more than 1,024 bits is desirable.

Acknowledgements. We would like to thank both our anonymous reviewers and Radia Perlman for their valuable input.

References

1. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP Message Format. RFC 4880 (November 2007)
2. Perlman, R.: An overview of PKI trust models. IEEE Network 13(6), 38–43 (1999)
3. Maurer, U.: Modelling a public-key infrastructure. In: Martella, G., Kurth, H., Montolivo, E., Hwang, J. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 325–350. Springer, Heidelberg (1996)
4. Eckersley, P., Burns, J.: An observatory for the SSLiverse. Talk at Defcon 18 (July 2010), `https://www.eff.org/files/DefconSSLiverse.pdf` (online; last retrieved in February 2011)
5. Pastor-Satorras, R., Vázquez, A., Vespignani, A.: Dynamical and correlation properties of the Internet. Phys. Rev. Lett. 87(25), 258701 (2001)
6. Newman, M.E.J.: Assortative mixing in networks. Phys. Rev. Lett. 89(20), 208701 (2002)
7. Newman, M.E.J.: The structure and function of complex networks. SIAM Review 45(2), 167–256 (2003)
8. Albert, R., Jeong, H., Barabasi, A.L.: Error and attack tolerance of complex networks. Nature 406(6794), 378–382 (2000)
9. Clauset, A., Shalizi, C.R., Newman, M.E.J.: Power-law distributions in empirical data. SIAM Review 51(4), 661–703 (2009)
10. Boguñá, M., Pastor-Satorras, R., Díaz-Guilera, A., Arenas, A.: Models of social networks based on social distance attachment. Phys. Rev. E 70(5), 056122 (2004)
11. Capkun, S., Buttyán, L., Hubaux, J.P.: Small Worlds in security systems: an analysis of the PGP certificate graph. In: NSPW 2002: Proc. 2002 Workshop on New Security Paradigms, pp. 28–35. ACM, New York (2002)
12. Li, L., Alderson, D., Doyle, J.C., Willinger, W.: Towards a theory of scale-free graphs: Definition, properties, and implications. Internet Mathematics 2(4), 431–523 (2005)
13. Newman, M.E.J., Park, J.: Why social networks are different from other types of networks. Phys. Rev. E 68(3), 036122 (2003)
14. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75–174 (2010)

15. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. Proc. National Academy of Sciences 105(4), 1118–1123 (2008)
16. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), 10008 (2008)
17. Gregory, S.: Finding overlapping communities in networks by label propagation. New Journal of Physics 12(10), 103018 (2010)
18. Lancichinetti, A., Fortunato, S.: Community detection algorithms: A comparative analysis. Phys. Rev. E 80(5), 056117 (2009)
19. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 026113 (2004)
20. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70(6), 066111 (2004)
21. Sotirov, A., Stevens, M., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: MD5 considered harmful today (2008), http://dl.packetstormsecurity.net/papers/attack/md5-considered-harmful.pdf (online; last retrieved in May 2011)
22. NIST: Approved Algorithms (2006), http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html (online; last retrieved in May 2011)
23. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A., Thom, E., Bos, J., Gaudry, P., Kruppa, A., Montgomery, P., Osvik, D., te Riele, H., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA modulus. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 333–350. Springer, Heidelberg (2010)
24. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: NIST special publication 800-57 part 1, recommendation for key management - part 1: General (revised) (2007), http://csrc.nist.gov/groups/ST/toolkit/key_management.html
25. Cederlöf, J.: Web of Trust statistics and pathfinder, http://www.lysator.liu.se/~jc/wotsap/ (online; last retrieved in February 2011)
26. Penning, H.P.: Analysis of the strong set in the PGP web of trust, http://pgp.cs.uu.nl/plot/ (online; last retrieved in February 2011)
27. Brinkmeier, M., Schank, T.: Network statistics. Network Analysis, 293–317 (2004)

A Common Terms and Graph Metrics

Based on the common notions of graph theory, we define some terms, following [27] herein. In the following, let V be the set of nodes of the graph G, with $|V| = n$. u and v indicate nodes.

A.1 Distances

Distances Between Nodes. The distance d between two nodes is defined as the length of the shortest path between these two nodes.

Distances in the Graph. The average distance of the graph, $\bar{d}$, is the average over all distances in the graph:

$$\bar{d} = \frac{1}{n^2 - n} \sum_{u \neq v \in V} d(u, v) \tag{1}$$

Eccentricity. The eccentricity of a node u, $\epsilon(u)$, is defined as the maximum distance to another node, i.e.

$$\epsilon(u) = \max\{d(u, v) | v \in V\} \tag{2}$$

Graph Radius and Diameter. The diameter of a graph is defined as the maximum over all eccentricities:

$$dia(G) = \max\{e(u) | u \in G\} \tag{3}$$

The *radius* is defined as the minimum over all eccentricities:

$$rad(G) = \min\{e(u) | u \in G\} \tag{4}$$

A.2 Node Neighborhoods

We define the *h-neighborhood* of a node v as the set of all nodes from which the distance to v is at most h:

$$N_h(v) = \{u \in V | d(v, u) \leq h\} \tag{5}$$

A.3 Clustering Coefficient

The *clustering coeffcient* indicates the probability that two neighbors of a node have an edge between them.

Let $G = (V, E)$ be the undirected graph. A triangle $\triangle = \{V_\triangle, E_\triangle\}$ is a complete sub-graph of G with $| \triangle | = 3$. The number of triangles of a node v is given by $\lambda(v) = |\{\triangle : v \in V_\triangle\}|$. A *triplet* of a node v is a sub-graph of G that consists of v, 2 edges, plus 2 more nodes such that both edges contain v. The number of triplets of a node v can be given as $\tau(v) = \binom{d(v)}{2}$. The *local clustering coefficient* of v is defined as

$$c(v) = \frac{\lambda(v)}{\tau(v)} \tag{6}$$

$c(v)$ indicates how many triplets of v are triangles. The *global clustering coefficient* of G can then be defined as:

$$C(G) = \frac{1}{|V'|} \sum_{v \in V'} c(v) \tag{7}$$

with $V' = \{v \in V : d(v) \geq 2\}$ to disallow non-defined values for $\tau(v)$.

A.4 Correlation of Node Degrees

Function *knn* as defined by Pastor-Satorras et al. Following Pastor-Satorras et al. [5], we define a measure for the correlation of node degrees:

$$< knn >= \sum_{k'} k' P_c(k'|k) \tag{8}$$

gives the average node degree of neighbors of nodes with degree k. $P_c(k'|k)$ indicates the probability that an edge that starts at a node with degree k ends at a node with degree k'.

Assortativity Coefficient. The *assortativity coefficient* [6] is a measure whose purpose is similar to the function defined in Definition 8. It measures the degree of *assortative mixing* in a graph: nodes with high degree that are connected mainly to other nodes with high degree. The *assortativity coefficient* takes values between -1 and 1. Positive values indicate assortative mixing, negative ones do not. According to Newman [6], assortative mixing is a property that distinguishes social networks from other real-world networks (e.g. technical or biological ones). It can thus be used to sub-differentiate between similar graphs that show a Small World effect.

A Practical Complexity-Theoretic Analysis of Mix Systems

Dang Vinh Pham[1], Joss Wright[2], and Dogan Kesdogan[1]

[1] Siegen University, Siegen, Germany
[2] Oxford Internet Institute, University of Oxford, Oxford, United Kingdom

Abstract. The *Minimal-Hitting-Set attack* (HS-attack) [10] is a well-known passive intersection attack against Mix-based anonymity systems, applicable in cases where communication behaviour is non-uniform and unknown. The attack allows an observer to identify uniquely the fixed set of communication partners of a particular user by observing the messages of all senders and receivers using a Mix. Whilst the attack makes use of a provably minimal number of observations, it also requires solving an NP-complete problem. No prior research, to our knowledge, analyses the *average* complexity of this attack as opposed to its worst case.

We choose to explore the HS-attack, as opposed to statistical attacks, to provide a baseline metric and a practical attack for unambiguously identifying anonymous users. We show that the average complexity of the HS-attack can vary between a worst-case exponential complexity and a linear-time complexity according to the Mix parameters. We provide a closed formula for this relationship, giving a precise measure of the resistance of Mixes against the HS-attack in practice, and allowing adjustment of their parameters to reach a desired level of strength.

1 Introduction

Modern research into network-level anonymity is widely regarded to have begun in 1981 with the introduction of the Mix by Chaum [3]. The Mix hides the linkage between senders and recipients of messages by ensuring that all senders and recipients are a member of some *anonymity set*.

The concepts underlying the Mix remain the basis for a wide variety of practical and theoretical anonymity systems. Chaum's model of the Mix, in its pure theoretical form, provides an upper limit to what is achievable by these approaches and thus remains an important subject for analysis. The work presented here provides insight into the limits of the Mix model in practical use, and thus aims to guide choices involved in building real-world implementations of Mix variants.

Although a Mix provides unlinkability between input and output messages with respect to a *global passive attacker*, it cannot protect the links between senders and recipients against *long term traffic analysis* attacks when the sender group is *open* [2, 9].

The anonymity property can be modelled with standard security techniques: a global passive attacker model, which provides a strong but realistic adversary, and with the creation of *anonymity sets* as a basis for the anonymity property that we seek to enforce. We consider in this paper an abstract model called the *Pure Mix* [9] that can be used

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 508–527, 2011.

to model more complex practical Mixes [14, 15]. Analysis of this model is believed to be applicable to other Mix models, with appropriate modifications, but these are not addressed here.

Berthold et al. [2] introduced a class of long term traffic analysis attacks on the Pure Mix, called *intersection attacks*, proving that Mixes with open sender groups cannot provide long term unlinkability if each sender repeatedly communicates with a fixed recipient. In practical usage, however, a sender may have several communication partners and a less restrictive model is therefore needed. Kesdogan et al. introduced the Disclosure [1] and Minimal-Hitting-Set (*HS*-) attacks [9, 10] for repeated communication with an arbitrary fixed set of recipients.

These attacks exploit the fact that a global passive attacker can selectively observe only *recipient anonymity sets* in which a particular sender, referred to as *Alice*, contributes a message. Given sufficiently many observations, Alice's recipient set is the smallest unique set intersecting each of the observations: the *unique minimal hitting set*. Unfortunately, computing this minimal set is known to be an NP-complete problem [8]. Many popular current attacks against Mixes analyse statistical properties [4, 5, 6, 11, 16] to deduce the most likely senders of given messages. These attacks, whilst allowing a level of innaccuracy in results, have the advantage of being much more efficiently computable.

We choose to focus on the Minimal-Hitting-Set attack for a number of reasons: firstly, as the HS-attack uses a provably minimal number of observations [9], it provides an important theoretical baseline for exact identification of a sender's recipients, resulting a metric for Mix anonymisation. Secondly, as we show, it remains a genuinely practical attack in many cases. As the attack is applicable to any distribution of user communications, even when this distribution is unknown to the attacker [10], it applies in many situations with unknown communication behaviour.

Finally, the underlying algorithmic structures related to the HS-attack, based on an NP-complete problem, are themselves an important topic. The research presented here sheds light on analysing the average case complexity of NP-complete problems, and thus those cases in which such problems are computationally tractable.

Contribution. This paper contributes, to our knowledge, the first robust and detailed security analysis of the Mix system based on the average computation required to unambiguously identify users with a provably minimal number of observations. It derives, for a given set of Mix parameters, a direct relationship between the number of observations required for identification and the average-case runtime complexity of the attack.

Our analysis is applicable to non-uniform user communication, and allows us to identify Mix parameters for which unambiguous identification of recipients is intractable in the average case. We also identify instances in which recipients of a sender can be efficiently identified by the HS-attack, providing a provably correct alternative to the more popular statistical approaches.

We show that the NP-completeness of the algorithm deployed by the HS-attack represents only a *worst-case* attack complexity, which provides a poor characterisation when considering systems where the *average* time to failure is of greater relevance.

In this work, therefore, we explore the complexity *structure* of an exact attack in order to determine the average-case complexity, and provide closed formulas that show the relation between the parameters of the Mix and the average-case complexity required to compromise its anonymity.

The Mix model we employ has been used to model practical Mix implementations such as Mixminion and Mixmaster [7, 15], as well as in other analyses [2, 1, 4, 9, 5, 16, 13].

Related Work. A security metric makes a quantitative statement concerning the resistance of a system to an attacker. Our attack model consists of a passively observing attacker against a given anonymity system. The attack that we consider relies solely on observations of this anonymity system. The success of the attack is therefore dependent on the attacker's *knowledge*, and knowledge gain, and on its *computational capabilities*. This form of attack is well known as *passive traffic analysis attack* in the literature. These attacks are hard to thwart, as they exploit the information leakage inherent in all anonymity systems.

Statistical long-term traffic analysis attacks are closest to our approach. These address cases in which Alice's communication behaviour reveals statistical patterns that allow identification of her likely recipients. By relaxing the requirement for absolute correctness, these attacks gain significant computational efficiency.

Greedy variants of the HS-attack, the SHS- and HS*-attacks, were suggested in [10]. These compute hitting sets guided by the frequency with which a peer was contacted while under observation. The result of the SHS-attack is a hitting set that is consistent with the observations made by the attacker, but which can miss Alice's real recipients (*exclusion-error*) or contain recipients not contacted by Alice (*inclusion-error*). In contrast to the SHS-attack, the HS*-attack accepts the greedily computed hitting set only if it is a unique smallest minimal hitting set. This attack can identify Alice's recipient set, but risks producing no result.

The Statistical-Disclosure attack (SDA) [4, 11, 6, 5] introduces the class of statistical attacks that focus on the likelihood that a single recipient is in Alice's recipient set. These attacks typically assume some knowledge of the distribution of communications amongst untargeted users, which must remain static during the attack. This approach introduces the possibility of both inclusion and exclusion errors, but results in much more efficient attacks. While both approaches provide advantages, a comparison of their relative effectiveness is beyond the scope of this work, and we will not discuss these attacks further.

One attack of note is the Perfect-Matching-Disclosure-attack (PMDA) [16], which applies statistical attacks to successively weight links between all senders and receivers in a Mix network. This more sophisticated statistical attack builds user communication-pattern profiles to inform its inferences, and allows for tradeoffs between accuracy and speed in disclosing communication links. Again, however, the nature and effectiveness of this attack is largely out of the scope of the current work, which focuses exclusively on provably correct attacks in order to provide a baseline *metric* for anonymity in Mixes.

Structure. Section 2 presents the Mix and attacker model used in this paper. The attack that we present is based on the *ExactHS algorithm* [12, 13] that computes all *minimal hitting sets*, in this case for sets of a user's possible communication partners. The results in this paper enable us to determine the average-case complexity of this algorithm, and thus the average complexity of unambiguously identifying a user's set of communication partners.

Proving the identity of a user's communication partner set is equivalent to proving that all other possible sets of recipients *cannot* be the user's partner set: a *disproof* of these sets. Section 3 presents our theoretical model that describes the number of peers in a possible set of recipients that must be considered in order to disprove it. The average worst case of this number of peers is derived in Sect. 4.

Section 5 applies our analyses to the ExactHS algorithm in order to obtain formulas for the average worst case complexity of identifying peers, and shows how this relates to the required number of observations. To support our theoretical results, we compare our analysis to simulations in Sect. 6. We provide conclusions and ideas for future work in Sect. 7.

2 Mix and Attacker Model

2.1 The Pure Mix Model

We consider the *Pure Mix* technique, as justified in [9], as a generalised and simplified model of practical real-world Mixes.

Our attacker model is that of a *global passive attacker* that observes all communication, but cannot inject, delay or alter messages. From this basis, we will use the following formal model of a pure Mix and information leakage for our analysis.

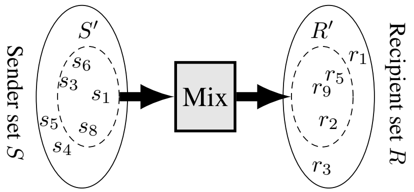


Fig. 1. Mix model

Formal Model of the Pure Mix Technique.

- A communication system consists of a set of senders, S, a set of recipients, R, and a Mix node[1] as shown in Fig. 1. If a sender $s \in S$ communicates with a recipient $r \in R$, then we say that r is a *peer partner* of s, or simply r is a *peer* of s.
- In each *communication round*[2] a subset $S' \subseteq S$ of all senders each send precisely one message to their peer partners. Let $R' \subseteq R$ be the set of intended recipients. The act of sending or receiving a message is not hidden to the attacker, therefore (S', R') represents the information leakage available to an attacker in each round.[3]

[1] S and R represent all users with the ability to send or receive messages in the system.

[2] A communication round consists of the Mix node collecting messages from a fixed number of distinct senders and, after applying the "Mix" protocol, forwarding the collected messages in random order to their intended recipients.

[3] Note that a sender can send to multiple recipients in distinct rounds, but cannot send multiple messages in a single round.

- We call S' the *sender anonymity set*, which is the set of all senders that may have sent a given message. The *recipient anonymity set* R' is the set of all recipients that may have received a message.
- We label the size of the *sender anonymity set*, $|S'|$, as b.
- The size of the *recipient anonymity set*, $|R'|$, is less than or equal to b, as each sender sends exactly one message per round but several senders may communicate with the same recipient. The size of the set of all recipients is $|R| = N$.

Attacker Model. The goal of the attacker is to compute, from a set of observations of traffic, all possible sets of peer partners of a target sender $Alice \in S$. These possibilities form *hypotheses* for the true set of Alice's peer partners, $\mathcal{H}_A$, which is assumed to be a fixed set of size $m = |\mathcal{H}_A|$. We call a peer $r \in \mathcal{H}_A$ an *Alice's peer*; a peer that does not communicate with Alice, $r \in R \setminus \mathcal{H}_A$, is called a *non-peer* and r is simply called *peer* if no distinction is required.

The attacker focuses on revealing Alice's peers by observing only those pairs (S', R'), where Alice participates as a sender. Under this condition we refer to the corresponding recipient set R' as an *observation*, $\mathcal{O}$. The set of all observations collected during t communication rounds is referred to as the *observation set* $\mathcal{OS} = \{\mathcal{O}_1, \ldots, \mathcal{O}_t\}$.

Alice's peer set can be revealed by the *Minimal-Hitting-Set attack* (HS-attack) [10], which computes all hypotheses from the set of observations. These hypotheses correspond to all sets of size m that are *hitting sets* in $\mathcal{OS}$. A hitting set is a set that intersects with all observations in $\mathcal{OS}$. A hitting set is *minimal* if no proper subset of it is a hitting set. The HS-attack succeeds if $\mathcal{OS}$ is consistent with only a single hypothesis. In this case Alice's peer set is unambiguously identified, and is thus the smallest *unique minimal hitting set* of size m. This attack has been proven to require a minimal number of observations to identify $\mathcal{H}_A$[9].

In applying the HS-attack, we assume that the size of Alice's peer set, m, is known, since learning m does not change the complexity class of the attack.

Learning m. The intuition behind our attack is that at least one of Alice's peers must appear in each observation[4], while this does not hold for any other set $\mathcal{H}$, where $\mathcal{H}_A \not\subseteq \mathcal{H}$. Therefore, after a large number of observations, t, Alice's peer set $\mathcal{H}_A$ remains the unique smallest minimal hitting set.

Assume the existence of a set in which $\mathcal{H} \neq \mathcal{H}_A$, where $|\mathcal{H}| < m$ happens to be a unique minimal hitting set. If p is the probability that any peer in $\mathcal{H}$ appears in a random observation, the probability that $\mathcal{H}$ remains a hitting set after t observations decreases according to an exponential function p^t. The probability of learning the wrong set of Alice's peers and the wrong value of m by the HS-attack is therefore negligible even for moderate t.

We can learn m in time $\sum_{m'=1}^{m} O(b^{m'} m' tb) = O(b^m mtb)$ by running the HS-attack for $m' = 1, \ldots, m$ with respect to the same t observations according to equation (2). If $m' < m$, then there will be no hitting sets of size m' and HS-attack thus detects incorrect m'.

Multiple Sending per Round. We assume that each sender sends only one message in each round to simplify the Mix model and our analysis. The HS-attack remains

[4] Recall that an "observation" refers to a round in which Alice participates.

applicable, however, if a sender can send multiple messages per round. This altered model does require slight modifications to the algorithm deployed by HS-attack, and thus a minor modification to the analysis. Due to space limitations, we omit this extended model here. Therefore, investigating the relation between the results from the simple model and from the extended model will be left for future work.

2.2 ExactHS Algorithm

ExactHS [12, 13], described in Alg. 1 determines the *hypotheses* in the Minimal-Hitting-Set attack. Unlike the original HS algorithm proposed in [10], which analyses all $\binom{N}{m}$ hitting sets, ExactHS considers *only* minimal hitting sets and thus drastically reduces computation [12, 13].

The method used by ExactHS to determine hitting sets corresponds to the theoretical model in Sect. 3, allowing us to apply the analyses of Sect. 4 to determine the average case complexity for unambiguously identifying Alice's peer set.

ExactHS recursively computes all minimal hitting sets with respect to the attacker's observation set $\mathcal{OS}$. We use the following notation:

$\mathcal{C}$: Set of at most m suspected[5] peers representing a subset of a possible hitting set. It is initially empty.

$\mathcal{OS}[r]$: Set of observations containing peer r, that is $\{\mathcal{O} \in \mathcal{OS} \mid r \in \mathcal{O}\}$. $|\mathcal{OS}[r]|$ is called the *frequency* of r. $|\mathcal{OS}[r]|$ is 0, if r is not in any observations of $\mathcal{OS}$.

$\mathcal{OS}[\{r_1, \ldots, r_k\}]$: Set of observations containing any $r_1, \ldots, r_k$, that is $\bigcup_{i=1}^{k} \mathcal{OS}[r_i]$.

We now describe in detail the steps taken by ExactHS on a line-by-line basis, as shown in Alg. 1.

Algorithm 1 ExactHS

```
1:  procedure ExactHS($\mathcal{OS}'$, $m'$, $\mathcal{C}$)
2:      if $\mathcal{OS}' = \{\}$ then
3:          return $\mathcal{C}$                                        ▷ $\mathcal{C}$ is a hitting set
4:      else if $m' \geq 1$ then                 ▷ add a peer to $\mathcal{C}$, if $\mathcal{C}$ contains less than $m$ peers
5:          choose $\mathcal{O} \in \mathcal{OS}'$
6:          while $(\{\} \notin \mathcal{OS}') \wedge (\max_{r_1,\ldots,r_{m'}} \{\sum_{l=1}^{m'} |\mathcal{OS}'[r_l]|\} \geq |\mathcal{OS}'|)$ do
7:              choose $r \in \mathcal{O}$                          ▷ $r$ will become element of $\mathcal{C}$
8:              ExactHS ($\mathcal{OS}' \setminus \mathcal{OS}'[r], m' - 1, \mathcal{C} \cup \{r\}$)   ▷ select remaining $(m' - 1)$ peers of $\mathcal{C}$
9:              $\mathcal{OS}' \leftarrow \bigcup_{\mathcal{O}_l \in \mathcal{OS}'} \{\mathcal{O}_l \setminus \{r\}\}$          ▷ remove $r$ in all observ. of $\mathcal{OS}'$
10:             $\mathcal{O} \leftarrow \mathcal{O} \setminus \{r\}$          ▷ do not choose $r$ in this recursion level again
```

The computation of the minimal hitting sets is initially invoked by calling the algorithm $ExactHS(\mathcal{OS}, m, \mathcal{C})$. For ease of reference we denote sets computed in the i-th level of recursion with the subscript i. Thus $\mathcal{C}_i, \mathcal{OS}'_i$ represents the sets calculated by ExactHS at the i-th recursive call of the algorithm. At each level of recursion in the algorithm, recursing to the next level extends the current set of peers $\mathcal{C}_i$ by exactly one peer, r, at Line 7 of Alg. 1. This peer is chosen from a designated observation $\mathcal{O} \in \mathcal{OS}'_i$ determined by the algorithm in Line 5. Thus: $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{r\}$.

[5] During execution, $\mathcal{C}$ either becomes a minimal hitting set, or it will be proved not to be a subset of any minimal hitting sets.

$\mathcal{OS}'_{i+1}$, defined at Line 8, results from removing all observations intersecting with r in $\mathcal{OS}'_i$; we need only focus on those observations that have not already been evaluated by $\mathcal{C}_i \cup \{r\}$ in earlier recursive calls.

If, at Line 2, the algorithm detects that all remaining observations in $\mathcal{OS}'_{i+1}$ intersect with $\mathcal{C}_{i+1}$, $\mathcal{C}_{i+1}$ is proven to be a hitting set, and ExactHS will not compute any set containing this $\mathcal{C}_{i+1}$ in the future. Line 6 will also detect if $\mathcal{C}_{i+1}$ is not a subset of any hitting set; this also causes any set containing it to be ignored in future levels of recursion. We refer to sets excluded by the algorithm as *finalised* sets.

After a selection of r in recursion level i, ExactHS removes, at Line 9, r from all observations of $\mathcal{OS}'_i$ and, at Line 10, from the designated observation $\mathcal{O}$. The algorithm thus extends $\mathcal{C}_i$ with a new peer r'.

ExactHS stops choosing new peers if it detects, at Line 6, that the cumulative frequency of all remaining m' peers is lower than the number of remaining observations; that is, if $\max_{r_1,\ldots,r_{m'}}\{\sum_{l=1}^{m'} |\mathcal{OS}'[r_l]|\} \ngeq |\mathcal{OS}'|$. Further explanations are in Sect. 5.

Complexity. ExactHS creates a finalised set $\mathcal{C}$ by starting with an empty set $\mathcal{C} = \{\}$ and adding the i-th peer to $\mathcal{C}$ in the *choice phase* of the i-th level of recursion, starting at line 6 of the algorithm. The number of recursive invocations of the choice phase is bounded from above by m.

In each choice phase there are at most b possible choices of a peer r_i, as only peers $r_1, \ldots, r_b$ of a fixed observation $\mathcal{O}$ can be selected. Due to the bound m for the number of recursive invocations of the choice phase, and the bound b for the number of choices in each phase, the algorithm computes at most b^m minimal hitting sets. This bound is tight, and determines the *worst case runtime complexity* $O(b^m mtb)$ of ExactHS, as proved in [12, 13]. $t = |\mathcal{OS}|$ is the number of observations collected by the attacker and mtb is the effort required to construct one finalised set.

Let us consider a concrete example with the parameters $m = 2, b = 2$, the Alice's peer set $\mathcal{H}_A = \{1, 2\}$ and the observations $\{1, 3\}, \{2, 4\}$. Here, ExactHS would compute $b^m = 4$ minimal hitting sets, namely: $\{1, 2\}, \{1, 4\}, \{3, 2\}, \{3, 4\}$.

In general, however, if ExactHS were to prove at level $x \leq m$ that a set is, or is not, a hitting set, then the number of finalised sets computed by ExactHS is bounded from above by (1) and the runtime is bounded by (2). The space complexity of ExactHS, as proved in [12], is $O((x + 1)tb)$, which is linear.

$$\text{Maximal number of sets: } b^x \qquad (1) \qquad\qquad \text{Runtime: } O(b^x mtb) \qquad (2)$$

Hitting Set Structure. In order to make a more detailed analysis of the ExactHS algorithm, we partition the set of minimal hitting sets of size m. Let $\mathcal{H}$ be a minimal hitting set where $|\mathcal{H}| = m$. We therefore assign it to one of the $m + 1$ disjoint classes $\mathfrak{H}_0, \ldots \mathfrak{H}_m$ with the following structure:

$$\mathfrak{H}_0 = \{\mathcal{H}_A\} \qquad \text{and} \qquad \mathfrak{H}_j \subseteq (R \setminus \mathcal{H}_A)^j \times \mathcal{H}_A^{m-j} \ \text{, for } j \leq m \ . \qquad (3)$$

A minimal hitting set $\mathcal{H}$ belongs to the class $\mathfrak{H}_j$ ($\mathcal{H} \in \mathfrak{H}_j$), if and only if it contains exactly $(m - j)$ distinct Alice's peers and j distinct non-peers. The class $\mathfrak{H}_0$ contains exactly one set, Alice's peer set $\mathcal{H}_A$, and $\mathfrak{H}_m$ represents minimal hitting sets consisting of only non-peers of Alice.

3 Estimation of the Number of Covered Observations

This section focuses on the complexity theoretic security of the Mix. We therefore assume that the observations in $\mathcal{OS}$ collected by the attacker provide sufficient information for the unambiguous identification of Alice's peer set $\mathcal{H}_A$. The main question we wish to answer is:

1. What is the average time complexity required to prove that $\mathcal{H}_A$ is a unique minimal hitting set?

Proving uniqueness of $\mathcal{H}_A$ in $\mathcal{OS}$ is hard as there are exponentially many possible hitting sets $\mathcal{H} = \{r_1, \ldots, r_m\} \neq \mathcal{H}_A$ that need to be disproved with respect to $\mathcal{OS}$. To mitigate this problem we avoid disproving all individual sets $\mathcal{H}$ answering the following question:

2. How many peers in $\mathcal{H}$ must be chosen to prove that $\mathcal{H}$ is not a hitting set?

We choose peers $r_1, \ldots, r_x \in \mathcal{H}$ by determining all observations including $\mathcal{C} = \{r_1, \ldots, r_x\}$, which we denote $\mathcal{OS}[\mathcal{C}]$. Given these chosen peers we know the observations $\mathcal{OS} \setminus \mathcal{OS}[\mathcal{C}]$ that have not yet been considered. We refer to the remaining peers in $\mathcal{H}$ as *non-chosen*. Whilst a peer is non-chosen, we do not known which observations contain that peer.

Assume, without loss of generality, that after choosing these x peers in $\mathcal{C} \subseteq \mathcal{H}$ we know that $\mathcal{H}$ cannot be a hitting set, because the cumulative frequency of the $(m - x)$ most frequent peers in $\mathcal{OS} \setminus \mathcal{OS}[\mathcal{C}]$ is less than $|\mathcal{OS} \setminus \mathcal{OS}[\mathcal{C}]|$. In this case we prove not only that $\mathcal{H}$ is not a hitting set, but also that any superset $\mathcal{H}'$ of $\mathcal{C}$ cannot be a hitting set, where $|\mathcal{H}'| = m$.

In general, if we know that every set can be disproved after choosing on average x peers, then using (2) the average runtime complexity of ExactHS is approximated by $O(b^x mtb)$, which answers our first question. A more detailed justification and discussion of this complexity is provided in Sect. 5.

The rest of this section provides the theoretical model for answering the question of how many peers in $\mathcal{H}$ must be chosen to prove that $\mathcal{H}$ is not a hitting set. The answer will be derived in Sect. 4.

3.1 Potential

In this section we introduce the definition of the *potential*: our estimation of the number of distinct observations covered by a set $\mathcal{H}$ in a given observation set $\mathcal{OS}$. This value allows us to estimate the number of peer choices required to disprove a set, and thus to understand the complexity of ExactHS. Note that this "estimation" is part of our analysis of the complexity, and does not affect the exactness of the attack itself.

We assume without loss of generality that all considered sets are of the structure $\mathcal{H} = \{r_1, \ldots, r_x, r_{x+1}, \ldots, r_m\}$. Each r_i represents a distinct peer, and the number of peers is $|\mathcal{H}| = m$. The first $0 \leq x \leq m$ peers $r_1, \ldots, r_x$ are always chosen, while the remaining $(m - x)$ peers are non-chosen. The potential of $\mathcal{H}$ is denoted by $Po(\mathcal{H})$.

$$\begin{aligned} Po(\mathcal{H}) = {} & |\mathcal{OS}[\{r_1, \ldots, r_x\}]| + |\mathcal{OS}[r_{x+1}] \setminus \mathcal{OS}[\{r_1, \ldots, r_x\}]| + \ldots \\ & + |\mathcal{OS}[r_m] \setminus \mathcal{OS}[\{r_1, \ldots, r_x\}]| \end{aligned} \tag{4}$$

There are two extreme cases. If all peers are chosen, then the potential is the number of observations covered by $\mathcal{H}$. If all peers are non-chosen, then the potential is the cumulative frequency of the peers of $\mathcal{H}$ in $\mathcal{OS}$. The more peers chosen in $\mathcal{H}$, the more accurately the potential represents the number of distinct observations intersecting with $\mathcal{H}$. $Po(\mathcal{H})$ thus never underestimates the number of observations intersecting with $\mathcal{H}$.

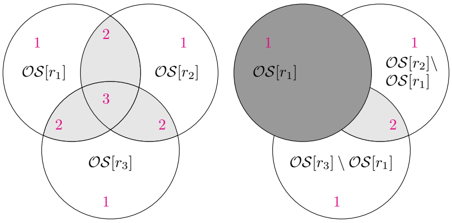

1
2
1
$\mathcal{OS}[r_1]$
$\mathcal{OS}[r_2]$
3
2
2
$\mathcal{OS}[r_3]$
1
1
1
$\mathcal{OS}[r_1]$
$\mathcal{OS}[r_2]\setminus$ $\mathcal{OS}[r_1]$
2
$\mathcal{OS}[r_3] \setminus \mathcal{OS}[r_1]$
1

Fig. 2. *Left:* Overestimation by $Po(\{r_1, r_2, r_3\})$, where all peers r_1, r_2, r_3 are non-chosen. *Right:* Overestimation by $Po(\{r_1, r_2, r_3\})$, where r_1 is chosen.

Overestimations are observations that are covered by more than one non-chosen peers in $\mathcal{H}$ as illustrated by the leftmost diagram in Fig. 2. We will analyse the overestimation of the potential, since it enables us to conclude how many peers in $\mathcal{H} \neq \mathcal{H}_A$ need to be chosen to disprove it.

Potential: All Peers Non-Chosen. The set of observations covered by r_i is represented by a circle around $\mathcal{OS}[r_i]$ for $i = 1, 2, 3$ in the left-hand picture in Fig. 2. The grey area represents those observations that are covered by at least two peers r_i, r_j for $i \neq j$. The number in the area shows the number of times observations in that area are counted in the potential. In this example $\mathcal{H} = \{r_1, r_2, r_3\}$ and we can see on the left picture how $Po(\mathcal{H})$ overestimates $|\mathcal{OS}[\mathcal{H}]|$, which is the number of observations covered by $\mathcal{H}$. The overestimation is caused by those observations that are covered by more than one of the peers r_1, r_2, r_3. The exact number of observations covered by $\mathcal{H}$ in the left picture in Fig. 2 can be computed by the inclusion exclusion formula.

$$\begin{aligned}|\mathcal{OS}[\mathcal{H}]| = {} & |\mathcal{OS}[r_1]| + |\mathcal{OS}[r_2]| + |\mathcal{OS}[r_3]| - |\mathcal{OS}[r_1] \cap \mathcal{OS}[r_2]| - \\ & |\mathcal{OS}[r_1] \cap \mathcal{OS}[r_3]| - |\mathcal{OS}[r_2] \cap \mathcal{OS}[r_3]| + |\mathcal{OS}[r_1] \cap \mathcal{OS}[r_2] \cap \mathcal{OS}[r_3]|\end{aligned}$$

As all peers in $\mathcal{H}$ are non-chosen, $Po(\mathcal{H}) = |\mathcal{OS}[r_1]| + |\mathcal{OS}[r_2]| + |\mathcal{OS}[r_3]|$. For the sake of simplicity we derive the following estimation from the equation above.

$$Po(\mathcal{H}) \leq |\mathcal{OS}[\mathcal{H}]| + |\mathcal{OS}[r_1] \cap \mathcal{OS}[r_2]| + |\mathcal{OS}[r_1] \cap \mathcal{OS}[r_3]| + |\mathcal{OS}[r_2] \cap \mathcal{OS}[r_3]|$$

Potential: General Case. The case when one peer r_1 is chosen while the other peers in $\mathcal{H}$ are non-chosen is illustrated by the right-hand picture of Fig. 2. By the definition of $Po(\{r_1, r_2, r_3\})$ in (4), choosing r_1 causes all observations containing it, represented by the dark circle, to be removed in the frequency consideration of the non-chosen peers. In this case $Po(\mathcal{H})$ overestimates $|\mathcal{OS}[\mathcal{H}]|$ by double-counting the grey area that represents observations that are covered by r_2 and r_3 but not by r_1. For simplicity we use the following estimation of $Po(\mathcal{H})$:

$$Po(\mathcal{H}) \leq |\mathcal{OS}[\mathcal{H}]| + |\mathcal{OS}[r_2] \cap \mathcal{OS}[r_3]| \ .$$

In general, if $0 \leq x \leq m$ peers $\{r_1, \ldots, r_x\}$ of $\mathcal{H} = \{r_1, \ldots, r_m\}$ are chosen, then the overestimation of the number of covered observations result from the non-chosen peers r_k, r_l for $x < k, l \leq m$. The overestimation is bounded by the size of the $\binom{m-x}{2}$ pairwise intersections $\mathcal{OS}[r_k] \cap \mathcal{OS}[r_l]$. This results in the following simplified estimation of the potential for the general case:

$$Po(\mathcal{H}) \leq |\mathcal{OS}[\mathcal{H}]| + \sum_{x<k,l\leq m;\, k\neq l} |\mathcal{OS}[r_k] \cap \mathcal{OS}[r_l]| \ . \tag{5}$$

Overestimation by Potential. In order to distinguish the effect of Alice's peers and non-peers to $Po(\mathcal{H})$, each peer $r \in \mathcal{H}$ is relabelled n for non-peers, and a for Alice's peer. Without loss of generality, every $\mathcal{H} \in \mathfrak{H}_j$, where $|\mathcal{H}| = m$ from now on has the following structure:

$$\mathcal{H} = \{\underbrace{n_1, \ldots, n_{x_1}, a_1, \ldots, a_{x_2}}_{x \text{ chosen peers}} \quad , \quad \underbrace{n_{x_1+1}, \ldots, n_j, a_{x_2+1}, \ldots, a_{m-j}}_{(m-x) \text{ non-chosen peers}}\} \ .$$

The number of chosen peers is $x = x_1 + x_2$, where $x_1 \leq j$ and $x_2 \leq m - j$. The variable j denotes the number of non-peers in hitting sets of the structure $\mathfrak{H}_j$. We still use the notation r_i to address the i-th peer in $\mathcal{H}$ if distinction is not important. As before, the first x peers $r_1, \ldots, r_x \in \mathcal{H}$ are chosen, while the remaining $(m-x)$ peers are non-chosen. We define $\mathcal{H}^{+A} = \mathcal{H} \cap \mathcal{H}_A$ as the subset containing only Alice's peers and $\mathcal{H}^{-A} = \mathcal{H} \setminus \mathcal{H}_A$ as the subset consisting of only non-peers.

The following estimations for $|\mathcal{OS}[\mathcal{H}]|$ and $|\mathcal{OS}|$ will be used next in inequality (9):

$$|\mathcal{OS}[\mathcal{H}]| \leq |\mathcal{OS}[\mathcal{H}^{+A}]| + \sum_{n\in\mathcal{H}^{-A}} |\mathcal{OS}[n] \setminus \mathcal{OS}[\mathcal{H}^{+A}]| \tag{6}$$

$$|\mathcal{OS}| \geq |\mathcal{OS}[\mathcal{H}^{+A}]| + \sum_{a\in(\mathcal{H}_A\setminus\mathcal{H}^{+A})} |\underbrace{\mathcal{OS}[a] \setminus \mathcal{OS}[\mathcal{H}_A \setminus \{a\}]}_{\text{observ. containing } a \text{ exclusively}}| \ . \tag{7}$$

An observation contains Alice's peer $a \in \mathcal{H}_A$ *exclusively* [9], if it does not contain any other peers of Alice.

We now mathematically formulate our earlier question; that is: how many peers must be chosen in order to prove that $\mathcal{H} \neq \mathcal{H}_A$ is not a hitting set in $\mathcal{OS}$? This is simple using the potential, as it estimates the number of observations covered by $\mathcal{H}$ in $\mathcal{OS}$. If $Po(\mathcal{H}) < |\mathcal{OS}|$ then $\mathcal{H}$ is clearly not a hitting set. On the other hand, if $Po(\mathcal{H}) \geq |\mathcal{OS}|$ then we must choose more peers in $\mathcal{H}$ for the disproof. The latter is formulated below. Inequality (9) then results from applying (5) and (6) on $Po(\mathcal{H})$ and (7) on $|\mathcal{OS}|$.

$$\begin{aligned}
&0 \leq Po(\mathcal{H}) - |\mathcal{OS}| &(8)\\
&\leq \sum_{x_2<k,l\leq m-j;\, k\neq l} |\mathcal{OS}[a_k] \cap \mathcal{OS}[a_l]| + \sum_{x_2<k\leq m-j;\, x_1<l\leq j} |\mathcal{OS}[a_k] \cap \mathcal{OS}[n_l]| \\
&+ \sum_{x_1<k,l\leq j;\, k\neq l} |\mathcal{OS}[n_k] \cap \mathcal{OS}[n_l]| + \sum_{n\in\mathcal{H}^{-A}} |\mathcal{OS}[n] \setminus \mathcal{OS}[\mathcal{H}^{+A}]| \\
&- \sum_{a\in(\mathcal{H}_A\setminus\mathcal{H}^{+A})} |\mathcal{OS}[a] \setminus \mathcal{OS}[\mathcal{H}_A \setminus \{a\}]| &(9)
\end{aligned}$$

For simplicity we restrict our analysis to those cases where the probability that a particular peer $r \in \mathcal{H}$ is contacted by a sender other than Alice, within a given observation $\mathcal{O}$, is significantly lower than the probability that Alice's peer is contacted by Alice. This allows us to ignore the possibility that some pair of peers $r_k, r_l \in \mathcal{H}$ is contacted by senders other than Alice in the same $\mathcal{O}$. This allows us to ignore counting the observations described below in (9):

$$\{\mathcal{O} \in \mathcal{OS}[r_k] \cap \mathcal{OS}[r_l] \mid r_k, r_l \in \mathcal{H} \text{ chosen by non-Alice senders in } \mathcal{O}\} \ . \tag{10}$$

We call the resulting simplified estimation of (9) the *difference* function $D(x, x_1, x_2, j)$:

$$\sum_{x_2<k,l\leq m-j;\, k\neq l} |\mathcal{OS}[a_k] \cap \mathcal{OS}[a_l]| + \sum_{x_2<k\leq m-j;\, x_1<l\leq j} |\mathcal{OS}[a_k] \cap \mathcal{OS}[n_l]| \ + \sum_{n\in\mathcal{H}^{-A}} |\mathcal{OS}[n] \setminus \mathcal{OS}[\mathcal{H}^{+A}]| - \sum_{a\in(\mathcal{H}_A\setminus\mathcal{H}^{+A})} |\mathcal{OS}[a] \setminus \mathcal{OS}[\mathcal{H}_A \setminus \{a\}]| \ . \tag{11}$$

4 Number of Peer Choices for a Disproof

4.1 Expectation of the Difference

In this section we compute the expectation of the difference function for a simplified communication model of Alice and the other senders, which we call *uniform communication*.

In this model the *cumulative communication* of all other senders leads to a uniform *background distribution* of communication with the peers such that, without Alice's communication, each peer $r \in R$ appears with the same *cumulative probability* of P_{nA} in an observation. Therefore each sender can select its peer according to an arbitrary distribution provided that $\forall r \in R : P(r \in \mathcal{OS}) = P_{nA}$, where $P(r \in \mathcal{OS})$ denotes the probability that r appears in the observations $\mathcal{OS}$ of the attacker without considering Alice's communication.

To simplify our analysis we assume that, in every round, each of the $(b-1)$ non-Alice senders choose their peers uniformly from the set R of N recipients with probability $\frac{1}{N}$. Thus, for every peer $r \in R$ its cumulative probability of appearing in an observation is $P_{nA} = 1 - (\frac{N-1}{N})^{b-1}$. We further assume that Alice contacts one of her m peers $a \in \mathcal{H}_A$ in each round, chosen according to the uniform distribution with the probability of $P_A = \frac{1}{m}$.

$$E_1(x, x_1, x_2, j) = t\binom{m-j-x_2}{2}\frac{2}{m}P_{nA}$$

$$E_2(x, x_1, x_2, j) = t(j-x_1)(m-j-x_2)\frac{1}{m}P_{nA}$$

$$E_3(x, x_1, x_2, j) = tj\frac{j}{m}P_{nA}$$

$$E_4(x, x_1, x_2, j) = tjm^{-1}\left(1-(m-1)N^{-1}\right)^{b-1}$$

The difference described by equation (11) is generic and can be analysed with respect to arbitrary communication models. It is sufficient, however, to consider uniform communications, and Sect. 5.1 will show a mapping from non-uniform to uniform communications that provide analytical bounds valid for both instances. For the sake of simplicity, all remaining analysis in this paper will refer to uniform communication unless otherwise stated.

The equations above represent the expectation of the four terms of equation (11), where the number of observations collected by the attacker is $t = |\mathcal{OS}|$.

The terms following t in E_1, E_2, E_3, E_4 are significant, and we discuss these here.

E_1: For Alice's peers $a_k, a_l \in \mathcal{H}^{+A}$, where $a_k \neq a_l$, the probability that Alice contacts a_k and one of the other $(b-1)$ senders contact a_l in an observation is $\frac{1}{m} P_{nA}$. Due to symmetry, the probability that a_k and a_l appear in an observation is $\frac{2}{m} P_{nA}$. This is multiplied by the number of possible pairs of non-chosen Alice's peers $\binom{m-j-x_2}{2}$.

E_2: For peers $a_k \in \mathcal{H}^{+A}$ and $n_l \in \mathcal{H}^{-A}$, the probability that Alice contacts a_k and one of the other $(b-1)$ senders contacts n_l is $\frac{1}{m} P_{nA}$. The factor $(m-j-x_2)$ shows the number of non-chosen Alice's peers a_k while the factor $(j-x_1)$ represents the number of non-chosen non-peers n_l.

E_3: Let $a_1, \ldots, a_j \in (\mathcal{H}_A \setminus \mathcal{H})$ be the j Alice's peers that are not in $\mathcal{H}$. The probability that a given non-peer $n_k \in \mathcal{H}^{-A}$ appears in an observation where Alice contacts one of $a_1, \ldots, a_j$ is $\frac{j}{m} P_{nA}$. The final factor j accounts for the fact that there are j non-peer n_k in $\mathcal{H}^{-A}$.

E_4: Alice's peer $a \in (\mathcal{H}_A \setminus \mathcal{H})$ is exclusive in an observation if Alice contacts a and none of the other $(b-1)$ senders contact any of the peers $a' \in (\mathcal{H}_A \setminus \{a\})$. The probability that a is exclusive is therefore $\frac{1}{m}\left(1 - \frac{m-1}{N}\right)^{b-1}$. The factor j accounts for this exclusivity probability for the j Alice's peers $a_1, \ldots, a_j \in (\mathcal{H}_A \setminus \mathcal{H})$ not appearing in $\mathcal{H}$.

Combining these expectations results in an expectation, $E_D(x, x_1, x_2, j)$, for the difference function $D(x, x_1, x_2, j)$ of:

$$\frac{t}{m}\left[((m-x-1)(m-j-x_2) + j^2)P_{nA} - j\left(1 - \frac{m-1}{N}\right)^{b-1}\right] . \qquad (12)$$

4.2 Average Number of Peer Choices

We obtain the average number of peer choices to disprove a set $\mathcal{H}$ by determining the value of x such that the expectation of the difference is 0. By detailed analysis of the property of E_D (in Appendix A) , we gain simple descriptions of assertions about the limits of the number of peer choices. These limits are summarised here.

Upper Bound of Average Worst Case Number of Peer Choices. If $\frac{N}{b-1} \geq 3m - 1$ and N, b, m is fixed, then the *upper bound of the average worst case number of peer choices* is x_{uw}. This value provides an estimate of the average maximal number of peer choices for a disproof, approaching the bound from above. This can be reformulated to determine the parameters N, b, m, such that a particular bound x_{uw} is obtained by (14).

$$x_{uw} = m - \frac{1}{2} - \sqrt{\frac{N}{b-1} - m + \frac{1}{4}} \quad \text{, where } x_{uw} \leq m \qquad (13)$$

$$b = \frac{N}{m^2 - 2m x_{uw} + x_{uw}^2 + x_{uw}} + 1 \qquad (14)$$

For full proofs of these results, see Appendix A.1.

5 Runtime Complexity

We have now determined how many peers must be chosen in order to disprove a hypothesis set, and so can answer our original question: what is the average complexity to identify unambiguously Alice's peer set $\mathcal{H}_A$?

The ExactHS algorithm reduces the space of sets that must be disproved to identify $\mathcal{H}_A$ by two strategies. Firstly, ExactHS reduces the search space to consider only minimal hitting sets, which is sufficient to identify $\mathcal{H}_A$ in [12, 13]. Secondly, it deploys the estimation of the number of covered observations based on the potential and implements the difference function (Alg. 1 Lines 6, 8). In Alg. 1 the set $\mathcal{C}$ represents $(m-m')$ chosen peers and $\{r_1, \ldots, r_{m'}\}$ represents hypothetical non-chosen peers. The algorithm constructs $|\mathcal{OS}'| = |\mathcal{OS} \setminus \mathcal{OS}[\mathcal{C}]| = |\mathcal{OS}| - |\mathcal{OS}[\mathcal{C}]|$ and $\sum_{l=1}^{m'} |\mathcal{OS}'[r_l]| = Po(\mathcal{C} \cup \{r_1, \ldots, r_{m'}\}) - |\mathcal{OS}[\mathcal{C}]|$, where $\mathcal{OS}$ is the initial set of observations of the attacker, which is equivalent to Equation (8). This allows direct application of the bounds derived in the last section to ExactHS.

The worst case number of peer choices, x, to disprove a set in the last section therefore corresponds to the worst case number of recursion levels x invoked in ExactHS.

To avoid significantly overestimating the strength of the system, we assume that the variance of the average number of peer choices x is negligible. (1) therefore results in an average number of finalised sets computed by ExactHS to identify $\mathcal{H}_A$ of: b^x .

To obtain the corresponding runtime complexity, the last term must be multiplied by tbm, resulting in $O(b^x tbm)$, and reaches a worst case complexity of $O(b^m tbm)$ when $x = m$. The following analysis consequently refers only to the number of finalised sets computed by ExactHS.

5.1 Upper Bound of Average Worst Case

The upper bound of the average worst-case complexity results from the upper bound of the average worst-case number of peer choices x_{uw} determined by (13). Applying that to b^x we derive the upper bound for the average maximal number of finalised sets computed by ExactHS for the unambiguous identification of $\mathcal{H}_A$:

$$b^{m-\frac{1}{2}-\sqrt{\frac{N}{b-1}-m+\frac{1}{4}}} \approx b^{m-\frac{1}{2}-\sqrt{\frac{1}{P_{nA}}-m+\frac{1}{4}}} \quad . \tag{15}$$

From the relations $P_{nA} = 1 - (1 - \frac{1}{N})^{b-1} \approx \frac{b-1}{N}$ and $P_A = \frac{1}{m}$ we conclude that:

- If every peer not contacted by Alice is at least as likely to appear in an observation as peers contacted by Alice, the average worst case complexity roughly equals the worst case complexity $O(b^m tbm)$. That is if $P_{nA} = \frac{1}{m-\frac{1}{4}}$.
- The average worst case complexity becomes linear $O(tbm)$ if every peer not contacted by Alice appears in observations with a probability close to $\frac{1}{m^2}$.

Non-uniform Communication. The analyses above apply to non-uniform background distribution by setting $P_{nA} = \max_{r \in R'}\{P(r \in \mathcal{OS})\}$ in (15). This maps an instance with non-uniform background communication and parameters $N' = |R'|, b, m$ to an instance of uniform communication with parameters $N = |R| = \frac{b-1}{P_{nA}}, b, m$, where R' and R is the recipient set of the first and second instances respectively. The average case complexity of the latter is at least as high as the former, as in uniform communication each of the $N = |R|$ peers appears with a probability of P_{nA} in an observation, while a smaller number of most likely peers of R' appears with that probability in non uniform communication.

Note that the cumulative background probability of the peers can be estimated in the global passive attacker model by considering observations in which Alice does not participate, enabling attackers, Mix providers and users to determine *a priori* the average worst case complexity of ExactHS for a distinct number of Alice's peer partners m.

We assume Alice's communication to be uniform when deriving the average case complexity not only for simplicity, but also because simulation reveals that it is the worst case for the average run time complexity. Informally, in a non-uniform communication some Alice's peers are even more statistic signification than the non-peers. Thus, making ExactHS focus on the most frequent peers reduces the hypothesis space and average time complexity. A formal proof of this is forthcoming.

Relation to Least Number of Observations by ExactHS. To determine efficiently the number of observations required by the ExactHS-attack, we can apply the algorithm to compute the lower bound of the HS-attack based on the *2x-exclusivity criteria* [9, 12] or use the mathematical analysis provided by [13].

We use here the formula for the least number of observations t to identify $\mathcal{H}_A$ by the minimal hitting set attack [13]. It provides, in contrast to the 2x-exclusivity formula in [9], a closed formula that directly represents the effect of Mix parameters.

$$t \approx m\big(\ln(b-1) - \ln(2^{1/m} - 1)\big)\big(1 - mN^{-1}\big)^{1-b} \tag{16}$$

This formula shows that ExactHS can reveal Alice's peer set after a number of observations t that is sub-exponential with respect to N, b, m. The number of observations for the identification of Alice's peers is thus an insufficient metric for the strength of the Mix, and we need to consider the average case complexity of ExactHS. Section 6 compares the theoretical results of this paper with attacks on simulated data.

Countermeasure against Attack. To prevent the ExactHS attack in practice, Mix providers can adjust the average case complexity $O(b^x tbm)$ to be close to the worst case complexity, such that $x = m - \epsilon$ for fixed security parameters m and ϵ chosen by the provider. To obtain this, the batch size b can be determined with respect to N, m, x according to equation (14). By doing so, applying our attack against users who uniformly contact $m' \geq m$ peer partners requires a time complexity bounded by $O(b^{m'-\epsilon} tbm')$. Users with $m' < m$ peer partners, however, or non-uniform communication should be aware that revealing their peer partners will be faster than $O(b^{m'-\epsilon} tbm')$.

We have derived a formula for the lower bound of the average case complexity of ExactHS, which could be used to adjust the least average time required for an attack on a Mix, however we omit this due to space limitations.

Note that ExactHS and statistical attacks are based on very different principles. Therefore, Mix configurations that are susceptible to ExactHS are not necessarily susceptible to the statistical attacks and vice versa. While it is outside of the scope of this paper, a comparison of the effectiveness of both classes of attacks with respect to different Mix configurations and countermeasures would be an interesting topic for future research.

6 Simulation

To support our mathematical analysis, we now show the ExactHS algorithm applied to randomly generated observations. These observations are generated under the uniform communication model of Sect. 4.1, which is chosen to allow direct comparison between the simulation and our theoretical results.

An attack is *successful* if ExactHS can unambiguously identify Alice's peer set $\mathcal{H}_A$; the simulation generates new observations until this occurs. The average number of observations required by an attack is therefore the mean of the number of observations of all successful attacks. To ensure that our results are statistically significant, experiments were repeated until 95% of the results fall within 5% of the empirically observed mean.

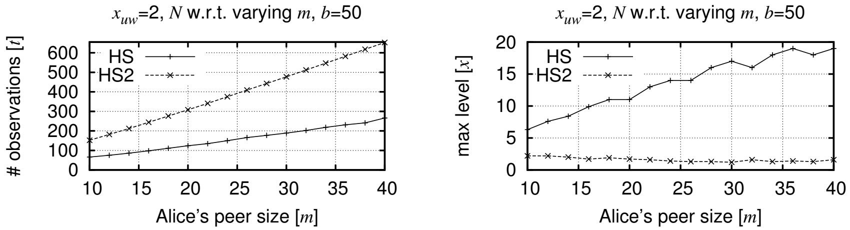


Fig. 3. Parameters N, b, m, where $x_{uw} = 2$. *Left:* Number of observations when ExactHS succeeds. *Right:* Empirical recursion level for disproof by ExactHS.

Average Worst Case. To demonstrate that our analysis closely predicts the empirical average worst case complexity of ExactHS, we apply attacks on observations of a Mix with parameters N, b, m that are chosen according to (14), where $x_{uw} = 2$. It is therefore expected that ExactHS succeeds on those configurations within a polynomial run time of $O(b^2 tbm)$, while its average worst case recursion level is bounded by 2.

Figure 3 shows the result of our simulation for fixed $b = 50$. The value of N is determined by (14) given fixed $x_{uw} = 2$; m values are shown on the x-axis. The value of N ranges from 3200 for $m = 10$ to 70000 for $m = 40$.

As the attack requires very few observations to succeed, the empirical probability distribution of the peers of the non-Alice senders at the termination of the attack strongly diverge from the function $P_{nA} \approx \frac{b-1}{N}$ from which they are drawn[6].

[6] Assume for example that $P_{nA} = 1/400$, but the attack succeeds after $|\mathcal{OS}| = 100$ observations, then the probability of each peer included by an observation in $\mathcal{OS}$ exceeds P_{nA} by at least a factor of four.

Due to the law of large numbers, this side effect diminishes for large number of observations. We therefore consider the application of ExactHS where the number of observations is twice that required by (16). This is shown in the graphs by the line labelled (HS2). This doubling is simply to aid demonstration of our results by reducing the side effects due to the small number of observations.

The left plot shows on the y-axis the average number of observations to identify Alice's peer set unambiguously. The line (HS) represents the mean of the least possible number of observations required by ExactHS in an information theoretic sense. The line (HS2) shows the number of observations which corresponds to twice the value of (16).

The right plot shows on the y-axis the average worst case level required to disprove a set by ExactHS under the conditions represented by the lines (HS) and (HS2). The line (HS) shows that the level is significantly higher than x_{uw} if ExactHS identifies $\mathcal{H}_A$ with the information theoretic minimal number of observations. This is due to the probabilities of many non-peers exceeding P_{nA} due to a low number of observations. With more observations, as in (HS2), we can see that the average worst case number of required peer choices is about x_{uw} for all selected N, b, m as predicted by (14). Collecting even more additional observations when applying ExactHS does not noticeably change the worst case number of peer choices.

7 Conclusion

Previous non-statistical analyses of Mixes have been based almost exclusively on the least number of observations for an attack, and on the fact that the unambiguous identification of Alice's peer set requires the solution of an NP-complete problem.

This paper is the first presentation, to our knowledge, of a detailed complexity-theoretic analysis of the problem of identifying a user's peer set beyond the worst case complexity determined by the NP-completeness of the underlying problem. We achieve this by contributing closed formulas that determine the average case complexity with respect to the Mix parameters. These theoretical results are further supported by simulations.

It is clear from our results that the identification of Alice's peers in a Mix network, whilst being intractable in the worst case, contains a broad range of realistic Mix configurations that are polynomially solvable. These configurations are serious threats for anonymity that can now be identified by our results (13), (15). Our analyses enable further to identify those configurations that are solvable only in exponential time by ExactHS, allowing for an increase in the anonymity of these systems.

In order to gain the average case complexity of the system, we employ the most efficient known algorithm that provides an exact result. Whilst the possibility exists that a more efficient algorithm could be discovered[7], our results are the first to provide an analysis of this form.

In the future, we intend to extend the analysis in this work to more complex and real-world Mix models. It is hoped that this will allow us to understand the effect that different mixing strategies have on anonymity. In a wider context, our analyses are concerned with the identification of average polynomial-time-solvable instances of an

[7] As is possible with, for example, the prime factorisation algorithms employed in cryptanalysis.

NP-complete problem. The results presented here may therefore be of use in identifying average polynomial-time instances of other interesting NP-complete problems, which would have wider applications beyond the restricted scope of security and privacy.

References

[1] Agrawal, D., Kesdogan, D., Penz, S.: Probabilistic Treatment of MIXes to Hamper Traffic Analysis. In: IEEE Symposium on Security and Privacy, pp. 16–27 (2003)

[2] Berthold, O., Langos, H.: Dummy traffic against long term intersection attacks. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 110–128. Springer, Heidelberg (2003)

[3] Chaum, D.L.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24(2), 84–88 (1981)

[4] Danezis, G.: Statistical Disclosure Attacks: Traffic Confirmation in Open Environments. In: Proceedings of Security and Privacy in the Age of Uncertainty, pp. 421–426 (2003)

[5] Danezis, G., Diaz, C., Troncoso, C.: Two-sided statistical disclosure attack. In: Borisov, N., Golle, P. (eds.) PET 2007. LNCS, vol. 4776, pp. 30–44. Springer, Heidelberg (2007)

[6] Danezis, G., Serjantov, A.: Statistical Disclosure or Intersection Attacks on Anonymity Systems. In: Fridrich, J. (ed.) IH 2004. LNCS, vol. 3200, pp. 293–308. Springer, Heidelberg (2004)

[7] Danezis, G., Troncoso, C.: Vida: How to use bayesian inference to de-anonymize persistent communications. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 56–72. Springer, Heidelberg (2009)

[8] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1990)

[9] Kesdogan, D., Agrawal, D., Pham, V., Rauterbach, D.: Fundamental Limits on the Anonymity Provided by the Mix Technique. In: IEEE Symposium on Security and Privacy (2006)

[10] Kesdogan, D., Pimenidis, L.: The Hitting Set Attack on Anonymity Protocols. In: Fridrich, J. (ed.) IH 2004. LNCS, vol. 3200, pp. 326–339. Springer, Heidelberg (2004)

[11] Mathewson, N., Dingledine, R.: Practical Traffic Analysis: Extending and Resisting Statistical Disclosure. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 17–34. Springer, Heidelberg (2005)

[12] Pham, V.: Analysis of the Anonymity Set of Chaumian Mixes. In: 13th Nordic Workshop on Secure IT-Systems (2008)

[13] Pham, D.V., Kesdogan, D.: A Combinatorial Approach for an Anonymity Metric. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 26–43. Springer, Heidelberg (2009)

[14] Serjantov, A., Danezis, G.: Towards an Information Theoretic Metric for Anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 259–263. Springer, Heidelberg (2003)

[15] Troncoso, C., Danezis, G.: The bayesian traffic analysis of mix networks. In: ACM Conference on Computer and Communications Security, CCS 2009, pp. 369–379 (2009)

[16] Troncoso, C., Gierlichs, B., Preneel, B., Verbauwhede, I.: Perfect matching disclosure attacks. In: Borisov, N., Goldberg, I. (eds.) PETS 2008. LNCS, vol. 5134, pp. 2–23. Springer, Heidelberg (2008)

A Analysis of Expectation Function for Number of Peer Choices

Relation to Number of Chosen Peers

Claim 1. *The expectation $E_D(x, x_1, x_2, j)$ is a monotonically decreasing function with respect to the number of chosen peers x, where $1 \leq x \leq m - \frac{1}{2}$.*

The proof consists of two parts. We will show that $E_D(x, x_1, x_2, j)$ is monotonically decreasing given that x_1 is fixed and then for the case that x_2 is fixed.

Proof (Monotonicity of $E_D(x, x_1, x_2, j)$ given fixed x_1). This analysis refers to the case that the number of chosen non-peers x_1 is fixed in the chosen peers x. By definition $x_2 = (x - x_1)$, therefore we replace all x_2 in (12) by $(x - x_1)$. The following function determines the gradient of the resulting function by computing its partial derivative with respect to x: $\frac{\partial E_D(x, x_1, x - x_1, j)}{\partial x} = \frac{tP_{nA}}{m}(2x - 2m - x_1 + j + 1)$.

This equation is less-than or equal 0, if:

$$x \leq m + 0.5(x_1 - j) - 0.5 \ . \tag{17}$$

We consider the inequality (17) for different cases of $(x_1 - j)$. By definition $x_1 \leq j$, therefore only the following cases exist:

$x_1 = j$: In this case E_D is a decreasing function if $x \leq m - \frac{1}{2}$.

$x_1 < j$: In this case E_D is always a decreasing function. The proof derives from the definition $x = (x_1 + x_2)$, where $x_2 \leq (m - j)$. Replacing x_2 in the first equation by the latter inequality, we obtain:

$$x \leq m + (x_1 - j) \quad \Rightarrow \quad x \leq m + 0.5(x_1 - j) - 0.5, \ \text{since } x_1 - j \leq -1 \ .$$

Therefore (17) is always fulfilled in this case.

This proves that $E_D(x, x_1, x_2, j)$ is a monotonically decreasing function with respect to the number of chosen peers x, where $1 \leq x \leq m - \frac{1}{2}$, given that x_1 is fixed. □

Proof (Monotonicity of $E_D(x, x_1, x_2, j)$ given fixed x_2). We now consider the case that the number of Alice's peers is fixed in the number of chosen peers x. The gradient of $E_D(x, x_1, x_2, j)$ with respect to x is now: $\frac{\partial E_D(x, x - x_2, x_2, j)}{\partial x} = \frac{tP_{nA}}{m}(-m + x_2 + j)$.

The relation $(x_2 + j) \leq m$ is given by definition, therefore the gradient is always less-than or equal to 0. This proves that $E_D(x, x_1, x_2, j)$ is a monotonically decreasing function, given that x_2 is fixed. □

We conclude from these two proofs that $E_D(x, x_1, x_2, j)$ is a monotonically decreasing function with respect to the number of chosen peers x, where $1 \leq x \leq m - \frac{1}{2}$. This completes the proof of Claim 1. All analyses in the rest of the paper implicitly assume $x \in [1, \ldots, m - 1]$.

Relation to Order of Peer Choice. This section will show that, in general, if one prefers to chose non-peers in $\mathcal{H} \in \mathfrak{H}_j$ first and then the remaining peers of Alice, then the number of choices required to disprove $\mathcal{H}$ is maximised.

Claim 2. *Let x be a fixed number of chosen peers and x_1 be the number of chosen non-peers, where $x_1 \leq j \leq x$. The expectation $E_D(x, x_1, x_2, j)$ with respect to x_1 is a monotonically increasing function.*

Proof. To analyse how E_D is related to the number of non-peer choices x_1, we compute the partial derivative of $E_D(x, x_1, x - x_1, j)$ with respect to $x_1 \leq j \leq x$, where x is fixed. This is: $\frac{\partial E_D(x,x_1,x-x_1,j)}{\partial x_1} = \frac{tP_{nA}}{m}(m - x - 1)$.

This equation is clearly greater than 0 (since $x \leq m - 1$ is assumed), therefore $E_D(x, x_1, x - x_1, j)$ is a monotonically increasing function for x_1 in the complete interval $[0, \dots, j]$. □

Note that $E_D(x, x_1, x - x_1, j)$ for $x_1 > j$ is, by definition of x_1, not defined. Given that $\mathcal{H}$ has $x \geq j$ chosen peers, $Po(\mathcal{H})$ is maximal if $x_1 = j$ of the chosen peers are non-peers. Disproving $\mathcal{H}$ therefore requires the maximal number of chosen peers if the non-peers are chosen first. To simplify the notation, and because of the importance of the number of non-peers, we will replace the notation $E_D(x, x_1, x - x_1, j)$ by the shorter notation $E_D(x, x_1, j)$ in the sequel.

A.1 Average Worst Case Number of Peer Choices

In this section we assume a worst case algorithm that chooses the peers of a set $\mathcal{H} \in \mathfrak{H}_j$ such that the number of peer choices x to disprove $\mathcal{H} \neq \mathcal{H}_A$ is maximal. According to the previous section this is the case if the non-peers are always chosen first in $\mathcal{H}$.

Claim 3. *Let* $\frac{N}{b-1} \geq 2(m-1)$. *The maximal number of peer choices x, such that* $E_D(x, x_1, j) = 0$ *with respect to N, b, m, j, is:*

$$x_w = m - 0.5 - \sqrt{jN(b-1)^{-1} - j^2 + j - mj + 0.25} \ . \tag{18}$$

We call x_w the *average worst case number of peer choices.*

Proof. In order to ensure that all non-peers are chosen first, we set $x_1 = j$. Given this, the maximal number of peer choices is the value x, such that $E_D(x, x_1, j)$ in (12) is 0.

$$\begin{aligned} 0 &= E_D(x, j, j) \\ &\leq \frac{t}{m}\Big[((m - x - 1)(m - x) + j^2)(1 - (1 - \frac{b-1}{N})) - j(1 - (b-1)\frac{m-1}{N})\Big] \ . \end{aligned}$$

We obtain (18) by computing the positive root of the last right hand side function for the variable x. Equation (18) is valid if the term within the square root is at least 0. That is, if:

$$0 \leq jN(b-1)^{-1} - j^2 + j - mj + 0.25 \ .$$

Since $j \leq m$ the above equation holds if: $N(b-1)^{-1} \geq 2(m-1)$.

Note that it is sufficient to assume $x_1 = j$ and $x \geq j$ for the proof. There is no need to consider the case $x < j$ for the average worst case number of peer choices, where $x_1 < j$ separately.

For an intuitive explanation, we assume a set $\mathcal{H} \in \mathfrak{H}_j$ for a maximal value j, such that $x_1 = j = x$ is the maximal number of non-peer choices to disprove $\mathcal{H}$. Let $\mathcal{H}' \in \mathfrak{H}_{j'}$ be another set, where $j' > j$. Since we assume that each Alice's peer are more frequently observed by the attacker than any non-peer, the relation $Po(\mathcal{H}') < Po(\mathcal{H})$ holds in most of the cases. We can particularly follow that $E_D(x, x_1, j') < E_D(x, x_1, j)$ implying that the maximal number of peer choices to disprove $\mathcal{H}$, as well as $\mathcal{H}'$ is x. Analysing the case $x_1 = j$ and $x \geq j$ is thus sufficient. A formal proof of this follows from a generalised form of (19), but is omitted here for brevity. □

Gradient of Worst Case Function $\boldsymbol{x_w}$. We now analyse the case where (18) is a monotonically decreasing function with respect to $\mathfrak{H}_j$ to simplify succeeding analyses. The next equation is the partial derivative of (18) with respect to j.

$$\frac{\partial x_w}{\partial j} = -\frac{1}{2} \frac{\left(N(b-1)^{-1} - 2j + 1 - m\right)}{(jN(b-1)^{-1} - j^2 + j - mj + 0.25)^{\frac{1}{2}}} \tag{19}$$

x_w is thus monotonically decreasing if the numerator in the above is at least 0.

$$0 \leq N(b-1)^{-1} - 2j + 1 - m \qquad \Rightarrow \qquad j \leq 0.5\left(N(b-1)^{-1} - m + 1\right)$$

Thus, if the maximal number of non-peer choices in a disproof is not larger than $\frac{1}{2}(\frac{N}{b-1} - m + 1)$, (18) is a monotonically decreasing function. If $\frac{N}{b-1} \geq 3m - 1$, then this case is necessarily fulfilled and we assume this condition for the remaining analyses.

Upper Bound of Average Number of Peer Choices. This section determines the upper bound of the average worst case number of peer choices x_w.

Claim 4. *Let $\frac{N}{b-1} \geq 3m-1$ and x_w be the average worst case number of peer choices. The maximal value of x_w for fixed N, b, m is:*

$$x_{uw} = m - \frac{1}{2} - \sqrt{\frac{N}{b-1} - m + \frac{1}{4}} \quad , \textit{ where } 0 \leq x_{uw} \leq m \quad . \tag{13}$$

We call x_{uw} the *upper bound of the average worst case number of peer choices.*

Proof. Let $\frac{N}{b-1} \geq 3m - 1$, then x_w is monotonic decreasing with respect to j. It is therefore maximal if we set $j = 1$ in (18) and thus obtain (13). □

In case of $\frac{N}{b-1} < 3m - 1$, the right hand side of equation (13) might not provide a maximal value for x_{uw}. Therefore we can conclude in this case that if $\frac{N}{b-1} = m - \frac{1}{4}$, then $x_{uw} \geq m - \frac{1}{2}$ and that x_{uw} increases if the value of $\frac{N}{b-1}$ decreases. This justifies the conclusions of Sect. 5.1.

From this analysis we can obtain an approximation of the ”lower bound of the average case complexity” of ExactHS. The derivation of these are omitted due to space limitation.

A Light-Weight Solution to Preservation of Access Pattern Privacy in Un-trusted Clouds

Ka Yang, Jinsheng Zhang, Wensheng Zhang, and Daji Qiao

Iowa State University, Ames, Iowa 50010, USA
{yangka,alexzjs,wzhang,daji}@iastate.edu

Abstract. Cloud computing is a new computing paradigm that is gaining increased popularity. More and more sensitive user data are stored in the cloud. The privacy of users' access pattern to the data should be protected to prevent un-trusted cloud servers from inferring users' private information or launching stealthy attacks. Meanwhile, the privacy protection schemes should be efficient as cloud users often use thin client devices to access the cloud. In this paper, we propose a lightweight scheme to protect the privacy of data access pattern. Comparing with existing state-of-the-art solutions, our scheme incurs less communication and computational overhead, requires significantly less storage space at the cloud user, while consuming similar storage space at the cloud server. Rigorous proofs and extensive evaluations have been conducted to demonstrate that the proposed scheme can hide the data access pattern effectively in the long run after a reasonable number of accesses have been made.

1 Introduction

Cloud computing [1, 12] enables enterprise and individual users to enjoy flexible, on-demand and high-quality services such as huge-volume data storage and processing, without the need to invest on expensive infrastructure, platform or maintenance. As more and more sensitive user data (e.g., financial records, health information, etc.) have been centralized into the cloud, cloud computing is facing great privacy and security challenges that may impede its fast growth and increased adoption if not well addressed. Rising to the challenges, researchers have proposed many schemes [7, 18, 23] to protect confidentiality and integrity of cloud data. Unfortunately, limited research has been conducted on the protection of users' privacy during their access to the cloud, such as the access frequency to each data item and the linkage between accesses of data items. Leakage of such access pattern information may enable potential privacy attacks such as focused attacks against selected data items. Cloud server may also infer a cloud user's activity pattern or private interest by tracking the user's access to a particular data item.

To strictly protect the privacy of data access pattern, the intention of every data access operation should be hidden so that observers of the operations cannot gain any meaningful information. Conforming to this strict requirement of access pattern privacy, Chor *et al.* [4], Ostrovsky *et al.* [14] and Itkis [10] introduced the notions of the private information retrieval (PIR) in an information theoretical setting and the computational PIR by restricting the database to perform only polynomial-time computations. Fully implementing the PIR notion is, however, expensive. As shown by Sion *et al.* [15],

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 528–547, 2011.

deployment of any single-server PIR protocol is not necessarily more efficient than a simple transfer of the entire database. Another approach to the strict preservation of data access pattern privacy is based on the notion of oblivious RAM (ORAM) [9]. In a latest ORAM implementation [20], about $\log n$ data items of the database should be scrambled every time after a single data item has been requested, where n is the total number of data items in the database. Let τ denote the size of a data item in bits. This ORAM scheme incurs a communication and computational complexity of $O(\log n \cdot \log \log n \cdot \tau)$ and requires $O(\sqrt{n} \cdot \tau)$ temporary user storage. The cost of this scheme is still rather expensive especially when the data are accessed frequently.

Although strict protection of data access pattern privacy is attractive, less strict protection, such as protecting the privacy of long-term access pattern, is also very useful in practice. For example, a malicious cloud server may use the statistical data access pattern of a user to infer the user's private information or conduct stealthy attacks. Moreover, being lightweight is also highly desired by users in cloud computing, as many of them often access the cloud with thin client devices such as smartphones. *Based on these considerations, we propose a lightweight scheme to preserve the privacy of long-term data access pattern in this paper.* The outline of the proposed scheme is as follows. Every time when a data item is needed by a user, (i) the user retrieves the desired data item together with additional dummy data items to hide the actual retrieval target; and (ii) the retrieved data items are re-encrypted and re-positioned before being stored back to the server to perturb the connections between data items and their storage locations at the server. The scheme records the storage locations of data items in index files, which are stored in a pyramid-like hierarchical structure at the cloud server to reduce communication, computational and storage overheads. Similar to data items, the access pattern to index files is also protected with additional dummies and re-positioning of the files after access. A set of delicately designed rules are used in the selection of dummy data items and index files as well as the repositioning of the files, which ensures that the connections between data items and their storage locations are reshuffled gradually, become more and more difficult to trace as the number of accesses increases, and eventually become fully un-trackable. Rigorous proofs and extensive evaluations have been conducted to demonstrate that the proposed scheme can hide the data access pattern in the long run, and the number of accesses required to preserve the access pattern privacy is reasonable in many situations.

The rest of the paper is organized as follows. Section 2 describes the system models. The proposed scheme is elaborated in Section 3, and Section 4 analyzes its security and overhead performances. Section 5 reports the evaluation results and Section 6 discusses the related work. Finally, Section 7 concludes the paper.

2 Models and Assumptions

2.1 System Model

We consider a basic cloud system with *a cloud server* and *a single cloud user*. The cloud user stores its sensitive data on the cloud server, which in turn provides an online interface for the cloud user to access the outsourced data. Later on, when the need for a data item arises, the cloud user requests it from the cloud server, updates the data item after usage, and then uploads the updated data item back to the server. Similar to [20,9], we assume that all the data items stored at the cloud server have the same size so the server cannot identify a data item from its size. In practice, this can be achieved conveniently by appending padding bits to short data items or dividing large data items into smaller ones.

2.2 Security Model

We assume that a cloud server is curious about the user's private information and may launch malicious attacks. Specifically, it may be interested in obtaining the user's data access pattern over the long term, which primarily includes the following information: *which data items that have been requested by the user and the number of times that a particular data item has been requested by the user.*

If the access pattern information is obtained, the cloud server may be able to launch various attacks. For example, the cloud server may attempt to infer the user's activity pattern or private interest via tracking the user's access to some particular data items. The cloud server may also launch focused attacks towards user's data that are accessed with very high frequency, or stealthily delete data that are never accessed to save its storage and maintenance costs without being noticed by the user.

As for the cloud user, we assume that it has a primitive encryption function that generates different cipher-texts over different input, and the cloud server does not have non-negligible advantage over the cloud user at determining whether a pair of encrypted items of the same length represent the same data item. We assume that data confidentiality and integrity are protected using existing techniques and the communication channel between the cloud user and the cloud server is secured using mechanisms such as SSL/IPSec. We do not consider denial of service attacks or timing attacks as they can be addressed independently from this work.

2.3 Design Goal

Our main design goal is to develop a lightweight solution to prevent the cloud server from knowing the cloud user's long-term access pattern to the data stored at the cloud server, while allowing the user to access the outsourced data with low communication and computational overhead. Specifically, we preserve the access pattern privacy by breaking the connections between the data items and their storage locations gradually.

3 The Proposed Scheme

3.1 System Setup

Before describing our proposed scheme in detail, we first explain the system setup.

Hierarchical Storage Structure at the Cloud Server. We study a system where a cloud user stores n distinct data items (denoted by d_i, $i = 1, \cdots, n$) at a cloud server. All data items are encrypted with the user's secret key before uploading. In addition to data items, the cloud server stores a hierarchy of index files with the following features:

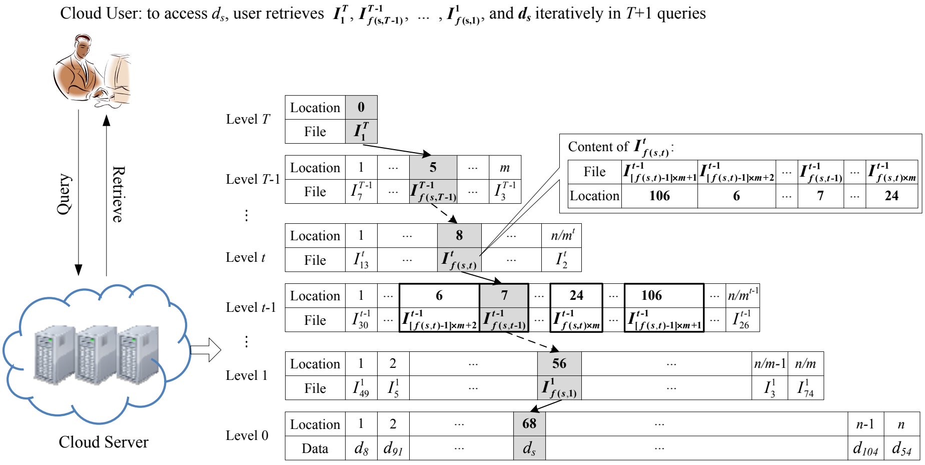

Cloud User: to access d_s, user retrieves I_1^T, $I_{f(s,T-1)}^{T-1}$, ... , $I_{f(s,1)}^1$, and d_s iteratively in T+1 queries
Query
Retrieve
Cloud Server
Level T
Location | 0
File | I_1^T
Level T-1
Location | 1 | ... | 5 | ... | m
File | I_7^{T-1} | ... | $I_{f(s,T-1)}^{T-1}$ | ... | I_3^{T-1}
Level t
Location | 1 | ... | 8 | ... | n/m^t
File | I_{13}^t | ... | $I_{f(s,t)}^t$ | ... | I_2^t
Content of $I_{f(s,t)}^t$:
File | $I_{[f(s,t)-1]\times m+1}^{t-1}$ | $I_{[f(s,t)-1]\times m+2}^{t-1}$ | ... | $I_{f(s,t-1)}^{t-1}$ | ... | $I_{f(s,t)\times m}^{t-1}$
Location | 106 | 6 | ... | 7 | ... | 24
Level t-1
Location | 1 | ... | 6 | 7 | ... | 24 | ... | 106 | ... | n/m^{t-1}
File | I_{30}^{t-1} | ... | $I_{[f(s,t)-1]\times m+2}^{t-1}$ | $I_{f(s,t-1)}^{t-1}$ | ... | $I_{f(s,t)\times m}^{t-1}$ | ... | $I_{[f(s,t)-1]\times m+1}^{t-1}$ | ... | I_{26}^{t-1}
Level 1
Location | 1 | 2 | ... | 56 | ... | n/m-1 | n/m
File | I_{49}^1 | I_5^1 | ... | $I_{f(s,1)}^1$ | ... | I_3^1 | I_{74}^1
Level 0
Location | 1 | 2 | ... | 68 | ... | n-1 | n
Data | d_8 | d_{91} | ... | d_s | ... | d_{104} | d_{54}

Fig. 1. System setup. Data items and index files form a pyramid-like hierarchical storage structure at the cloud server. Each index file records the storage locations of m index files at its next lower level. For example, the content of $I_{f(s,t)}^t$ is shown in the callout box, and the m level-$(t-1)$ index files associated with $I_{f(s,t)}^t$ are shown as bold boxes in the figure. Here, $f(s,t) = \lceil \frac{s}{m^t} \rceil$. To obatin data item d_s, the cloud user performs a sequence of queries iteratively in a top-down manner, to obtain T index files (marked as gray boxes), one at each level of the hierarchy.

- As shown in Fig. 1, there is a total of $T = \lceil \log_m n \rceil \geqslant 1$ levels of index files, where $m > 1$ is a design parameter. In Section 4.2, we analyze the relation between m and the communication, computational and storage overheads incurred by our solution. To simplify the presentation, we assume that $\log_m n$ is an integer in the rest of the paper.
- At level t ($t = 1, \cdots, T$), there are $\frac{n}{m^t}$ index files (denoted by I_j^t, $j = 1, \cdots, \frac{n}{m^t}$). So the total number of index files in the hierarchy is $\sum_{t=1}^{T} \frac{n}{m^t} = \frac{n-1}{m-1}$.
- Each index file records the storage locations of m index files at its next lower level. Specifically, I_j^t at level t contains the storage location information of the following index files at level $(t-1)$: $I_{(j-1)m+1}^{t-1}, I_{(j-1)m+2}^{t-1}, \cdots, I_{jm}^{t-1}$, as illustrated in the callout box in Fig. 1.
- There is only a single index file at the top level (i.e., level T): I_1^T.
- Data items form the bottom level (i.e., level 0) of the hierarchy.
- We assume that the files at different levels of the hierarchy are stored at non-overlapping storage spaces.

Note that, as shown in Fig. 1, there is no fixed order-correspondence between an index file (or a data item) and its storage location. This is due to the design nature of our proposed scheme, whose key idea is to randomize the storage locations of index files and data items after each access. Details of the scheme will be discussed in Section 3.2.

Iterative Query Process by the Cloud User. With such a pyramid-like hierarchical storage structure, we have the following observation about the relation between a data item and its index files: *the storage location of the data item d_s is recorded in the level-1 index file $I^1_{f(s,1)}$, whose storage location information is in turn recorded in the level-2 index file $I^2_{f(s,2)}$, so on and so forth, till the top-level index file I^T_1; here, $f(s,t)$ is defined as $f(s,t) = \lceil \frac{s}{m^t} \rceil$*. This relation is illustrated in Fig. 1 as a linked chain of gray boxes from top level T to bottom level 0.

Based on the above observation, we know that the user can obtain the desired data item d_s by performing a sequence of queries to obtain these T index files in the chain: I^T_1, $I^{T-1}_{f(s,T-1)}$, $\cdots$, $I^1_{f(s,1)}$, in a top-down manner through the hierarchy; once $I^1_{f(s,1)}$ is obtained, the user gets to know the storage location of d_s and can then issue the final query to obtain the data item. After the access, the data items and index files are updated, re-encrypted and uploaded back to the server.

We assume that the user requests the data items in rounds. To simplify the presentation, we assume that the user requests a single data item in each round. The proposed scheme may be extended to support requests of multiple data items in each round without much difficulty. In the following section, we explain our proposed scheme in detail. Table 1 lists the notations to be used in the rest of the paper.

Table 1. Notations Used in the Paper

Notation	Description
n	the total number of data items
$\mathcal{D}$	the set of all data item IDs
m	the number of storage locations recorded in an index file
I^t_j	the j-th index file at level t of the hierarchy
$\xi(j,t)$	the set of IDs of files whose storage locations are recorded in the level-t index file of ID j
$\mathcal{L}^t$	the set of storage locations of level-t files
$f(i,t)$	the ID of the index file that corresponds to data item d_i at level t
$\mathcal{Q}^t_{\text{pre}}(t \geqslant 1)$	the set of IDs and locations of level-t index files accessed in the previous round
$\mathcal{Q}^t_{\text{cur}}(t \geqslant 1)$	the set of IDs and locations of level-t index files to be accessed in the current round
$\mathcal{Q}^0_{\text{pre}}$	the set of IDs and locations of data items accessed in the previous round
$\mathcal{Q}^0_{\text{cur}}$	the set of IDs and locations of data items to be accessed in the current round

3.2 Scheme Description

Scheme Overview. Our proposed scheme is executed every time when the cloud user needs to request a data item. The key ideas of the scheme include: (i) extra dummy data items and index files (called *dummies* for short) are requested to hide the actual files of the user's interest; (ii) multiple dummies are selected so that the user's request at each round has the same format, which is a necessity to hide the access pattern [9] and (iii) the retrieved files are re-encrypted and re-positioned before being stored back to the server so as to break the connections between files and their storage locations at the server. Generally, these rules ensure that the connections between files and their storage locations are reshuffled gradually, become more and more difficult to trace as the number of accesses increases, and eventually become fully un-trackable. Detailed explanations and analysis will be presented in the following sections.

- *Assumption:* The following assumption is made on the initial condition when our scheme starts: *for any $t = 1, \cdots, T-1$, the mappings between level-t and level-$(t-1)$ files are unknown to the cloud server.* In other words, for any particular data item, the server has no knowledge about the corresponding index files; similarly, for any particular index file, the server has no knowledge about the corresponding index files at the upper layers.
- *Data Structures Recording Access History:* Our scheme makes use of past file access history when selecting dummies. To facilitate such mechanism, the historical information about the previous round of file access at layer t is recorded in a data structure denoted as $\mathcal{Q}^t_{\text{pre}}$, which consists of six fields: D_R, D_S and D_S recording the file IDs, and L_R, L_S and L_S recording their storage locations, respectively. The data structures are stored in cipher-text in a designated storage space at the server, and we denote the storage location of $\mathcal{Q}^t_{\text{pre}}$ as *Hist*$[t]$.
- *Structure of the Algorithm:* The pseudo-code of our scheme is presented in Algorithm 1 in Appendix 1. The scheme starts by selecting dummy data items. Then, it works iteratively to select, download, process and upload the index files, from the top level to the bottom level of the index hierarchy. In each iteration, it performs similar operations including *Selection & Downloading*, *Random Reshuffling*, and *Re-encryption & Uploading* of index files. Finally, the desired data item and the selected dummy data items are downloaded, randomly reshuffled, re-encrypted and uploaded. Detailed explanations of the operations are presented next, with a simple example given in Fig. 2.

Selection of Dummy Data Items. When the cloud user intents to retrieve a data item (denote its ID by $\mathcal{Q}^0_{\text{cur}}.D_R$), it also requests the following dummy data items to conceal its intention:

- the first dummy (whose ID is denoted as $\mathcal{Q}^0_{\text{cur}}.D_S$): the dummy that may swap its storage location with $\mathcal{Q}^0_{\text{cur}}.D_R$ after access with a probability of $1/2$;
- the second dummy (whose ID is denoted as $\mathcal{Q}^0_{\text{cur}}.D_N$): the dummy that will not swap its storage location with others.

$\mathcal{Q}^0_{\text{cur}}.D_S$ and $\mathcal{Q}^0_{\text{cur}}.D_N$ are selected to make sure that the user's request at each round has the same format: *the user always requests three data locations, out of which two and only two of them are from the ones accessed in the previous round.* Note that requiring user's request at each round to have the same format is necessary to hide the true access pattern [9]. Specifically, it hides the information about whether user's requests at two rounds are intended for the same data item. Also note that the second dummy is needed in order to guarantee that each access can keep the same format. Detailed explanations are presented in Appendix 2. To maintain the same format in each access, the data structure $\mathcal{Q}^0_{\text{pre}}$ is downloaded from the server, which records the information about the data items (namely, the data IDs and their corresponding locations) accessed in the previous round. Then, the dummies for the current round are selected according to the following rules:

- For the first dummy (i.e., $\mathcal{Q}^0_{\text{cur}}.D_S$): (i) If the intended data item is the same as the intended data item or the first dummy in the previous round, then the first dummy

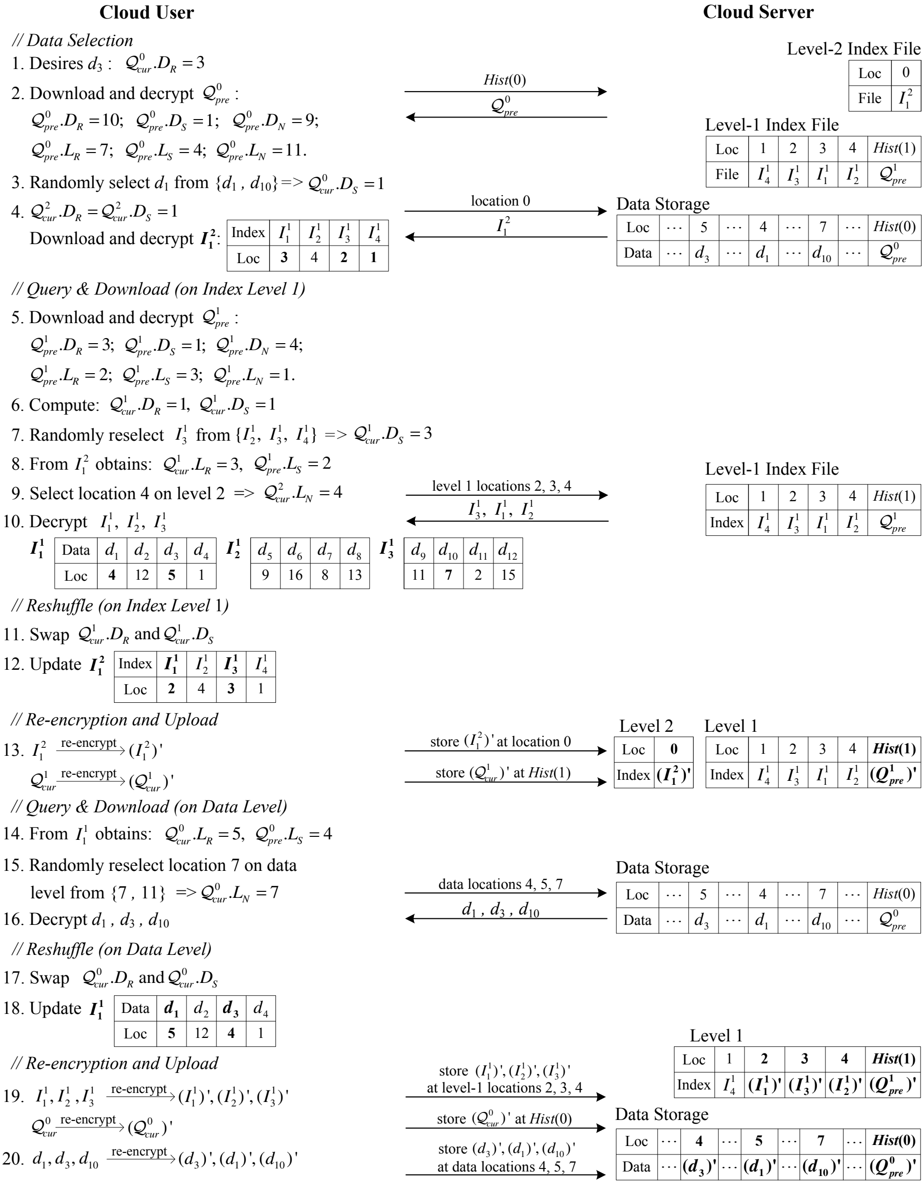

Cloud User
Cloud Server
// Data Selection
1. Desires d_3 : $Q^0_{cur}.D_R = 3$
2. Download and decrypt Q^0_{pre} :
$Q^0_{pre}.D_R = 10$; $Q^0_{pre}.D_S = 1$; $Q^0_{pre}.D_N = 9$;
$Q^0_{pre}.L_R = 7$; $Q^0_{pre}.L_S = 4$; $Q^0_{pre}.L_N = 11$.
3. Randomly select d_1 from $\{d_1, d_{10}\}$ => $Q^0_{cur}.D_S = 1$
4. $Q^2_{cur}.D_R = Q^2_{cur}.D_S = 1$
Download and decrypt I^2_1:
Index I^1_1 I^1_2 I^1_3 I^1_4
Loc 3 4 2 1
// Query & Download (on Index Level 1)
5. Download and decrypt Q^1_{pre} :
$Q^1_{pre}.D_R = 3$; $Q^1_{pre}.D_S = 1$; $Q^1_{pre}.D_N = 4$;
$Q^1_{pre}.L_R = 2$; $Q^1_{pre}.L_S = 3$; $Q^1_{pre}.L_N = 1$.
6. Compute: $Q^1_{cur}.D_R = 1$, $Q^1_{cur}.D_S = 1$
7. Randomly reselect I^1_3 from $\{I^1_2, I^1_3, I^1_4\}$ => $Q^1_{cur}.D_S = 3$
8. From I^2_1 obtains: $Q^1_{cur}.L_R = 3$, $Q^1_{pre}.L_S = 2$
9. Select location 4 on level 2 => $Q^2_{cur}.L_N = 4$
10. Decrypt I^1_1, I^1_2, I^1_3
I^1_1 Data d_1 d_2 d_3 d_4 / Loc 4 12 5 1
I^1_2 d_5 d_6 d_7 d_8 / 9 16 8 13
I^1_3 d_9 d_{10} d_{11} d_{12} / 11 7 2 15
// Reshuffle (on Index Level 1)
11. Swap $Q^1_{cur}.D_R$ and $Q^1_{cur}.D_S$
12. Update I^2_1 Index I^1_1 I^1_2 I^1_3 I^1_4 / Loc 2 4 3 1
// Re-encryption and Upload
13. $I^2_1 \xrightarrow{\text{re-encrypt}} (I^2_1)'$
$Q^1_{cur} \xrightarrow{\text{re-encrypt}} (Q^1_{cur})'$
// Query & Download (on Data Level)
14. From I^1_1 obtains: $Q^0_{cur}.L_R = 5$, $Q^0_{pre}.L_S = 4$
15. Randomly reselect location 7 on data level from $\{7, 11\}$ => $Q^0_{cur}.L_N = 7$
16. Decrypt d_1, d_3, d_{10}
// Reshuffle (on Data Level)
17. Swap $Q^0_{cur}.D_R$ and $Q^0_{cur}.D_S$
18. Update I^1_1 Data d_1 d_2 d_3 d_4 / Loc 5 12 4 1
// Re-encryption and Upload
19. $I^1_1, I^1_2, I^1_3 \xrightarrow{\text{re-encrypt}} (I^1_1)', (I^1_2)', (I^1_3)'$
$Q^0_{cur} \xrightarrow{\text{re-encrypt}} (Q^0_{cur})'$
20. $d_1, d_3, d_{10} \xrightarrow{\text{re-encrypt}} (d_3)', (d_1)', (d_{10})'$
$Hist(0)$
Q^0_{pre}
location 0
I^2_1
level 1 locations 2, 3, 4
I^1_3, I^1_1, I^1_2
store $(I^2_1)'$ at location 0
store $(Q^1_{cur})'$ at $Hist(1)$
data locations 4, 5, 7
d_1, d_3, d_{10}
store $(I^1_1)', (I^1_2)', (I^1_3)'$ at level-1 locations 2, 3, 4
store $(Q^0_{cur})'$ at $Hist(0)$
store $(d_3)', (d_1)', (d_{10})'$ at data locations 4, 5, 7
Level-2 Index File
Loc 0 / File I^2_1
Level-1 Index File
Loc 1 2 3 4 $Hist(1)$ / File I^1_4 I^1_3 I^1_1 I^1_2 Q^1_{pre}
Data Storage
Loc ··· 5 ··· 4 ··· 7 ··· $Hist(0)$ / Data ··· d_3 ··· d_1 ··· d_{10} ··· Q^0_{pre}
Level-1 Index File
Loc 1 2 3 4 $Hist(1)$ / Index I^1_4 I^1_3 I^1_1 I^1_2 Q^1_{pre}
Level 2
Loc 0 / Index $(I^2_1)'$
Level 1
Loc 1 2 3 4 $Hist(1)$ / Index I^1_4 I^1_3 I^1_1 I^1_2 $(Q^1_{pre})'$
Data Storage
Loc ··· 5 ··· 4 ··· 7 ··· $Hist(0)$ / Data ··· d_3 ··· d_1 ··· d_{10} ··· Q^0_{pre}
Level 1
Loc 1 2 3 4 $Hist(1)$ / Index I^1_4 $(I^1_1)'$ $(I^1_3)'$ $(I^1_2)'$ $(Q^1_{pre})'$
Data Storage
Loc ··· 4 ··· 5 ··· 7 ··· $Hist(0)$ / Data ··· $(d_3)'$ ··· $(d_1)'$ ··· $(d_{10})'$ ··· $(Q^0_{pre})'$

Fig. 2. An example of the access procedure of a cloud user. There is a total of $n = 16$ data items and $T = 2$ levels of index files stored at the cloud server. We use d'_i to represent that data item d_i appears differently after re-encryption. In this example, data items d_1, d_9, d_{10} were accessed in the previous round. It shows how the user operates when it is interested in obtaining data item d_3 in the current round.

will be selected uniformly at random from the set of all data items excluding the intended data item of the current round. (ii) Otherwise, the first dummy will be randomly selected from the intended data item or the first dummy in the previous round with equal probability. (Refer to lines 3 to 7 in Step 1 of Algorithm 1.)

- For the second dummy (i.e., $\mathcal{Q}^0_{\text{cur}}.D_N$), its selection depends on the selection results of the first dummy: (i) If both the intended data item and the first dummy have appeared in the previous round, the second dummy will be selected uniformly at random from the set of all data storage locations excluding the locations accessed in the previous round. (ii) Otherwise, the second dummy will be selected uniformly at random from the locations accessed in the previous round excluding locations of the already-selected files. (Refer to lines 12 to 20 in Step 2 of Algorithm 1 when $t = 0$.)

In the example given in Fig. 2, in the previous round, data $\#10$ was intended by the user and data $\#1$ was selected as the first dummy. Since data $\#3$ is needed in the current round (i.e., case (ii) in the first dummy selection rules), the user randomly selects the first dummy, which is data $\#1$ in this example, from data $\#10$ and data $\#1$ (as shown by step 3). As the selected data items did not both appear in the previous round (i.e., case (ii) in the second dummy selection rules), the second dummy's location, which is 7 in this example (as shown by step 15), is selected from data $\#10$ and data $\#9$'s locations (i.e., data locations $\#7$ and $\#11$).

Selection, Downloading, Processing and Uploading of Index Files. First, the single top-level index file is downloaded and decrypted, and its ID is recorded in $\mathcal{Q}^T_{\text{cur}}.D_R$, $\mathcal{Q}^T_{\text{cur}}.D_S$, and $\mathcal{Q}^T_{\text{cur}}.D_N$, i.e., $\mathcal{Q}^T_{\text{cur}}.D_R = \mathcal{Q}^T_{\text{cur}}.D_S = \mathcal{Q}^T_{\text{cur}}.D_N = 1$ (as shown by step 4 in the example of Fig. 2). Then, three index files for each level t, where $(T-1) \geqslant t \geqslant 1$, are selected, downloaded, processed and uploaded, in an iterative and top-down manner. Without loss of generality, the following describes the operations for iteration t.

Selection & Downloading of Level-t Index Files. The files that contain the level-t indices of the intended data item ($\mathcal{Q}^0_{\text{cur}}.D_R$) and the first dummy ($\mathcal{Q}^0_{\text{cur}}.D_S$) are first selected to access. The IDs of these files are denoted as $\mathcal{Q}^t_{\text{cur}}.D_R$ and $\mathcal{Q}^t_{\text{cur}}.D_S$ respectively. Note that, these file IDs can be found out by using the afore-defined $f(\cdot,\cdot)$ function, i.e., $\mathcal{Q}^t_{\text{cur}}.D_R = f(\mathcal{Q}^0_{\text{cur}}.D_R, t)$ and $\mathcal{Q}^t_{\text{cur}}.D_S = f(\mathcal{Q}^0_{\text{cur}}.D_S, t)$. Then, similar to the selection of dummy data items, additional dummy index files are selected to make sure that, in each round, three level-t index files are accessed and exactly two of them appeared in the previous round. The following rules are applied in the selection:

- For the first dummy index file (i.e., $\mathcal{Q}^t_{\text{cur}}.D_S$): If the intended data item and the first dummy share the same level-t index file, the first dummy index file is re-selected uniformly at random from the index files whose storage locations are stored in files $\mathcal{Q}^{t+1}_{\text{cur}}.D_R$ or $\mathcal{Q}^{t+1}_{\text{cur}}.D_S$, i.e., the level-$(t+1)$ intended index file and the first dummy index file downloaded in the previous iteration of this algorithm. (Refer to lines 8 to 10 in Step 2 of Algorithm 1.)
- For the second dummy index file (i.e., $\mathcal{Q}^t_{\text{cur}}.D_N$): (i) If the intended index file and the first dummy index file have both appeared in the previous round, the second

dummy index file will be selected uniformly at random from all level-t index file locations excluding the locations that appeared in the previous round. (ii) Otherwise, the second dummy index file will be selected uniformly at random from the locations that appeared in the previous round excluding locations of the already-selected files. (Refer to lines 12 to 20 in Step 2 of Algorithm 1 when $t \neq 0$.)

After the level-t index files have been selected, the locations of files $\mathcal{Q}^t_{\text{cur}}.D_R$ and $\mathcal{Q}^t_{\text{cur}}.D_S$ can be found by searching their indices in the downloaded level-$(t+1)$ index files, i.e., files $\mathcal{Q}^{t+1}_{\text{cur}}.D_R$ and $\mathcal{Q}^{t+1}_{\text{cur}}.D_S$. Then the locations of the three level-t index files are provided to the server and the files can be downloaded. Note that, the locations are presented to the server in an arbitrary order, so that the server cannot distinguish between desired index files and dummies. The downloaded files are then decrypted with the user's key.

In the example given in Fig. 2, since the intended data item and the first dummy share the same level-1 index file I^1_1, the user randomly selects a new first dummy index file, which is I^1_3 in this example, from level-1 index files $\{I^1_2, I^1_3, I^1_4\}$ (as shown by steps 6 and 7). Then the user looks up I^2_1 to find out the storage locations $\mathcal{Q}^1_{\text{cur}}.L_R$ and $\mathcal{Q}^1_{\text{cur}}.L_S$ (as shown by step 8). Since both I^1_1 and I^1_3 were accessed in the previous round, the user selects the second dummy index file with location $\#4$ (as shown by step 9). Hence, the user retrieves the files from level-1 storage locations $\#2$, $\#3$ and $\#4$.

Random Reshuffling of Selected Level-t Index Files. The intended index file ($\mathcal{Q}^t_{\text{cur}}.D_R$) and the first dummy index file ($\mathcal{Q}^t_{\text{cur}}.D_S$) may swap their storage locations with a probability of $1/2$. If the swap happens, the index information of these files should be updated in their index files $\mathcal{Q}^{t+1}_{\text{cur}}.D_R$ and $\mathcal{Q}^{t+1}_{\text{cur}}.D_S$, respectively. In the example given in Fig. 2, since files $\mathcal{Q}^1_{\text{cur}}.D_R$ and $\mathcal{Q}^1_{\text{cur}}.D_S$ are swapped, the user updates I^2_1 accordingly (as shown by steps 11 and 12).

Re-encryption & Uploading of Index Files. Now, we have completed the processing of level-$(t+1)$ index files $\mathcal{Q}^{t+1}_{\text{cur}}.D_R$, $\mathcal{Q}^{t+1}_{\text{cur}}.D_S$ and $\mathcal{Q}^{t+1}_{\text{cur}}.D_N$. To hide content and/or location changes made to them, these files should be re-encrypted before being uploaded back to the server. In our scheme, re-encryption is performed by applying the Cipher Block Chaining (CBC) encryption techniques [13] on the file content, where the first block of the file is a non-reappearing nonce. The user's key is used in the re-encryption. This way, the same secret key can be reused for encrypting all files, which simplifies the key management at the cloud user. Such re-encryption process ensures that a computationally bounded adversary does not have non-negligible advantage at determining whether a pair of encrypted data items (before and after re-encryption, respectively) carry the same data content.

After re-encryption, files $\mathcal{Q}^{t+1}_{\text{cur}}.D_R$, $\mathcal{Q}^{t+1}_{\text{cur}}.D_S$ and $\mathcal{Q}^{t+1}_{\text{cur}}.D_N$ are uploaded to their locations, respectively, but in an arbitrary order to make it difficult for the cloud server to track these files. At the end of iteration t, data structure $\mathcal{Q}^t_{\text{pre}}$ should be replaced by $\mathcal{Q}^t_{\text{cur}}$, then re-encrypted and uploaded to location *Hist*$[t]$. This way, next time when $\mathcal{Q}^t_{\text{pre}}$ is downloaded, it will reflect the mostly recent access history.

In the example given in Fig. 2, I^2_1 and $\mathcal{Q}^1_{\text{cur}}$ are re-encrypted and uploaded to the server at the storage locations $\#0$ and *Hist*$[1]$, respectively (as shown by step 13).

Downloading, Processing and Uploading of Data Items. After the above steps, the level-1 index files have been downloaded and decrypted. Based on the index information in these files, the desired data item and two additional dummy data items can be downloaded from the cloud server and decrypted with the user's key. Upon the user's access to the desired data item has been completed, the intended data item and the first dummy may swap their storage locations with a probability of $1/2$, and if the swap happens, changes will be made to the level-1 index files $\mathcal{Q}^1_{\text{cur}}.D_R$ and/or $\mathcal{Q}^1_{\text{cur}}.D_S$, respectively. Finally, the three level-1 index files and the three data items are re-encrypted and uploaded to the cloud server. Also, data structure $\mathcal{Q}^0_{\text{pre}}$ is updated to $\mathcal{Q}^0_{\text{cur}}$, re-encrypted and uploaded to the server. The re-encryption and uploading operations are performed in the similar manner as described above.

In the example given in Fig. 2, the user looks up I^1_1 to find the storage locations $\mathcal{Q}^0_{\text{cur}}.L_R = 5$ and $\mathcal{Q}^0_{\text{cur}}.L_S = 4$. As afore-explained, the user selects the second dummy's storage location $\mathcal{Q}^0_{\text{cur}}.L_R = 7$ (as shown by steps 14 and 15). Since data items $\mathcal{Q}^0_{\text{cur}}.D_R$ and $\mathcal{Q}^0_{\text{cur}}.D_S$ are swapped, the content of I^1_1 is updated (as shown by steps 17 and 18). Finally, the re-encrypted level-1 index files, $\mathcal{Q}^0_{\text{cur}}$ and data items are uploaded to the server respectively.

4 Security and Overhead Analysis

In this section, we first show that the proposed scheme can preserve the privacy of user data access pattern in the long run. That is, after a sufficiently large number of accesses, the frequency with which each data item has been accessed cannot be figured out by the cloud server. Then we discuss the practical implications of this security property through analyzing how our scheme can deal with some typical attacks that are based on the knowledge of data access pattern. Finally we analyze the overhead of the proposed scheme.

4.1 Security Analysis

We first show that the access pattern of index file locations, which can be observed by the cloud server, does not reveal extra information about the data access pattern. In the proposed scheme, index files are used to facilitate user query and data access. The content of an index file is protected by being re-encrypted after each access, based on the user's secret key and a random non-repeating nonce. Hence, it is impossible for the server to gain information about the data access pattern from the content of index files. The following theorem states that observing the access pattern of index file storage locations does not reveal more information about data access pattern than observing only the access pattern of data storage locations.

Theorem 1. The cloud server cannot gain any advantage in inferring user's data access pattern through observing the access pattern of index file storage locations.

Proof. Refer to [21].

As the observed access pattern of index file locations does not help in inferring data access pattern, we next study what can be inferred from observing only the access pattern of data storage locations. The following theorem formally states the property that, if the cloud server can only observe the access pattern of data storage locations, the data access pattern, namely, the data item requested by a cloud user and the frequency with which each data item has been accessed by a cloud user, can be preserved in the long run.

Theorem 2. If a cloud user has accessed the data items, despite the user access sequence, for a sufficiently large number of times, each storage location at the cloud server is accessed uniformly at random.

Proof. Refer to Appendix 3 for a sketch of the proof and [21] for the detailed proof.

Note that the proof of Theorem 2 also implies that, after a sufficiently large number of accesses, the server does not have non-negligible advantage at determining whether a specific data storage location corresponds to a particular data item.

Discussion. To further understand the practical implications of the above security property, we now discuss a few typical attacks that are based on the knowledge of data access pattern, and analyze how our scheme can deal with the attacks.

Security Against Tracking Data Items. Suppose the cloud server has identified a particular user data item via other means, e.g., physical spying. It may want to keep track of this data item thereafter. Using our proposed scheme, due to the property described in Theorem 2, after a sufficiently large number of accesses, the server does not have non-negligible advantage at determining which location the target data item is at. For example, after the first round that the target item has been accessed, from the server's perspective, the target item may be stored at any of the three accessed locations with an equal probability of $1/3$. Then if any of these three locations is accessed in the next round, the probability will be divided further among the newly accessed locations. Therefore, by solely observing the storage locations accessed by the user, the server could lose track of the target data item quickly.

Security Against Focused Attacks on Selected Data Items. Some of the cloud user's data items may be requested with very high frequency. These files are often important to the user. If a malicious cloud server knows which data items are frequently accessed, it may launch intensive attacks on the data, attempting to find out the content or contextual information of the data. Note this, such attacks are sometimes feasible in practice, for example, when the adopted data encryption algorithm or the key chosen by the user is not sophisticated enough, or some side information about the data can be obtained in other means. Using our proposed scheme, due to the property described in Theorem 2, all data storage locations will be equally accessed in the long run. Hence, the server cannot identify which data items are frequently requested by the user. Similarly, some of the cloud user's data items may be requested with very low frequency, e.g., backup data. A malicious cloud server may want to stealthily delete these rarely-accessed user data items to save storage and maintenance cost for itself without being noticed by the user. Such attack can also be stopped as our proposed scheme prevents the server from identifying rarely requested data items.

4.2 Overhead Analysis

Communication and Computational Overhead. With our proposed scheme, to access a single data item, the cloud user needs to obtain the following information from the cloud server:

- Three index files at each level of the storage hierarchy; each index file records the storage locations of m index files at its next lower level and it takes $\log n$ bits to represent a storage location.
- One access history file at each level of the storage hierarchy; each access history file records the IDs and storage locations of three index files (at this level) that were accessed in the previous round; hence, it contains six fields and each field is $\log n$-bit long.
- The desired data item and two additional dummy data items; let τ denote the size of each data item in bits.

Recall that there is a total of $\log_m n$ levels in our proposed hierarchical storage structure. Therefore, the overall communication and computational overhead for accessing a single data item can be calculated as:

$$\mathrm{OH}_{\text{c\&c}} = m \log n \cdot 3 \log_m n + 6 \log n \cdot \log_m n + 3\tau. \tag{1}$$

It is easy to verify that:

$$\begin{cases} \min \mathrm{OH}_{\text{c\&c}} = \mathrm{OH}_{\text{c\&c}}|_{m=4} = 9(\log n)^2 + 3\tau; \\ \max \mathrm{OH}_{\text{c\&c}} = \mathrm{OH}_{\text{c\&c}}|_{m=n} = (3n+6)\log n + 3\tau. \end{cases} \tag{2}$$

Storage Overhead. As explained in Section 3.1, the total number of index files in our proposed scheme is $\frac{n-1}{m-1}$. Each index file records the storage locations of m index files at its next lower level and it takes $\log n$ bits to represent a storage location. Therefore, the overall storage overhead at the cloud server can be calculated as:

$$\mathrm{OH}_{\text{s\_server}} = m \log n \cdot \frac{n-1}{m-1} + n\tau. \tag{3}$$

It is easy to verify that:

$$\begin{cases} \min \mathrm{OH}_{\text{s\_server}} = \mathrm{OH}_{\text{s\_server}}|_{m=n} = n \log n + n\tau; \\ \max \mathrm{OH}_{\text{s\_server}} = \mathrm{OH}_{\text{s\_server}}|_{m=2} = 2(n-1)\log n + n\tau. \end{cases} \tag{4}$$

At the user side, to operate our proposed scheme, the cloud user needs to store one access history file, three index files, and three more index files or data items at any given time. Therefore, the required storage at the user side is:

$$\mathrm{OH}_{\text{s\_user}} = 6 \log n + 3m \log n + \max\{3m \log n, 3\tau\}. \tag{5}$$

Overhead Comparison. Based on the above overhead analysis, we set $m = 4$ in our scheme. In Table 2, we compare our scheme with one of the state-of-the-art access pattern preservation schemes for single-cloud-server systems [20].

Table 2. Overhead Comparison

	Comm./Comp.	Storage (server side)	Storage (user side)
Our Scheme ($m = 4$)	$O((\log n)^2 + \tau)$	$O(n \max\{\log n, \tau\})$	$O(\max\{\log n, \tau\})$
Scheme in [20]	$O(\log n \cdot \log\log n \cdot \tau)$	$O(n \cdot \tau)$	$O(\sqrt{n} \cdot \tau)$

It is interesting to see that, as long as the size of a data item (τ, in bits) is larger than $\log n$ where n is the total number of data items, which usually holds true in practical cloud storage applications, our scheme is more efficient. Specifically, our scheme (i) consumes similar storage space at the cloud server; (ii) usually incurs significantly less communication and computational overhead; and (iii) requires significantly less storage space at the cloud user, which facilitates the employment of our proposed scheme on thin user devices such as mobile phones. Note that the better efficiency performance of our scheme is achieved under a less stringent privacy requirement than [20]; instead of requiring strict privacy protection to the data access pattern, our scheme aims to protect the privacy of the data access pattern in the long run.

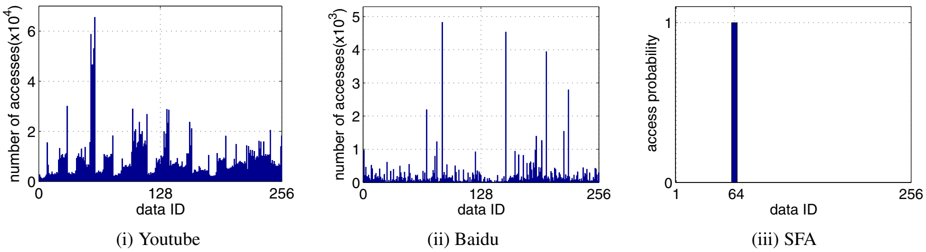
5 Performance Evaluation

5.1 Evaluation Setup

To evaluate the performance of the proposed scheme, we have collected two user access traces from two popular cloud service providers: *Youtube* [22] and *Baidu* [2]. As shown in Figs. 3(i) and (ii), both the Youtube user and the Baidu user have 256 files stored at the server. Different files have been accessed with different frequencies over time. Moreover, we have created an additional user who always requests the same file from the server, called the *SFA* (Single File Access) user, as shown in Fig. 3(iii). We use the SFA user to emulate an extreme access pattern. The total number of files stored at the server for the SFA user is also 256.

5.2 Preservation of Access Frequency Privacy

To study how well our proposed scheme preserves a cloud user's access frequency privacy, we propose to use *entropy* to measure the distribution of the user's access frequencies to different files. Specifically, let C_i denote the number of accesses to the file stored at storage location i. Then, the access frequency to location i is $F_i = \frac{C_i}{\sum_i C_i}$, and the entropy of access frequency is $H_F = -\sum_i F_i \log(F_i)$. For example, H_F of the Youtube and Baidu traces is around 7.6 and 6.5, respectively, which can be calculated by counting the number of accesses to each file in Figs. 3(i) and (ii). Clearly, for a given set of files stored at the server, the maximum entropy is achieved when all file locations

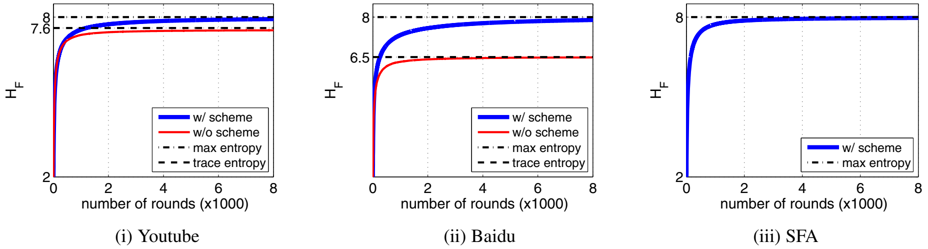


(i) Youtube (ii) Baidu (iii) SFA

Fig. 3. Data access traces and distribution used in the performance evaluation

have been accessed with an equal probability. This means that, the maximum entropy for accessing 256 files is $H_F^{\max}(256) = -256 \times \frac{1}{256}\log(\frac{1}{256}) = 8$.

We evaluate how the entropy of access frequency changes as the number of access rounds increases. Fig. 4 plots the results (averaged over 100 simulation runs) for different access scenarios. It can be seen clearly from the figures that, with our scheme, the entropy of access frequency improves over the original trace, and converges gradually to the maximum entropy in all simulated scenarios. This confirms our analytical study in Section 4 and Theorem 2 that the access frequency distribution converges towards the uniform distribution in the long run.



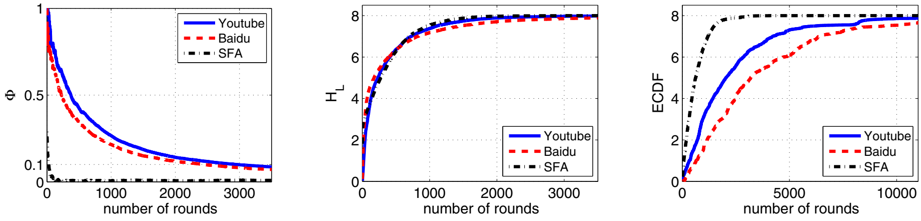
(i) Youtube (ii) Baidu (iii) SFA

Fig. 4. The entropy of access frequency vs. the number of access rounds for a particular simulation run under different access scenarios. In (iii), because the SFA user always requests the same data item at each round, the entropy of access frequency without using our proposed scheme is always zero, which is not shown in the figure.

5.3 Preservation of Access Order Privacy

In this section, we demonstrate the effectiveness of the proposed scheme in preserving the access order privacy. We do so by evaluating the correlation between the output access sequences (i.e., the sequence of the requested data items' storage locations) for the same input access sequence (i.e., the sequence of actual data items requested by the user). Specifically, in each simulation run, we simulate the access procedure using the same input access sequence twice and calculate the *correlation coefficient* (denoted as Φ) between the two output sequences. A smaller Φ indicates that the two output sequences are less correlated, and thus the access order privacy is better preserved. Note that, using our scheme, the server observes accesses to three storage locations at each round. Therefore, it won't be able to get the exact sequence of the requested data items' storage locations, which also helps to preserve the access order privacy.

Figs. 5(i) plot the Φ values (averaged over 100 simulation runs) as the number of access rounds increases for different access scenarios. We can see that Φ decreases as the number of access rounds increases, thus the correlation between the output sequences becomes looser. Notice that Φ never reaches zero (i.e., perfect access order privacy) in the simulation, which is due to the randomness and finite length of the output sequence. As a result, Φ remains at small values (e.g., < 0.1) after a number of accesses.



(i) (ii) (iii)

Fig. 5. (i) Average correlation coefficient (Φ) between output sequences for the same input sequences with our proposed scheme. (ii) Average entropy of location distribution vs. the number of access rounds for the most frequently requested data item. (iii) Average entropy of location distribution vs. the number of access rounds for the least frequently requested data item.

5.4 Preservation of Data Item's Location Privacy

As discussed in Section 4.1, when the user employs our proposed scheme, the cloud server loses track of a certain data item gradually over time. In other words, from the server's perspective, the uncertainty of a data item's storage storage location increases gradually over time. Similar to the evaluation of access frequency privacy, we also use *entropy* to measure the uncertainty of a particular data item's storage location from the server's perspective. It is defined as $H_L = -\sum_i p_i \log(p_i)$, where p_i is the probability that the data item is at storage location i from the server's perspective. We evaluate how the entropy of the data item's location distribution grows as the number of access rounds increases. For each access scenario, we collect the statistics of the most accessed data item and the least accessed data item, and results (averaged over 100 simulation runs) are plotted in Figs. 5(ii) and (iii), respectively. From the figures, we can see that a data item's location distribution entropy reaches the maximum regardless of their real access frequency. Note that, without our proposed scheme, a data item's location distribution entropy is zero because its location is fixed and known to the server.

6 Related Work

Although many schemes [19,18,23] have been proposed to protect data confidentiality and data integrity for the cloud computing paradigm, little effort has been made to protect users' access pattern privacy. Private Information Retrieval (PIR) [5, 15, 11], Oblivious RAM [9, 20] and Steganographic File Systems (SFS) [24, 16, 6]are the works most related to our solution.

Private Information Retrieval: PIR schemes aim to allow clients to retrieve information from a database while maintaining the privacy of the queries to the database. Fully implementing the PIR notion is, however, expensive. As shown by Sion *et al.* [15], deployment of any single-server PIR protocol is not necessarily more efficient than a simple transfer of the entire database due to computational costs. On the other hand, PIR schemes typically do not address data confidentiality, which makes PIR schemes unsuitable to be applied in the un-trusted cloud environments.

Oblivious RAM: In order to prevent the users' access pattern from being revealed, *Oblivious RAM (ORAM)* [20, 8] has been proposed. In a latest version of ORAM, Williams *et al.* [20] proposed to user encrypted Bloom Filter [3] to reshuffle and scramble data in the database. In Section 4.2, we have shown that our scheme is much more efficient in terms communication, computational and storage overheads in practical cloud storage applications under a less stringent privacy requirement.

Steganographic File Systems: Research efforts on steganographic file systems [24, 16, 6] are also related to our proposed design. The major differences lie in that, the research on SFS targets at protecting the information about existence and/or locations of sensitive files through hiding both short-term and long-term access patterns, while our proposal mainly targets at protecting long-term access pattern at low cost.

Recently, there is a concurrent effort [17] that addresses a similar problem as the one in our work. Their solution and ours share similar high-level ideas such as usage of dummies, hierarchical storage structure and file reshuffling. However, there are several key differences between the two solutions. For example, our solution yields provable security and overhead performances and does not require user-side LRU cache or an empirical statistical access model.

7 Conclusions and Future Work

In this paper, we present a lightweight solution to the preservation of a cloud users' data access pattern privacy in un-trusted clouds. Rigorous proofs have been provided to show that the proposed scheme can provide full protection to data access pattern privacy in the long run. Extensive evaluations have also been conducted to show that the scheme can protect the data access pattern privacy effectively after a reasonable number of accesses have been made. In the future work, we plan to enhance the scheme such that it can support private and efficient data updates, including data changes, data insertions and data deletions.

Acknowledgments. We would like to thank the shepherd and the reviewers for their valuable suggestions for strengthening the paper. The research reported in this paper was supported in part by the Information Infrastructure Institute (iCube) of Iowa State University, the Security and Software Engineering Research Center (S^2ERC), the National Science Foundation under Grants CNS 0831874 and CNS 0716744, and the office of Naval Research under Grant N000140910748.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB-EECS (2009)
2. Baidu, `http://passport.baidu.com/?business&aid=6&un=chenfoxlord#7`
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13 (1970)
4. Chor, B., Gilboa, N.: Computationally private information retrieval. In: Proc. STOC 1997 (1997)
5. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proc. FOCS 1998 (1998)
6. Diaz, C., Troncoso, C., Preneel, B.: A framework for the analysis of mix-based steganographic file systems. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 428–445. Springer, Heidelberg (2008)
7. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: Proc. SOSP 2003 (2003)
8. Goldreich, O.: Towards a theory of software protection and simulation by oblivious rams. In: Proc. STOC 1987 (1987)
9. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious ram. In: JACM 1996 (1996)
10. Itkis, G.: Personal communication, via oded goldreich (1996)
11. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: Proc. IEEE Symposium on Foundations of Computer Science (1997)
12. Mell, P., Grance, T.: Draft: Nist working definition of cloud computing (2010)
13. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
14. Ostrovsky, R., Shoup, V.: Private information storage. In: Proc. STOC 1997 (1997)
15. Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: Proc. NDSS 2007 (2007)
16. Troncoso, C., Diaz, C., Dunkelman, O., Preneel, B.: Traffic analysis attacks on a continuously-observable steganographic file system. In: Furon, T., Cayre, F., Doërr, G., Bas, P. (eds.) IH 2007. LNCS, vol. 4567, pp. 220–236. Springer, Heidelberg (2008)
17. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. ICDCS 2011 (2011)
18. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Proc. IWQoS 2009 (2009)
19. Wang, C., Wang, Q., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: Proc. ICDCS 2010 (2010)
20. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: Proc. CCS 2008 (2008)
21. Yang, K., Zhang, J., Zhang, W., Qiao, D.: A light-weight solution to preservation of access pattern privacy in un-trusted clouds. Technical Report (2011), `http://www.public.iastate.edu/~yangka/PatternFull.pdf`
22. Youtube, `http://www.youtube.com/user/supercwm`
23. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained access control in cloud computing. In: Proc. INFOCOM 2010 (2010)
24. Zhou, X., Pang, H., Tan, K.L.: Hiding data accesses in steganographic file system. In: Proc. ICDE 2004 (2004)

Appendix 1

What follows is the algorithm for the proposed access procedure of a cloud user. The details of the algorithm are described and explained in Section 3.2.

Algorithm 1. Proposed Access Procedure of a Cloud User

```
Step 1: Selection of Data Items (of IDs $\mathcal{Q}^0_{\mathbf{cur}}.D_R$ and $\mathcal{Q}^0_{\mathbf{cur}}.D_S$) to Access
$\mathcal{Q}^0_{\mathrm{cur}}.D_R \leftarrow UserRequest()$; $k \leftarrow UserKey()$;      // input user's desired data and secret key
$Download\&Decrypt_k(\mathcal{Q}^0_{\mathrm{pre}}, Hist[0])$;
                         // get access history of data items from location $Hist[0]$ & decrypt it
if $\mathcal{Q}^0_{\mathrm{cur}}.D_R \in \{\mathcal{Q}^0_{\mathrm{pre}}.D_R, \mathcal{Q}^0_{\mathrm{pre}}.D_S\}$ then
    $\mathcal{Q}^0_{\mathrm{cur}}.D_S \leftarrow RandomSelectOne(\mathcal{D} \setminus \{\mathcal{Q}^0_{\mathrm{cur}}.D_R\})$;
else
    $\mathcal{Q}^0_{\mathrm{cur}}.D_S \leftarrow RandomSelectOne(\{\mathcal{Q}^0_{\mathrm{pre}}.D_R, \mathcal{Q}^0_{\mathrm{pre}}.D_S\})$;
end if
Step 2: Query for Index Files and Data Items
$Download\&Decrypt_k(I^T_1, 0)$;      // download top-level index file from location $0$ & decrypt it
$\mathcal{Q}^T_{\mathrm{cur}}.D_R \leftarrow 1$; $\mathcal{Q}^T_{\mathrm{cur}}.D_S \leftarrow 1$; $\mathcal{Q}^T_{\mathrm{cur}}.D_N \leftarrow 1$;
for $(t \leftarrow (T-1); t \geqslant 0; t--)$ do
    // Step 2.1: Selection of Level-t Index Files and $\mathcal{Q}^0_{cur}.L_N$
    if $t > 0$ then
        $Download\&Decrypt_k(\mathcal{Q}^t_{\mathrm{pre}}, Hist[t])$;      // get access history of level-t index files
        $\mathcal{Q}^t_{\mathrm{cur}}.D_R \leftarrow f(\mathcal{Q}^0_{\mathrm{cur}}.D_R, t)$; $\mathcal{Q}^t_{\mathrm{cur}}.D_S \leftarrow f(\mathcal{Q}^0_{\mathrm{cur}}.D_S, t)$;
                   // find out files storing level-t indices of data items $\mathcal{Q}^0_{cur}.D_R$ and $\mathcal{Q}^0_{cur}.D_s$
        if $\mathcal{Q}^t_{\mathrm{cur}}.D_R = \mathcal{Q}^t_{\mathrm{cur}}.D_S$ then
            $\mathcal{Q}^t_{\mathrm{cur}}.D_S \leftarrow RandomSelectOne(\xi(\mathcal{Q}^{t+1}_{\mathrm{cur}}.D_R, t+1) \cup \xi(\mathcal{Q}^{t+1}_{\mathrm{cur}}.D_S, t+1) \setminus \{\mathcal{Q}^t_{\mathrm{cur}}.D_R\})$;
        end if
    end if
    if $\{\mathcal{Q}^t_{\mathrm{cur}}.D_R, \mathcal{Q}^t_{\mathrm{cur}}.D_S\} \subseteq \mathcal{Q}^t_{\mathrm{pre}}$ then
        $\mathcal{Q}^t_{\mathrm{cur}}.L_N \leftarrow RandomSelectOne(\mathcal{L}^t \setminus \{\mathcal{Q}^t_{\mathrm{pre}}.L_R, \mathcal{Q}^t_{\mathrm{pre}}.L_S, \mathcal{Q}^t_{\mathrm{pre}}.L_N\})$;
    else
        if $\mathcal{Q}^t_{\mathrm{cur}}.D_R \in \mathcal{Q}^t_{\mathrm{pre}}$ then
            $\mathcal{Q}^t_{\mathrm{cur}}.L_N \leftarrow RandomSelectOne(\{\mathcal{Q}^t_{\mathrm{pre}}.L_R, \mathcal{Q}^t_{\mathrm{pre}}.L_S, \mathcal{Q}^t_{\mathrm{pre}}.L_N\} \setminus \{\mathcal{Q}^t_{\mathrm{cur}}.L_R\})$;
        else
            $\mathcal{Q}^t_{\mathrm{cur}}.L_N \leftarrow RandomSelectOne(\{\mathcal{Q}^t_{\mathrm{pre}}.L_R, \mathcal{Q}^t_{\mathrm{pre}}.L_S, \mathcal{Q}^t_{\mathrm{pre}}.L_N\} \setminus \{\mathcal{Q}^t_{\mathrm{cur}}.L_S\})$;
        end if
    end if
    $Download\&Decrypt_k(\mathcal{Q}^t_{\mathrm{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^t_{\mathrm{cur}}.\{L_R, L_S, L_N\})$;
    /* download files of IDs $\mathcal{Q}^t_{cur}.\{D_R, D_S, D_N\}$ from locations specified by $\mathcal{Q}^t_{cur}.\{L_R, L_S, L_N\}$
                                         respectively but in an arbitrary order & decrypt them */
    // Step 2.2: Random Reshuffling
    if $RandomSelectOne(\{0, 1\}) = 1$ then
        $Swap(\mathcal{Q}^t_{\mathrm{cur}}.D_R, \mathcal{Q}^t_{\mathrm{cur}}.D_S)$;
        Update $\mathcal{Q}^t_{\mathrm{cur}}.\{L_R, L_S\}$ in level-$(t+1)$ index files of IDs $\mathcal{Q}^{t+1}_{\mathrm{cur}}.\{D_R, D_S\}$;
    end if
    // Step 2.3: Reencryption/Uploading of Level-$(t+1)$ Files and Level-t Access History
    $Reencrypt_k\&Upload(\mathcal{Q}^{t+1}_{\mathrm{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^{t+1}_{\mathrm{cur}}.\{L_R, L_S, L_N\})$;
    /* reencrypt & upload files of IDs $\mathcal{Q}^{t+1}_{cur}.\{D_R, D_S, D_N\}$ to locations specified by
                                   $\mathcal{Q}^{t+1}_{cur}.\{L_R, L_S, L_N\}$ respectively but in an arbitrary order */
    $Reencrypt_k\&Upload(\mathcal{Q}^t_{\mathrm{cur}}, Hist[t])$;
end for
Step 3: Reencryption and Uploading of Accessed Data Items
$Reencrypt_k\&Upload(\mathcal{Q}^0_{\mathrm{cur}}.\{D_R, D_S, D_N\}, \mathcal{Q}^0_{\mathrm{cur}}.\{L_R, L_S, L_N\})$;
```

Appendix 2

We now explain why our proposed scheme requires the user to download two dummy data items together with the intended data item in each access.

Suppose the scheme only downloads one dummy data item (whose ID is denoted as $\mathcal{Q}^0_{\text{cur}}.D_R$) together with the intended data item (whose ID is denoted as $\mathcal{Q}^0_{\text{cur}}.D_S$). We let the dummy data item be selected to make sure that the user's request at each round has the same format: *the user always requests two data locations, out of which one and only one of them is from the ones accessed in the previous round.* The rules for selecting the dummy data item are: (i) if the intended data item has been accessed in the previous round, the dummy is selected uniformly at random from the data items that have not been accessed in the previous round; (ii) otherwise, the dummy is selected from the two accessed data items with equal probability.

Similarly, we would like to have the same format at each round of index file access: *at each index level, the user always requests two index file locations, out of which one and only one of them is from the ones accessed in the previous round.* Unfortunately, this may not always be possible with a single dummy index file. An example is given in Fig. 6 to illustrate the problem.

Level-2 Index File

I_1^2

Index	I_1^1	I_2^1	I_3^1	I_4^1
Loc	3	4	2	1

Level-1 Index File

I_1^1

Data	d_1	d_2	d_3	d_4
Loc	4	12	5	1

I_2^1

d_5	d_6	d_7	d_8
9	16	8	13

...

Fig. 6. In this example, there are $n = 16$ data items and $T = 2$ levels of index files stored at the cloud server. The contents of index files I_1^2, I_1^1 and I_2^1 are shown in the figure.

In Fig. 6, suppose in the first round, the user needs data item d_1 and data item d_2 is randomly selected to be the dummy. Since d_1 and d_2 share the same level-1 index file I_1^1, the user needs to randomly select a new dummy index file. Suppose the user selects I_2^1 as the dummy index file. Then in the second round, suppose the user needs data item d_5. According to the selection rules, the user randomly selects a dummy from d_1 and d_2. However, no matter whether d_1 or d_2 is selected as the dummy, the user needs to retrieve I_1^1 and I_2^1 in order to get the storage locations of d_5 and the selected dummy. Note that both I_1^1 and I_2^1 have been accessed in the previous round; this violates the desired access format.

A quick remedy to the problem may be as following: when selecting the dummy, the user randomly selects the dummy from data items that do not share the same index files (except for the top level) with the intended data item. It is easy to see that this selection rule can avoid the afore-described problem. However, such remedial action may leak information about user's access pattern in some situations. For example, in Fig. 6, if the user accesses d_1 consecutively, data locations where d_2, d_3 and d_4 are stored will never be accessed, which may leak information about the data item of user's interest.

There may exist more sophisticated rules that can preserve the user's long-run access pattern using a single dummy, which we are not aware of at the moment. So instead, in this work, we adopt an efficient two-dummy solution to guarantee that user's access at each around has the same format.

Appendix 3

In this section, we sketch the proof for Theorem 2. Please refer to [21] for the detailed proof. In the proposed scheme, at each round of access, the user accesses three data items, where two of them (D_R and D_S) randomly swap their locations after the access and the other D_N does not. Therefore, the selection of D_N does not affect the location distribution of the data items. As a result, in the proof, we only need to consider the behavior of D_R and D_S.

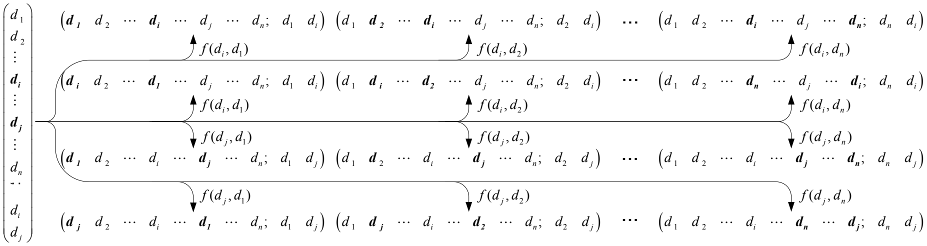

$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_i\ d_j)$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_1\ d_i)$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_2\ d_i)$
$\cdots$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_n\ d_i)$
$f(d_i,d_1)$
$f(d_i,d_2)$
$f(d_i,d_n)$
$(d_i\ d_2\ \cdots\ d_1\ \cdots\ d_j\ \cdots\ d_n;\ d_1\ d_i)$
$(d_1\ d_i\ \cdots\ d_2\ \cdots\ d_j\ \cdots\ d_n;\ d_2\ d_i)$
$\cdots$
$(d_1\ d_2\ \cdots\ d_n\ \cdots\ d_j\ \cdots\ d_i;\ d_n\ d_i)$
$f(d_i,d_1)$
$f(d_i,d_2)$
$f(d_i,d_n)$
$f(d_j,d_1)$
$f(d_j,d_2)$
$f(d_j,d_n)$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_1\ d_j)$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_2\ d_j)$
$\cdots$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_n\ d_j)$
$f(d_j,d_1)$
$f(d_j,d_2)$
$f(d_j,d_n)$
$(d_j\ d_2\ \cdots\ d_i\ \cdots\ d_1\ \cdots\ d_n;\ d_1\ d_j)$
$(d_1\ d_j\ \cdots\ d_i\ \cdots\ d_2\ \cdots\ d_n;\ d_2\ d_j)$
$\cdots$
$(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_n\ \cdots\ d_j;\ d_n\ d_j)$

Fig. 7. One-step transition from an arbitrary state $(d_1\ d_2\ \cdots\ d_i\ \cdots\ d_j\ \cdots\ d_n;\ d_i\ d_j)$ to other reachable states in MC-1. $f(\cdot,\cdot)$ is the transition probability function.

Let d_i $(i = 1, \cdots, n)$ denote the data items and P_i denote the probability with which d_i is actually requested by the user in each round of access. We model the data access process with a homogeneous Markov chain denoted as MC-1, as shown in Fig. 7. Each state of MC-1 is $(\sigma;\ d_i\ d_j)$. Here, σ is a permutation of $(d_1, \cdots, d_n)$, which stands for one distribution of the n data items to n storage locations. i and j are two distinct numbers from $\{1,\ \cdots,\ n\}$, and d_i and d_j is D_R and D_S respectively. Hence, there is a total of $n!\binom{n}{2}$ distinct states in MC-1.

In the proof, we show that MC-1 converges to a steady state. In the steady state, all permutations of data items are equally likely to happen. Consequently, every data item is uniformly randomly distributed to all storage locations in the steady state. This implies that each storage location will be accessed uniformly at random in the long run.

Optimizing Mixing in Pervasive Networks: A Graph-Theoretic Perspective

Murtuza Jadliwala, Igor Bilogrevic, and Jean-Pierre Hubaux

LCA1, EPFL, Lausanne, Switzerland
firstname.lastname@epfl.ch

Abstract. One major concern in pervasive wireless applications is location privacy, where malicious eavesdroppers, based on static device identifiers, can continuously track users. As a commonly adopted countermeasure to prevent such identifier-based tracking, devices regularly and simultaneously change their identifiers in special areas called mix-zones. Although mix-zones provide spatio-temporal de-correlations between old and new identifiers, pseudonym changes, depending on the position of the mix-zone, can incur a substantial cost on the network due to lost communications and additional resources such as energy. In this paper, we address this trade-off by studying the problem of determining an optimal set of mix-zones such that the degree of mixing in the network is maximized, whereas the overall network-wide mixing cost is minimized. We follow a graph-theoretic approach and model the optimal mixing problem as a novel generalization of the vertex cover problem, called the *Mix Cover (MC)* problem. We propose three bounded-ratio approximation algorithms for the MC problem and validate them by an empirical evaluation of their performance on real data. The combinatorics-based approach followed here enables us to study the feasibility of determining optimal mix-zones regularly and under dynamic network conditions.

1 Introduction

Recent advances in wireless and mobile computing technology have resulted in rapid proliferation and use of this technology for a variety of pervasive computing and data-sharing applications. Popular instances of this networking technology include vehicular and pervasive social networking systems and applications such as vehicular safety messaging [48,37], pervasive or local-area social networking [41,4,10], dating [1,2,33], personal safety [39] and micro-blogging [21]. Communication devices such as mobile phones (in personal networks) or wireless on-board computers (in vehicular networks) communicate with each other directly in a peer-to-peer fashion or with third-party service providers through an infrastructure such as a base station or road-side unit.

Users in such pervasive systems continuously face privacy risks, especially in terms of location privacy, from malicious eavesdroppers and curious service providers. Users' location information revealed as a result of this threat can be used by malicious parties to track their movements and preferences [20] or

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 548–567, 2011.

to identify users and their availabilities by inferring their home/work locations [25], which can be later used for accomplishing malicious goals [3]. Third-party service providers, however, are generally trusted and claim to utilize the collected personal and location information to further enhance context-aware services but can inadvertently harm users' privacy if the collected data is improperly shared with commercial partners or leaked in an unauthorized fashion.

One widely adopted strategy to overcome the location privacy concerns in such pervasive systems, which is inspired by Chaum's seminal work on mix networks [13,14], is to regularly mix [35] or change [29,12] device identifiers including application, IP and device MAC addresses. Recently, non-IP networks such as cellular networks adopt a similar approach; they identify a subscriber's device with a Temporary Mobile Subscriber Identity or TMSI that changes every time the subscriber moves to a new geographical area.

In order to maximize anonymity, mixing or changing of user identifiers should occur in a spatiotemporal region, called *mix-zones* [8,9], where a group of nodes do not transmit any information (or identifiers); on leaving the mix-zone the communication resumes with a new identifier or pseudonym for each user or device. Mix-zones serve to mix or provide de-correlation between pseudonyms and device associations, which makes it difficult for an adversary to continuously track users by linking the device and its pseudonym. Let us focus on pervasive and mobile networking scenarios where users or devices generally move on a fixed (pre-defined) network of roads, for example, vehicular or pedestrian handheld networking scenarios. Earlier research has shown that mix-zones in such networks are most effective (in protecting privacy) when they are defined at points with higher input and output ports such as *road intersections* [11].

Although effective in improving the privacy of users, the pseudonym-change (or mix) operation is not free and induces a cost on the network (and its users), which is determined by factors such as the significance of the intersection to users and the network, traffic intensity at the intersection (both entering and leaving) and intersection context, for example, time-of-day. This cost is primarily due to the loss in communication due to routing disruptions [43] or silent periods [29] and the loss of computation resources such as energy due to the pseudonym change operation itself and its related side-effects.

This results in an interesting trade-off between the number of mix-zones that can be deployed on the road network for privacy enhancement and the resulting cost due to such a deployment. An ideal situation from the privacy perspective, although infeasible from the cost point-of-view, is to deploy a mix-zone at each and every intersection of the road network under consideration. Such a deployment of mix-zones is trivial in theory, but difficult to realize and sustain in practice due to the resulting costs. A more realistic and feasible goal would be to maximize the coverage (of roads) of the deployed mix-zones, and hence the privacy provided by them, and to minimize the associated costs due to such a deployment. Moreover, the goal is not only to determine such an optimal and cost-efficient placement of mix-zones but also to study if there are algorithms that can find such a solution efficiently (in computation time and space). This is because, as pseudonym change costs at intersections are highly dynamic and depend on factors such as intersection context and traffic intensity that continuously change over time, there is a need to regularly determine the optimal and most cost-efficient set of mix-zones. In order to design efficient algorithms for the above optimization problem, a thorough theoretical analysis of the problem from a combinatorial perspective is first required.

In this paper, we model the problem of optimal mix-zone placement as a graph-based optimization problem where roads are represented by graph edges and intersections by vertices. Vertices are weighted based on the cost (per device) of mix-zone placement at each vertex and edges are weighted based on the demand or traffic intensity of the corresponding road in *each direction*. The problem of optimal mix-zone placement - we refer to as the *Mix Cover problem (MC)* - is then to determine a set of intersections (or vertices) for mix-zone placement, such that all the roads in the network are associated with at least one mix-zone and the overall cost of the mix-zone placement is minimized. The mix cover problem nicely models the mix-zone placement problem in pervasive networks and is a generalization of the well-known Vertex Cover (VC) problem [32], and a special case of the Facility Terminal Cover (FTC) problem [47]. To the best of our knowledge, this generalization, and specifically in the setting of pervasive networks, has not been addressed in the literature. We show that the mix cover problem is a combinatorially hard problem and propose three bounded-ratio approximation algorithms for the same. The first algorithm is based on a linear programming relaxation of an Integer Program (IP) formulation of the problem, whereas the remaining two algorithms take advantage of the "divide and conquer" strategy of [47] which was used to solve the FTC problem. We analytically study the solution quality and running-time guarantees of the algorithms by deriving their worst-case approximation ratio and running-time, respectively. Finally, we perform an extensive comparative analysis of the proposed algorithms by evaluating them on real US road-traffic data.

2 Background and Related Work

In the following section, we provide a short overview of concepts in complexity theory and combinatorial optimization used throughout the paper and outline other research efforts on the mix-zone placement problem.

2.1 Preliminaries: Combinatorial Hardness and Approximations

A decision problem $S \subseteq \{0,1\}^*$ is said to have an *efficiently verifiable proof system* if there exists a polynomial p and a polynomial-time verification algorithm V such that the following two conditions hold:

- Completeness: For every $x \in S$, there exists y of length at most $p(|x|)$ such that $V(x, y) = 1$.
- Soundness: For every $x \notin S$ and every y, it holds that $V(x, y) = 0$.

The class NP is the class of decision problems that have an efficiently verifiable proof system. A polynomial-time computable function f is called a *Karp-reduction* of S to S' (in other words, S is Karp reducible to S') if, for every x, it holds that $x \in S$ if and only if $f(x) \in S'$. A set S is *NP-complete* if it is in NP and every set in NP is Karp-reducible to it. A set S is *NP-hard* if every set in NP is Karp-reducible to it, but its membership within NP is not known. It is not known whether every problem in NP can be efficiently (in polynomial time) solved. But, if any single problem in the set of NP-hard problems can be solved efficiently, then every problem in NP can also be solved efficiently. Thus, NP-hard problems are considered "harder" than NP problems in general, and are believed to have no polynomial-time exact solutions. Algorithms for such hard problems, also called *optimization problems*, that run in polynomial time and produce a near-exact or sub-optimal solution are called *approximation algorithms*. Approximation algorithms that can guarantee that the solution output by them can be no more (if minimization problem) or less (if maximization problem) than a factor σ times the optimal solution is called a *σ-approximation algorithm* for that problem. More details on these topics can be found in [22,24].

2.2 Mix-Zone Placement Problem

The concept of using mix-zones in road networks, as a means to improve the location privacy of the mobile devices, has been proposed in [29,11,18]. Freudiger et al. [19] were the first to study and formulate the problem of optimal mix-zone placement in road networks. Here, the authors measure the *effectiveness* of mixing by measuring the probability of error of an adversary in correctly assigning exiting flows to their corresponding entering flows at a mix-zone. By using linear programming, they determine an optimal set of mix-zones that maximize the overall mixing effectiveness. In contrast, our model and solution is more general where we study the trade-off between maximizing the *coverage* of mix-zones and minimizing their *deployment cost*.

In another related effort, Alpcan and Buchegger [5] use game theory to model the attack and optimal defense strategies of the adversary and users in vehicular networks. Humbert et al. [31] also study the problem of optimal mix-zone placement from a game-theoretic perspective. They model the problem of mix-zone placement as a game between mobile users who want to protect their privacy and a local adversary who wants to track them by strategically placing eavesdropping stations. Here, the authors focus on deriving mix-zone deployment strategies locally at each intersection, whereas in our work, we study the problem of achieving a globally optimal deployment strategy. Palanisamy et al. [38] propose a framework and a suite of algorithms for mix-zone construction, which considers the inherent characteristics of road networks. Similar to earlier results, these mix-zone deployment strategies protect against specific adversarial attacks and only consider local intersection parameters for mix-zone deployment.

We extend the state of the art in optimal mix-zone deployment as follows. First, we study the problem of optimal mix-zone deployment from a global (network-wide) perspective. Moreover, our model and assumptions are generic enough to include other privacy metrics [44,45], in addition to the basic mix-zone coverage guarantee. Second, the analytical results obtained in this paper shed light on the feasibility of optimally deploying mix-zones in dynamic real-time road-network settings autonomously by mobile devices. Finally, the results outlined in this paper are also significant from the combinatorics viewpoint, as the generalization of the VC problem studied here has not been discussed before in the literature.

3 Problem Statement

3.1 System Model

Consider a wireless and pervasive mobile networking system where users (or vehicles) carry wireless communication devices that can either communicate with each other in a peer-to-peer fashion or through infrastructure-based base station(s). Examples of such networking systems include, but are not limited to, wireless peer-to-peer mobile-phone based pervasive social networking platforms such as Nokia Instant Community (NIC) [41] and vehicle-based wireless communication systems or VANETs [23]. Each mobile device in the network includes some identifying information or pseudonym in its communication, such as a MAC address or an application-level identifier, which is used for identifying the device and for routing communications within the network [40].

In order to prevent trivial tracking by an eavesdropping adversary, wireless devices regularly change their identifiers or pseudonyms. Various techniques for privacy protection, which use multiple pseudonyms or identifiers, have been studied in the literature [9,35,11,12]. In order to prevent trivial linkability of old and new pseudonyms, devices must coordinate their pseudonym changes, in space and time, with other devices, in order to achieve spatial and temporal de-correlation. Such regions for achieving spatial and temporal de-correlation of devices and (old and new) pseudonyms are also referred to as *mix-zones* [9]. In a mix-zone, spatial de-correlation is achieved by mobile device(s) changing their pseudonyms in a coordinated fashion whereas temporal de-correlation is achieved by either remaining silent for a short period of time [29], by encrypting communications after pseudonym change [18], or by means of a mobile proxy [42]. Mix-zones can be *passive* or *active*, depending on the actions taken by the devices immediately after the pseudonym change operation [31]. We assume that an off-line Certification Authority (CA), run by an independent trusted third-party, loads the mobile devices with a set of pseudonyms prior to deployment.

Road *intersections* are considered to be good spots for mix-zone deployment (and coordinated pseudonym change operations) as they provide optimal spatio-temporal de-correlation, as also observed in [19,31]. It is clear that mix-zones incur a communication overhead [43] and thus must be carefully placed (with appropriate parameters [30]) in order to reduce the cost induced on the end-users and to provide high location privacy (or high user-identifier de-correlation). The cost of deploying a mix-zone at any intersection can be a weighted sum of various factors, including the extra resource requirements of devices for mixing and the resulting communication disruption due to mixing at that intersection. We do not quantify these parameters in this work, but we can use existing results in the literature for representing these costs [43,31].

We assume that all the intersections, over the area under consideration, are connected with each other by a network of roads. Each road can be used to reach either one of the intersection that it connects, i.e., there is a two-way movement of users (or devices, vehicles, etc.) on the road. The *demand* for an intersection on a road is the average number of users using the road to reach that particular intersection. Thus, each road has two demands, one for each intersection connected by the road. Accordingly, unidirectional roads have just one demand, i.e., the one in the direction of the intersection; the other demand is zero. For simplicity, we assume that any two intersections are connected only by a single road; multiple roads between any two intersection can be combined into a single road by simply adding their respective demands.

3.2 Privacy Requirement

Given the system model discussed above, we want to investigate the problem of determining the most effective and cost-efficient mixing strategy in large pervasive networking scenarios. In other words, we address the problem of determining an optimal selection of intersections for mix-zone deployment such that all the roads in the network are covered and the *overall* cost due to mixing is minimized. We say that a road is *fully-covered* if and only if *both* the end points (intersections) of the road have mix-zones deployed on it, i.e., there is mixing at both intersections of the road. A network is said to be *fully-covered* (or has a *full cover*) if and only if all the roads in the network are fully-covered.

It is easy to see that in the system model discussed above, a full covering of the network can only be achieved if and only if all the intersections in the network are selected for mixing or mix-zone deployment. Such a mixing or full covering strategy is not only trivial but also ideal from the privacy viewpoint. But from a cost perspective, such a covering may be difficult to achieve in practice due to the network size or infeasible due to the overall cost of mixing at the intersections.

Let us now define a more general version of the full cover described above, called *mix cover*. A network is said to be *mix covered* if and only if each of the roads in the network have at least one of its intersections where a mix-zone is deployed. A fully-covered network is also mix covered and some of the roads in a mix covered network may be fully-covered, i.e., both the intersections of a road may have mix-zones deployed. From the privacy perspective, a mix covered network can *guarantee* that any user (or device) traversing the road network can traverse *at most two* roads (or *at most one intersection*) without encountering a mix-zone. From the practical standpoint, a mix cover is a reasonable mixing strategy for most deployment scenarios and adversarial models. We focus on the problem of determining a cost-efficient mix cover by modeling it as a *graph-based optimization problem*, as discussed next.

3.3 Graph-Theoretic Framework and the Mix Cover (MC) Problem

Let us represent the road network described above by an *undirected graph* $G \equiv (V, E, w, d)$. There exists a vertex $v \in V$ corresponding to each intersection in the road network and $|V| = n$ is the total number of intersections (vertices[1]) in the area of the road network under consideration. Each road connecting any two intersections u and v is represented by an edge $e \equiv (u, v) \in E$, where E is the set of all edges (or roads) and $|E| = m$ is the total number of roads (edges). There exists only a single edge (u, v) between any two pair of vertices u and v in G. Given the undirected graph G, let $w : V \rightarrow \mathbb{R}^+$ be the *cost function* that assigns a positive cost to each vertex. The cost at each vertex represents the average cost (per user) of mix-zone deployment (or mixing) at that intersection; the higher the cost, the higher the amount of communication and device resources spent by each user for mixing at that intersection is. We represent by w_u the cost of a vertex $u \in V$. Let $d : E \rightarrow (\mathbb{R}^+, \mathbb{R}^+)$ be the *demand function* that assigns a pair of positive demands to each edge where each demand value in the pair represents the demand for a particular vertex connected by the edge. This demand pair could signify, in the case of vehicular (or pedestrian) networks, the average traffic (or pedestrian) intensity on the road in each direction. For any edge $e(u, v) \in E$, we represent the demand as $d_e = (d_e^u, d_e^v)$, where d_e^u, d_e^v is the demand on edge e for intersections u and v, respectively. The value of $d_e^u = 0$ if u is not one of the end points of the edge e.

Given the above graph representation of the road network, we are interested in the problem of efficiently determining the *optimal* mix cover of the network. Each vertex chosen in the mix cover should be able to handle the demands from all the edges it covers. In other words, each intersection should be able to accommodate even the largest demand made at it; we refer to this ability of each intersection as the *capacity* of the mix-zone at that intersection. The capacity at a vertex is zero if there is no mix-zone at that vertex. The optimality criteria is based on an assignment of capacities to vertices or intersections such that the demands of all edges are met and the overall cost minimized. Formally, we can represent the problem of determining the optimal mix cover, referred by us as the *Mix Cover (MC) problem*, in the graph $G \equiv (V, E, w, d)$ as a generalization of the *Vertex Cover (VC)* problem. VC is a fundamental problem in graph theory and a vertex cover of an undirected graph $G \equiv (V, E)$ is a subset of vertices $V_C \subseteq V$ which contains at least one vertex of all the edges in E and the VC problem is to determine a vertex cover V_C of the smallest cardinality. The VC problem is $NP-hard$ and the decision version of the same is known to be $NP-complete$ [32]. The Mix Cover (MC) problem can be formally defined as:

Problem 1. Given an undirected graph $G \equiv (V, E, w, d)$, where w is the cost function associated with the set of vertices and d is the demand function associated with the set of edges, as discussed above, determine a subset $V_{MC} \subseteq V$ and

[1] Readers should note that from this point on we will use "vertex" and "intersections" (similarly, "road" and "edge") interchangeably and the intended meaning will be implicit from the context.

a capacity $c(v)$ for each vertex $v \in V_{MC}$ such that for each edge $e \equiv (u, v) \in E$ at least one of the vertices u and v is in V_{MC} and associated with a capacity $c(u) \geq d_e^u$ and $c(v) \geq d_e^v$ respectively, and the total weighted cost, $\sum_{x \in V_{MC}} c(x)w_x$, of all vertices in V_{MC} is minimized.

Thus, given graph $G \equiv (V, E, w, d)$ of the road network, the MC problem determines a mix cover of the network such that the overall (network-wide) weighted cost of the mix cover is minimized. The total intersection cost at each intersection v is the intersection mixing cost w_v times the capacity $c(v)$ at v. The capacity at any intersection v is at least the maximum demand at that intersection from all roads covered by it. The overall (network-wide) weighted cost of the mix cover is the sum of all the total intersection costs at each intersection in the mix cover. Figure 1 shows one such feasible solution.

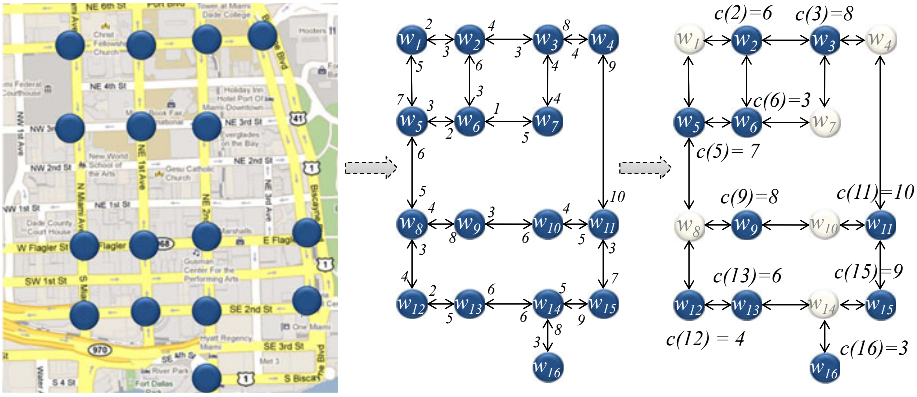


Fig. 1. Mix Cover example on downtown Miami (FL). On the left, the dark circles indicate all intersections where mix-zone placement is possible. The graph representation is shown in the middle and a feasible mix cover is shown on the right, where the dark circles are included in the solution and the shaded circles are not.

The MC problem is very similar to another generalization of the VC problem called the Facility Terminal Cover (FTC) problem [47,28], but there is an important difference between the two problems. Given a graph $G \equiv (V, E, w, d)$, where $w : V \rightarrow \mathbb{R}^+$ and $d : E \rightarrow \mathbb{R}^+$ (denoted as w_v and d_e for vertex v and edge e, respectively), the FTC problem is to find a set $V_{FTC} \subseteq V$ and a capacity $c(v)$ for each vertex $v \in V_{FTC}$ such that for each edge $e \equiv (u, v) \in E$ at least one of the vertices u and v is in V_{FTC} and associated with a capacity $c(u) \geq d_e$, and the total weighted capacity $\sum_{x \in V_{FTC}} c(x)w_x$ is minimized. As we can see from the FTC problem definition, the assumed graph model assigns only a *single* demand value to an edge and so the selected capacity for covering any edge depends only on the demand value for this same edge. This is different from the MC problem where each edge has two demand values and the selected capacity for covering any edge depends on the (demand value associated with the) vertex that is used to cover that same edge. The FTC problem can be considered as a special case of the MC problem, i.e., the MC problem reduces to the FTC problem when both the demand values are equal for all the edges in the graph. The formulation and algorithms of the FTC problem cannot be directly used to solve the much more general MC problem; although we will use one of the solution strategy [47] of the FTC problem for solving the MC problem.

There is another generalization of the VC problem called the minimum Generalized Vertex Cover (GVC) problem [27]. In GVC, contrary to VC, an edge incurs a cost (or demand, as in our case) depending on the *number* of its vertices that belong to the solution. Once again, such a generalization of the VC is different from the one that we are interested in, because in our case the demand incurred by the edge does not depend on the number of its vertices in the solution rather on which vertex is included in the solution. To the best of our knowledge, this is the first paper to model and study the problem of optimal mixing or mix-zone placement in pervasive networks as a unique generalization of the VC problem, which we believe has not been studied before.

4 Algorithms and Combinatorial Results

Let us first analyze the combinatorial hardness of the MC problem. We state the following theorem for the hardness of the MC problem.

Theorem 1. *The MC problem is NP-hard.*

Theorem 1 is straightforward, as we can easily reduce any instance of the weighted vertex cover problem to an instance of the MC problem in polynomial time. This can be done by defining a simple demand function for the graph instance of the weighted vertex cover problem as $d_e \equiv (d_e^u = 1, d_e^v = 1), \forall e$, where $e \equiv (u, v)$ is an edge of the graph instance. Thus, as the weighted vertex cover is NP-hard, we can claim that the MC problem is also NP-hard. The MC problem also seems difficult to approximate and we do not believe it has a Polynomial-Time Approximation Scheme ($PTAS$). This is because the VC problem itself, which is considered to be much simpler than the MC problem, is not believed to have an approximation ratio within 1.3606 unless $P = NP$ [16]. In the following sections, we outline two approximation strategies for the MC problem. The first is based on a linear programming formulation of the problem, whereas the other two algorithms employ a "divide and conquer" strategy by utilizing the round and group approach for solving the FTC problem [47].

4.1 Linear Programming Algorithm

We first formulate the MC problem as an Integer Program (IP), more specifically a 0-1 Program. Let z_e^v be a binary decision variable for each edge e and its corresponding vertex v which indicates whether the vertex v is included in the mix cover (solution) to cover edge e or not, i.e., $z_e^v = 1$ if edge e is covered by vertex v and $z_e^v = 0$ if not. Let x_v be the decision variable indicating the capacity and w_v indicate the cost of each vertex $v \in V$. Then, the IP formulation of the MC problem can be obtained as follows:

$$\begin{aligned} \min \ & \sum_{v \in V} w_v x_v \\ \text{subject to} \ & z_e^u + z_e^v \geq 1, \forall e \equiv (u,v) \in E \\ & x_v \geq z_e^v d_e^v, \forall v \in V, e \in E \\ & z_e^v \in \{0,1\}, \forall v \in V, e \in E \end{aligned}$$

Now, solving an Integer Program is a well-known hard problem [32]. Fortunately, efficient (polynomial time) techniques [6] exist for solving a Linear Program (LP) relaxation of the Integer program. If the LP relaxation has an integral solution then that can also be the solution to the above IP. In general, solving the LP relaxation of the problem can give a fractional feasible solution, from which a feasible (and possibly non-optimal) solution to the above IP can be obtained. The LP relaxation of the problem is as shown below:

$$\begin{aligned} \min \ & \sum_{v \in V} w_v x_v \\ \text{subject to} \ & d_e^v x_u + d_e^u x_v \geq d_e^u d_e^v, \forall e \equiv (u,v) \in E \\ & x_v \geq 0, \forall v \in V \end{aligned}$$

Let $(\bar{x}, \{\bar{z}_e | \forall e \in E\})$ be an optimal solution to the above LP formulation, where $z_{e,i} = \frac{x_i}{d_e^i}$ is the entry of the vector $\bar{z}_e$ representing the value of the decision variable corresponding to vertex i (to cover edge e), and x_j is the j^{th} entry of $\bar{x}$ and represents the capacity value at the vertex j. The value of $z_{e,i} = 0$ if i is not a vertex in edge e. We can see that any optimal solution $(\bar{x}, \{\bar{z}_e\})$ produced by solving the above LP is a feasible fractional solution to the MC problem. It is also clear that an optimal solution OPT to the MC problem is always a feasible solution to the above LP formulation. Thus, the above LP relaxation for the MC problem is correct. Based on this, we can prove the following bound on the approximation quality for the MC problem.

Theorem 2. *There exists a polynomial time 2-approximation for MC.*

For conciseness, the proof of this theorem has been moved to the Appendix. Theorem 2 shows that a constant ratio approximation is possible for the MC problem. Algorithms for linear programming, such as the simplex algorithms [15], are efficient in practice with a polynomial (in number of constraints) number of iterations, excluding the number of arithmetic operations [36]. But, Klee and Minty [34] showed that the number of iterations performed by some variants of the simplex can be exponential. Moreover, there is always a possibility, depending on the demand values, of the method producing an unbounded or an infinite number of solutions. To overcome these problems, we take advantage of a deterministic linear-time (in number of edges) approach for FTC proposed by Xu et al. [47], as discussed next.

4.2 "Divide and Conquer" Algorithms

In their algorithm, Xu et al. divide the input graph instance into multiple subgraphs by first rounding the edge demands and then grouping them based on the rounded edge values. They then apply the Weighted Vertex Cover (WVC) algorithm on each subgraph to obtain the solution to the FTC problem on the input graph. They show that their algorithm produces a 8-approximation when a 2-approximation algorithm [7,26] is used for WVC.

One of the main differences between FTC and MC is that in FTC the input graph instance has all edges with a single demand value, whereas in the MC problem, each edge has two demands, depending on the vertex chosen to cover the edge. Moreover, the MC problem is not directly reducible to the FTC problem unless the two demand values for each edge are equal. Below we outline two algorithms for solving the MC problem; they utilize the round and group strategy of [47]. In order to take advantage of their approach to solve the MC problem, we first need to transform the input graph instance so that all edges have equal demand values. Based on how this transformation is done, we will later see that the overall solution quality is accordingly influenced.

Largest Demand First (LDF) Algorithm. In our first, and more straightforward approach, we transform an input instance of the MC problem from $G \equiv (V, E, w, d)$ to $G' \equiv (V, E, w, d')$ such that, for each edge, both the new demands of the edge are equal and with value equal to the larger of the two original demand values. The intuition behind such a transformation is that if a vertex is able to cover the larger demand, then it will definitely be able to cover any demand smaller or equal to the larger demand. Then, the final demand values of the edges in the new graph instance G' are *rounded* off to the closest power of 2 of the larger demand value chosen in the previous step. Lemma 1 shows that any solution of the MC problem on such a transformed version (G') of the original graph is also a feasible solution for the MC problem on original graph (G). After obtaining G', it is first divided into subgraphs (G_k) based on the rounded edge demands (2^k), with each subgraph containing only edges of the same demand value. A known minimum WVC algorithm (such as [7,26]) is then used to obtain the minimum weighted vertex cover for each subgraph G_k. The mix cover is finally obtained by combining solutions from each of the individual subgraphs in the previous step. The LDF algorithm is outlined in Algorithm 1.

Lemma 1. *Any solution to the MC problem on the transformed graph instance $G' \equiv (V, E, w, d')$ is also a feasible solution to the MC problem on the original graph instance $G \equiv (V, E, w, d)$. Moreover, $OPT(G') \leq 2\alpha OPT(G)$, where $OPT(.)$ is the optimal solution and $\alpha = \max\{|d_e^u - d_e^v| \mid \forall e \equiv (u, v) \in G\}$, i.e., the maximum difference, over all edges, between the two edge demand values.*

We have the following result for the solution quality and running time of LDF.

Theorem 3. *The LDF algorithm is a linear time (in terms of the number of edges and vertices), $4\alpha\beta$-approximation algorithm for the MC problem, where*

Algorithm 1: Largest Demand First (LDF) Algorithm

```
input  : A graph $G \equiv (V, E, w, d)$.
output: A mix cover $S_{MC} \equiv (v, c(v))$ of $G$, where $v \in V$ and $c(v)$ is the capacity assigned
        to $v$.
for all $e \equiv (u, v) \in E$ do
    $d'_e \equiv (d'^u_e = \max\{d^u_e, d^v_e\}, d'^v_e = \max\{d^u_e, d^v_e\})$;
    if $2^{k-1} \leq d'^u_e = d'^v_e \leq 2^k$ then
        $d'_e \equiv (d'^u_e = 2^k, d'^v_e = 2^k)$;
    end
end
Let $G' \equiv (V, E, w, d')$;
Let $G_k \equiv (V_k, E_k, w)$ be a subgraph of $G'$ induced by edges $E_k = \{e \in E | d'_e = 2^k\}$;
for all $G_k$ do
    if $V_k \neq \phi$ then
        $S_{G_k}$ = WVC-2Approx($G_k \equiv (V_k, E_k, w)$);
    else
        $S_{G_k} = \phi$;
    end
end
$S_{MC} = \phi$;
for all $S_{G_k}$ such that $S_{G_k} \neq \phi$ do
    $c(v) \leftarrow \max\{2^k | \forall k$ s.t. $v \in S_{G_k}\}$;
    $S_{MC} \leftarrow (v, c(v))$;
end
```

$\beta > 1$ is the approximation ratio of the minimum WVC algorithm used and α is as defined in Lemma 1.

The proof of Lemma 1 and Theorem 3 can be found in the Appendix. Now, let us present a second solution strategy based on a transformation that chooses the smaller of the two demand values.

Smallest Demand First (SDF) Algorithm. In the LDF algorithm, we transform the input graph instance into an instance where the smaller edge demand is replaced by the larger one. This guarantees that each edge has the same (and a single) demand value and that the mix cover of such a transformed instance is also a feasible mix cover of the original instance. In practice, it is clear that such a strategy will produce a highly sub-optimal solution because there may be vertices in the final solution that may cover edges with much lower actual demand values. In order to overcome this issue, we propose another strategy for solving the MC problem, called the Smallest Demand First (SDF) algorithm.

This SDF algorithm, as outlined in Algorithm 2, consists of three phases. In the first phase, in contrast to the LDF algorithm, we transform the input graph instance $G \equiv (V, E, w, d)$ of the MC problem into an instance $G'' \equiv (V, E, w, d'')$ where the larger edge demand is now replaced by the smaller one. In this phase, an additional task during edge demands transformation is to remember the largest demand (d^v_{max}) to be covered at each vertex. In the second phase, similar to the LDF algorithm, we use the round and group strategy to obtain a mix cover for the transformed instance. In the final phase, we assign capacities to the vertices based on the output of the previous phase and the largest demand d^v_{max} determined in the first phase. Lemma 2 gives the relationship between the MC problem on the transformed version G'' and the original graph G.

Algorithm 2: Smallest Demand First (SDF) Algorithm

```
input  : Graph $G \equiv (V, E, w, d)$.
output: Mix cover $S_{MC} \equiv (v, c(v))$ of $G$, where $v \in V$ and $c(v)$ is the capacity assigned to $v$.
for all $v \in V$ do
    $d^v_{max} = 0$;
end
for all $e \equiv (u, v) \in E$ do
    if $d^u_e > d^u_{max}$ then
        $d^u_{max} = d^u_e$;
    end
    if $d^v_e > d^v_{max}$ then
        $d^v_{max} = d^v_e$;
    end
    $d''_e \equiv (d''^u_e = \min\{d^u_e, d^v_e\}, d''^v_e = \min\{d^u_e, d^v_e\})$;
    if $2^{k-1} \leq d''^u_e = d''^v_e \leq 2^k$ then
        $d''_e \equiv (d''^u_e = 2^k, d''^v_e = 2^k)$;
    end
end
Let $G'' \equiv (V, E, w, d'')$;
Let $G_k \equiv (V_k, E_k, w)$ be a subgraph of $G''$ induced by edges $E_k = \{e \in E | d''_e = 2^k\}$;
for all $G_k$ do
    if $V_k \neq \phi$ then
        $S_{G_k}$ = WVC-2Approx($G_k \equiv (V_k, E_k, w)$);
    else
        $S_{G_k} = \phi$;
    end
end
$S_{MC} = \phi$;
for all $S_{G_k}$ such that $S_{G_k} \neq \phi$ do
    $c(v) \leftarrow \max\{\max\{2^k | \forall k \text{ s.t. } v \in S_{G_k}\}, d^v_{max}\}$;
    $S_{MC} \leftarrow (v, c(v))$;
end
```

Lemma 2. *$OPT(G'') \leq \frac{2}{\alpha} OPT(G)$, where $G'' \equiv (V, E, w, d'')$ is the shortest demand first transformation of the original graph instance $G \equiv (V, E, w, d)$, $OPT(.)$ is the optimal solution of the MC problem on the input graph insance and $\alpha = \max\{|d^u_e - d^v_e| \mid \forall e \equiv (u, v) \in G\}$, i.e., the maximum difference, over all edges, between the two edge demand values in the original graph instance.*

It is easy to see that a feasible solution for the MC problem on G'' may not necessarily be a feasible solution to the MC problem on the original graph instance G. Moreover, in the worst case, $OPT(G'')$ may include only those vertices that correspond to larger demand values in the original graph. This observation and an argument similar to Lemma 1 can be used to prove Lemma 2. For conciseness, we omit the details. We have the following result for the approximation ratio.

Theorem 4. *The Smallest Demand First or SDF algorithm is a 4β-approximation algorithm for the MC problem, where $\beta > 1$ is the approximation ratio of the minimum WVC algorithm. Moreover, the algorithm runs in $O(mn)$ time, where n is the number of vertices and m is the number of edges in the graph.*
The proof for Theorem 4 can be found in the Appendix. It is clear from Theorem 4 that the SDF algorithm guarantees the same approximation ratio as the deterministic algorithm of Xu et al. [47] but runs slower. We now evaluate the practical efficiency of the proposed approaches by executing them on real vehicular road-network data.

5 Empirical Evaluation

In this section, we evaluate the performance of the proposed algorithms by implementing them in Matlab and executing them on a multi-core desktop computer. For our experiments, we construct the input graph instance using real road-traffic data (intersections, roads and bi-directional AADT traffic intensities) from the official transportation databases for Florida [17] and Virginia [46]. The results are outlined in Table 1.

Table 1. Performance of the proposed Mix Cover algorithms on real road traffic data

		SMALL SIZE GRAPH 25% of municipalities			**MEDIUM SIZE GRAPH** 65-85% of municipalities			**FULL SIZE GRAPH** Entire State		
Tot. # of *v* / *e*	Florida	2557 / 2640			6326 / 6960			7557 / 8310		
	Virginia	2408 / 2514			5726 / 6728			5881 / 6952		
		Constant	**Uniform**	**Gaussian**	**Constant**	**Uniform**	**Gaussian**	**Constant**	**Uniform**	**Gaussian**
# of *v* in the MC solution	A 1 - Florida	1243	1267	1238	2891	2990	2909	3481	3600	3545
	A 2 - Florida	1217	1244	1216	2863	2949	2877	3452	3502	3452
	A 1 - Virginia	1346	1373	1350	3328	3404	3345	3523	3592	3532
	A 2 - Virginia	1376	1386	1364	3376	3426	3374	3550	3607	3551
Ratio MC solution obj. fct. / naïve obj. fct.	A 1 - Florida	0.51	0.40	0.48	0.49	0.38	0.46	0.49	0.38	0.46
	A 2 - Florida	0.45	0.35	0.42	0.43	0.34	0.40	0.43	0.34	0.40
	A 1 - Virginia	0.82	0.79	0.81	0.82	0.79	0.81	0.82	0.79	0.81
	A 2 - Virginia	0.79	0.76	0.78	0.79	0.77	0.79	0.80	0.77	0.79
Duration of the MC simulation [sec]	A 1 - Florida	13.08	15.84	16	54.56	58.52	65.48	68.43	71.84	83.73
	A 2 - Florida	14.89	18.26	18.37	59.59	62.16	71.1	77.34	77.94	93.36
	A 1 - Virginia	12.78	15.08	15.58	43.02	51.22	54.98	42.47	49.13	50.7
	A 2 - Virginia	13.45	16.05	16.53	46.68	54.47	57.58	46.17	51	54

For each state, we considered three different sizes of the respective road network graphs: a small graph that corresponds to 25% of the total number of municipalities, a medium (65-85%) and a full state graph. For each such graph, we evaluated the performance of the proposed algorithms for three vertex weight distributions, namely, constant, uniform and positive Gaussian. The constant distribution assigns the same weight (=1) to all vertices, the uniform draws the weights uniformly at random from the interval [1,100], whereas the Gaussian has an expected value of 50 and a standard deviation of 10. Based on these parameters and the traffic intensities (or vertex demands) for each state, we measured the ratio between the MC solution objective function and the worst-case (naïve) solution (which includes all vertices of the graph in the vertex cover), the number of vertices in the MC solution, the individual vertex capacities and the duration of the simulation. The values in Table 1 are averaged over 100 runs.

The results confirm that, as predicted by the analytical evaluation, SDF (A2) performs consistently better than LDF (A1) for all graph sizes. Compared to the naïve solution, the proposed algorithms achieve a lower mix-zone cost, as low as 34% of the trivial solution cost. For all combinations of parameters, the uniform distribution achieves the best (lowest) objective function ratio, followed by the Gaussian and the constant distributions. The uniform distribution, which assigns (on average) the same weight to an equal number of vertices, makes it easier to determine feasible capacities to vertices that have a lower weight, while still covering all edges of the graph. In the Gaussian scenario, most of the weights will be close to the mean, and thus the search for the vertices that minimize the weighted cost will be more complex, leading to a worse solution and more demanding in terms of computation time. In the case where all vertices have the same weights, there are no chances of finding a feasible solution consisting of vertices with a lower weight than others. Hence, the ratio of the MC solution to that of the naïve solution is the worst in this scenario.

Depending on size of the graph and the respective demands, the number of mix-zones to be deployed is between 46% (Florida) and 58% (Virginia) of the total number of vertices. In Florida, SDF performs slightly better than LDF as it requires a smaller number of mix-zones. Although the differences amount to 2-3% (up to 102 fewer mix-zones), such result is consistent across all graph sizes. In Virginia, on the contrary, LDF performs slightly better than SDF (up to 30 fewer mix-zones). This indicates that, although relatively small, the performance of the two algorithms are influenced by the road network topology, and further investigations are required in order to determine the effects of the road topology on the performance of the proposed algorithms.

Intuitively, as the traffic patterns evolve during the day in each region, such algorithms would be executed multiple times per day in order to adapt the solutions to the traffic intensities throughout the day. Regarding the execution efficiency, the experimental results show that a feasible solution to the MC problem can be determined in 13 sec (small graph) and 94 sec (full State graph), which is a reasonable requirement in case such computations are done in a dynamic fashion multiple times per day.

In order to avoid unbounded solutions in the LP formulation, we had to reduce the graph size (and thus the number of constraints) of the road network. Considering a reduced (Florida) graph with 515 vertices, we obtained a ratio of 0.24 between the objective function of the MC solution with respect to the naïve one, which is a better result than LDF and SDF, but the fraction of intersections with mix-zones to the total number of intersections increased to 97%. For such a small graph, the LP required between 29 seconds (constant distribution) and 66 seconds (Gaussian distribution), which corresponds to two and four times the requirement of LDF and SDF, respectively, with the same weight distributions. Similar relative differences were obtained when increasing the number of vertices from 515 to 1024, except that the durations grew by a factor of 20 as compared to LDF and SDF. The results suggest that the LP formulation yields on average better (lower) costs for the mix-zone deployment, at the expense of a significant increase in computation time and number of mix-zones. Hence, the LP approach appears to be better suited for smaller graphs with a lower intersection/road density, such as peripheral and rural areas.

6 Conclusion

We addressed the problem of optimizing mix-zone placement in pervasive networking applications by formulating it as a graph-based optimization problem, referred to as the Mix Cover or MC problem. We proposed three algorithms to solve the MC problem: the first algorithm is based on a LP relaxation of the problem and the remaining two approaches take advantage of a “divide and conquer” strategy proposed by Xu et al. [47]. We proved important analytical results, such as the solution quality and running-time guarantees, for the proposed approaches. In order to shed light on their feasibility in a realistic pervasive network setting, we performed extensive experimental evaluation of the proposed approaches with real road network and traffic data. Experimental results confirmed the analytical results and also showed that these approaches can compute an approximate mix cover, even for fairly large road networks, in a reasonable amount of time using standard computing resources.

References

1. http://en.wikipedia.org/wiki/Lovegetty
2. http://en.wikipedia.org/wiki/Bluedating
3. http://pleaserobme.com/
4. Ahtiainen, A., Kalliojarvi, K., Kasslin, M., Leppanen, K., Richter, A., Ruuska, P., Wijting, C.: Awareness networking in wireless environments: Means of exchanging information. In: IEEE Vehicular Technology Magazine (September 2009)
5. Alpcan, T., Buchegger, S.: Security games for vehicular networks. IEEE Transactions on Mobile Computing (2011)
6. Aspvall, B., Stone, R.: Khachiyan's linear programming algorithm. Journal of Algorithms 1(1), 1–13 (1980)
7. Bar-Yehuda, R., Even, S.: A linear time approximation algorithm for the weighted vertex cover algorithm. Journal of Algorithms (1981)
8. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Computing 2(1), 46–55 (2003)
9. Beresford, A.R., Stajano, F.: Mix zones: User privacy in location-aware services. In: PerCom Workshop (2004)
10. Buchegger, S., Schiöberg, D., Vu, L.-H., Datta, A.: Peerson: P2P social networking: early experiences and insights. In: EuroSys SNS Workshop, pp. 46–52 (2009)
11. Buttyán, L., Holczer, T., Vajda, I.: On the effectiveness of changing pseudonyms to provide location privacy in vANETs. In: Stajano, F., Meadows, C., Capkun, S., Moore, T. (eds.) ESAS 2007. LNCS, vol. 4572, pp. 129–141. Springer, Heidelberg (2007)
12. Buttyán, L., Holczer, T., Weimerskirch, A., Whyte, W.: Slow: A practical pseudonym changing scheme for location privacy in vanets. In: IEEE VNC (2009)
13. Chaum, D.: Untraceable electronic mail, return addresses and digital pseudonyms. Comm. ACM 24(2), 84–88 (1981)
14. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. J. Cryptology 1(1), 66–75 (1988)
15. Dantzig, G.B.: Linear Programming and Extensions. Princeton Press, Princeton (1963)
16. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex-cover. Annals of Mathematics (2005)
17. Florida State traffic data, http://www.dot.state.fl.us/planning/statistics/gis/trafficdata.shtm
18. Freudiger, J., Raya, M., Felegyhazi, M., Papadimitratos, P., Hubaux, J.-P.: Mix zones for location privacy in vehicular networks. In: Win-ITS (2007)

19. Freudiger, J., Shokri, R., Hubaux, J.-P.: On the optimal placement of mix zones. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 216–234. Springer, Heidelberg (2009)
20. Freudiger, J., Shokri, R., Hubaux, J.-P.: Evaluating the privacy risk of location-based services. In: Financial Cryptography (2011)
21. Gaonkar, S., Li, J., Choudhury, R.R., Cox, L.P., Schmidt, A.: Micro-blog: sharing and querying content through mobile phones and social participation. In: MobiSys, pp. 174–186 (2008)
22. Garay, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
23. Giordano, E., Tomatis, A., Ghosh, A., Pau, G., Gerla, M.: C-VeT: An Open Research Platform for Vanets: Evaluation of Peer to Peer Applications in Vehicular Networks. In: IEEE VTC (2008)
24. Goldreich, O.: Computational Complexity: A Conceptual Perspective (2008)
25. Golle, P., Partridge, K.: On the anonymity of home/Work location pairs. In: Tokuda, H., Beigl, M., Friday, A., Brush, A.J.B., Tobe, Y. (eds.) Pervasive 2009. LNCS, vol. 5538, pp. 390–397. Springer, Heidelberg (2009)
26. Gonzalez, T.F.: A simple LP-free approximation algorithm for the minimum weight vertex cover problem. Information Processing Letters 54(3), 129–131 (1995)
27. Hassin, R., Levin, A.: The minimum generalized vertex cover problem. ACM Trans. Algorithms 2 (January 2006)
28. Hochbaum, D., Levin, A.: The Multi-integer Set Cover and the Facility Terminal Cover Problem. Networks 53, 63–66 (2009)
29. Huang, L., Matsuura, K., Yamane, H., Sezaki, K.: Enhancing wireless location privacy using silent period. In: IEEE WCNC (2005)
30. Huang, L., Yamane, H., Matsuura, K., Sezaki, K.: Towards modeling wireless location privacy. In: PETS (2006)
31. Humbert, M., Manshaei, M.H., Freudiger, J., Hubaux, J.-P.: Tracking games in mobile networks. In: Alpcan, T., Buttyán, L., Baras, J.S. (eds.) GameSec 2010. LNCS, vol. 6442, pp. 38–57. Springer, Heidelberg (2010)
32. Karp, R.M.: Complexity of Computer Computations. Plenum Press, New York (1972)
33. Khiabani, M.: Metro-sexual (2009), http://bit.ly/theranMetroSexual
34. Klee, V., Minty, G.J.: How good is the simplex algorithm? In: Shisha, O. (ed.) Inequalities, vol. III, pp. 159–175. Academic Press, London (1972)
35. Li, M., Sampigethaya, K., Huang, L., Poovendran, R.: Swing & swap: user-centric approaches towards maximizing location privacy. In: WPES (2006)
36. Megiddo, N.: On the complexity of linear programming. In: Advances in Economic Theory, Fifth World Congress, pp. 225–268. Cambridge University Press, Cambridge (1987)
37. Merlin, C.J., Heinzelman, W.B.: A study of safety applications in vehicular networks. In: MASS 2005 (2005)
38. Palanisamy, B., Liu, L.: Mobimix: Protecting location privacy with mix zones over road networks. In: ICDE 2011 (2011)
39. Paulos, E., Goodman, E.: The familiar stranger: anxiety, comfort, and play in public places. In: CHI, pp. 223–230 (2004)
40. Pfitzmann, A., Köhntopp, M.: Anonymity, unobservability, and pseudeonymity - a proposal for terminology. In: International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability (2001)
41. Rhiain.: Nokia instant community gets you social, http://conversations.nokia.com/2010/05/25/nokia-instant-community-gets-you-social/

42. Sampigethaya, K., Huang, L., Li, M., Poovendran, R., Matsuura, K., Sezaki, K.: CARAVAN: Providing location privacy for VANET. In: ESCAR (2005)
43. Schoch, E., Kargl, F., Leinmüller, T., Schlott, S., Papadimitratos, P.: Impact of pseudonym changes on geographic routing in vANETs. In: Buttyán, L., Gligor, V.D., Westhoff, D. (eds.) ESAS 2006. LNCS, vol. 4357, pp. 43–57. Springer, Heidelberg (2006)
44. Shokri, R., Freudiger, J., Jadliwala, M., Hubaux, J.-P.: A distortion-based metric for location privacy. In: ACM WPES (2009)
45. Shokri, R., Theodorakopoulos, G., Le Boudec, J.-Y., Hubaux, J.-P.: Quantifying location privacy. In: IEEE S&P (2011)
46. Virginia State traffic data, http://www.virginiadot.org/info/2009_traffic_data.asp
47. Xu, G., Yang, Y., Xu, J.: Linear Time Algorithms for Approximating the Facility Terminal Cover Problem. Networks 50, 118–126 (2007)
48. Xu, Q., Mak, T., Ko, J., Sengupta, R.: Vehicle-to-vehicle safety messaging in dsrc. In: VANET (2004)

Appendices

Proof (Proof for Lemma 1). The first part of the lemma is straightforward. As the vertex and edge sets of both G and G' are the same, a cover for G' (that covers all edges of G') is also a cover for G. Moreover, as the edge demands are rounded off to the largest and then to the closest power of 2, the selected capacity of the vertices for a solution (or mix cover) in G' will always be greater than the demands of the corresponding edges in G. Thus, a mix cover for G' is also a feasible mix cover for G.

Now, let us prove the second part. Let $V_{OPT(G)}$ be the set of vertices of the optimal solution $OPT(G)$ for the MC problem on the graph instance G and let $C_{OPT(G)} = \{c(v_i)|v_i \in V_{OPT(G)}\}$ be the capacities assigned to each vertex in the optimal solution. Consider a solution S such that it has the same set of vertices as $V_{OPT(G)}$ and with capacities $C_S = \{2(c(v_i)+\alpha)|v_i \in S\}$, where α is as defined above. We can see that (S, C_S) is always a feasible solution to G'. This is because, firstly, we always select the larger demand value for each edge e. Thus, even in the worst case, where all vertices $v_i \in V_{OPT(G)}$ in the optimal solution $OPT(G)$ are such that $d_e^{v_i} < d_e^{v_j}, \forall e \equiv (v_i, v_j) \in E$, capacities selected in C_S will always overcome the difference in demands. Secondly, the rounded demands of each edge e is only at most twice that of the original (larger) demand. Thus,

$$\begin{aligned} OPT(G') &\leq \sum_{v_i \in S} 2(c(v_i) + \alpha) \\ &\leq \sum_{v_i \in S} 2\alpha c(v_i) \leq 2\alpha OPT(G) \end{aligned}$$

□

Proof (Proof for Theorem 2). For the sake of convenience, let us denote the IP formulation of the MC problem as IP-MC and its LP relaxation as LP-MC. It is easy to see that any optimal solution OPT to IP-MC is also a feasible solution to the LP-MC and has the same objective function value. Moreover, LP-MC is indeed a relaxation of IP-MC and an optimal solution $(\bar{x}, \{\bar{z}_e\})$ to LP-MC is a feasible fractional solution to IP-MC. Thus, we can see that the value of the objective function (as the objective functions for both the formulations are the same) of an optimal solution $(\bar{x}, \{\bar{z}_e\})$ to LP-MC is at most that of the optimal solution OPT to IP-MC. Now, given the optimal solution $(\bar{x}, \{\bar{z}_e\})$ to LP-MC, we know that for any $e \equiv (u, v) \in E$, as $z_e^u + z_e^v \geq 1$, at least one of the following $z_e^u \geq \frac{1}{2}$ or $z_e^v \geq \frac{1}{2}$ is true. Let us apply the following transformation δ to $(\bar{x}, \{\bar{z}_e\})$: If $z_{e,i} \geq \frac{1}{2}$, for any e and i, then $\delta(z_{e,i}) = 1$ and if $z_{e,i} < \frac{1}{2}$ then put $\delta(z_{e,i}) = 0$. Also, $\delta(x_i) = 2x_i$ if for any $i \in V$ there is $\delta(z_{e,i}) = 1$.

It is easy to see that $\delta(\bar{x}, \{\bar{z}_e\})$ is a feasible solution to the IP-MC problem. Moreover, the linearity of the objective function guarantees that the objective function value (or the total weighted cost) of $\delta(\bar{x}, \{\bar{z}_e\})$ is at most twice the objective function value of $(\bar{x}, \{\bar{z}_e\})$. As the cost of $(\bar{x}, \{\bar{z}_e\})$ is at most OPT, the cost of $\delta(\bar{x}, \{\bar{z}_e\})$ is at most two times the cost of OPT, i.e., $\delta(\bar{x}, \{\bar{z}_e\}) \leq 2 \cdot OPT$. Thus, $\delta(\bar{x}, \{\bar{z}_e\})$ is a 2-approximation of the MC problem. Moreover, as LP-MC can be solved in polynomial time [6], the proof follows. □

Proof (Proof for Theorem 3). Let $WVC{-}2Approx(G_k)$ denote the output (overall minimum weight) of applying a β-approximation minimum WVC algorithm to the subgraph G_k of G'. If $OPT(G_k)$ is the corresponding optimal solution, then we have the following inequality:

$$\text{WVC-2Approx}(G_k) \leq \beta OPT(G_k)$$
$$OPT(G_k) \geq \frac{1}{\beta}\text{WVC-2Approx}(G_k) \tag{1}$$

From Lemma 2 of [47] we know that,

$$OPT(G') \geq \frac{1}{2}\sum_{k=0}^{K} 2^k OPT(G_k) \tag{2}$$

where K is max. value of the exponent after the rounding. From Lemma 1,

$$OPT(G') \leq 2\alpha OPT(G) \tag{3}$$

From (1) and (2), we have

$$OPT(G') \geq \frac{1}{2}\sum_{k=0}^{K} \frac{2^k}{\beta}\text{WVC-2Approx}(G_k)$$
$$\geq \frac{1}{2\beta}\sum_{k=0}^{K} 2^k\text{WVC-2Approx}(G_k)$$

Combining the above inequality with Eqn. 3 we have,

$$2\alpha OPT(G) \geq \frac{1}{2\beta}\sum_{k=0}^{K} 2^k\text{WVC-2Approx}(G_k)$$
$$\sum_{k=0}^{K} 2^k\text{WVC-2Approx}(G_k) \leq 4\alpha\beta OPT(G) \tag{4}$$

The left-hand side of the inequality in (4) clearly denotes the objective function computed from the output of the LDF algorithm. From this inequality, it is clear that the LDF algorithm is a $4\alpha\beta$-approximation of the MC problem.

Now, let us observe the time complexity of the LDF algorithm. Tasks such as determining the larger demand per edge, rounding the demand value and assigning capacities can be completed in linear time, in the worst-case. Moreover, Bar-Yehuda and Even [7] showed that a 2-approximation can be obtained for a WVC problem in linear time. Thus, the LDF algorithm runs in linear time. □

Proof (Proof for Theorem 4). We use a similar argument that we use to prove Theorem 3. From Eqn. 1 in the proof of Theorem 3 we know that:

$$OPT(G_k) \geq \frac{1}{\beta}\text{WVC-2Approx}(G_k)$$

where, β is the approximation ration of the weighted vertex cover algorithm WVC-2Approx and G_k is the induced subgraph of G'' with edge demands 2^k. Now, let K'' be the maximum value of the exponent (of 2) after the rounding the shortest demands on each edge. From Lemma 2 of [47] we know that, $OPT(G'') \geq \frac{1}{2}\sum_{k=0}^{K} 2^k OPT(G_k)$. From Lemma 2 we have, $OPT(G'') \leq \frac{2}{\alpha}OPT(G)$

From the above we have,

$$\begin{aligned}
\frac{2}{\alpha}OPT(G) &\geq \frac{1}{2}\sum_{k=0}^{K} 2^k OPT(G_k)\\
4OPT(G) &\geq \alpha\sum_{k=0}^{K} 2^k OPT(G_k)\\
4\beta OPT(G) &\geq \alpha\sum_{k=0}^{K} 2^k \text{WVC-2Approx}(G_k)\\
4\beta OPT(G) &\geq \sum_{k=0}^{K} (2^k + \alpha)\text{WVC-2Approx}(G_k) \qquad (5)
\end{aligned}$$

The right-hand side of (5) clearly denotes an upper bound on the objective function computed by the SDF algorithm. From this inequality, it is clear that the SDF algorithm is a 4β-approximation algorithm for the MC problem.

Now, let us observe the time complexity of the SDF algorithm. The SDF algorithm has an additional step, as compared to the LDF algorithm, which is the largest demand determination on each vertex. It is easy to see that this step, in the worst case, takes $O(n^2)$ additional steps and thus one order of time more than the LDF algorithm. □

A New RFID Privacy Model[*]

Jens Hermans[**], Andreas Pashalidis, Frederik Vercauteren[***],
and Bart Preneel

Department of Electrical Engineering - COSIC,
Katholieke Universiteit Leuven and IBBT,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. This paper critically examines some recently proposed RFID privacy models. It shows that some models suffer from weaknesses such as insufficient generality and unrealistic assumptions regarding the adversary's ability to corrupt tags. We propose a new RFID privacy model that is based on the notion of indistinguishability and that does not suffer from the identified drawbacks. We demonstrate the easy applicability of our model by applying it to multiple existing RFID protocols.

Keywords: RFID, authentication, identification, privacy model.

1 Introduction

As Radio Frequency Identification (RFID) systems are becoming more common (for example in access control [10,30], product tracking [10], e-ticketing [27,30], electronic passports [18]), managing the associated privacy and security concerns becomes more important [34]. Since RFID tags are primarily used for authentication purposes, 'security' in this context means that it should be infeasible to 'fake' a legitimate tag. 'Privacy', on the other hand, means that adversaries should not be able to identify, trace, or link tag appearances.

Several models for privacy and security in the context of RFID systems have been proposed in the literature. In this paper, we critically examine some of these models. In particular, we focus on *general* models[1]. For some of these models we show that, despite their intended generality, it remains unclear how to apply

[*] This work was supported in part by (a) the Research Council K.U.Leuven: GOA TENSE (GOA/11/007), (b) the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), (c) the 'Trusted Architecture for Securely Shared Services' (TAS3) project, supported by the 7th European Framework Programme with contract number 216287, and (d) the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

[**] Research assistant, sponsored by the Fund for Scientific Research - Flanders (FWO).

[***] Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

[1] We do not discuss some of the early proposals that were made in the context of one specific protocol.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 568–587, 2011.

them to protocols other than the protocol in the context of which they were proposed. Other existing models do not support adversaries that can tamper with tags. However, considering such adversaries is important because, as low-cost devices, tags are hardly protected against physical tampering. In particular, it has been shown that side-channel attacks may enable an adversary to extract secrets from the tag [17,21,22,26], and so-called 'reset' attacks force the tag to re-use old randomness [3,9,15]. The adversary can mount reset attacks by inducing power drops or by otherwise influencing the physical environment of the tag. Adversaries that can tamper with tags are therefore realistic.

Subsequently we propose a new model that borrows concepts from previous models, including virtual tag references, the corruption model that Vaudenay [32] introduced and the notion of 'narrow' and 'wide' adversaries to construct a new model. We believe that the new model is easier to apply. Also note that, although presented as a model for RFID privacy, it is not limited to the RFID setting; the model may also apply to other setups, in which the participants should not be identifiable or linkable.

Structure of the paper. Section 2 introduces the basic definitions for RFID systems and some notation. Section 3 discusses a selection of existing models, their underlying assumptions, their usability, and some further technicalities. Section 4 presents our model for RFID privacy which is then applied to some of the stronger existing RFID protocols in Section 5. In the appendices, our model is extended to a multi-indistinguishability setup, which allows multi-bit challenges. Mutual authentication is also discussed there.

2 Definitions

Throughout this paper we use a common model for RFID systems, similar to the definitions introduced in [8,32]. An RFID system consists of a set of tags $\mathcal{T}$, and a reader R. Each tag is identified by an identifier ID. The memory of the tags contains a state S, which may change during the lifetime of the tag. The tag's ID may or may not be stored in S. Each tag is a transponder with limited memory and computation capability.

Tags can also be corrupted: the adversary has the capability to extract secrets and other parts of the internal state from the tags it chooses. The reader R consists of one or more transceivers and a central database. The reader's task is to identify legitimate tags (i.e. to recover their IDs), and to reject all other incoming communication. The reader has a database that contains for every tag, its ID and a matching secret K.

Definition 1 (RFID Framework [32]). *An RFID scheme consists of the following algorithms:*

- *SetupReader*(1^k)*: setup the reader by generating the necessary keys, depending on the security parameter k. The function returns the public and private keys of the reader. Public keys are assumed to be publicly released by the algorithm, private keys are stored in the reader.*

- *SetupTag(ID): return the tag specific secret K and the initial state S of the tag. The pair (ID, K) will be stored in the reader, the state S in the tag. Note that K is not necessarily stored in the tag, but the definition of the protocol might include K in the state S.*
- *Protocol: a polynomial-time interactive protocol between a reader and a tag. The reader ends with a tape output.*

All the models discussed below fit the above general RFID system definition.

A function $f : \mathbb{N} \to \mathbb{R}$ is called 'polynomial' in the security parameter $k \in \mathbb{N}$ if $f(k) = O(k^n)$, with $n \in \mathbb{N}$. It is called 'negligible' if, for every $c \in \mathbb{N}$ there exists an integer k_c such that $f(k) \leq k^{-c}$ for all $k > k_c$. We denote a negligible function by ϵ.

If $\mathcal{T}$ is a set, $t \in_R \mathcal{T}$ means that t is chosen uniformly at random from $\mathcal{T}$. $|\mathcal{T}|$ denotes the cardinality of the set. If $\mathcal{A}$ is an algorithm, then $\mathcal{A}^{\mathcal{O}}$ denotes the fact that $\mathcal{A}$ has access to the oracle $\mathcal{O}$.

3 Existing Privacy Models

This section discusses certain existing RFID privacy models. Most models feature a correctness (no false negatives), security (no false positives) and privacy definition.

Note that covering all existing models would exceed the scope of this paper by far. Many models, including the ones introduced in [2,7,11,14,16,20,31] do not allow corrupted tags to be traced. We have selected two such models [14,20] for further discussion, in addition to the stronger models of Vaudenay [32] and Canard et al. [8].

3.1 Vaudenay

Several concepts from the privacy model introduced by Vaudenay [32] are used in our model. We therefore present this in detail.

Adversarial model. The adversary of the Vaudenay model has the ability to influence all communication between a tag and the reader and can therefore perform man-in-the-middle attacks on any tag that is within its range. It may also obtain the result of the authentication of a tag, i.e. whether the reader accepts or rejects the tag. The adversary may also 'draw' (at random) tags and then 'free' them again, moving them inside and outside its range. During these interactions the adversary has to use a virtual identifier (not the tag's real ID) in order to refer to the tags that are inside its range. Finally the adversary may corrupt tags, thereby learning their entire internal state.

The above interactions take place over eight oracles that the adversary may invoke: `CreateTag(ID)`, `DrawTag(distr)` $\to$ $(vtag)$, `Free(`$vtag$`)`, `Launch` $\to \pi$, `SendReader(`m, π`)` $\to m'$, `SendTag(`$m, vtag$`)` $\to m'$, `Result(`π`)` $\to x$ and `Corrupt(`$vtag$`)`. $vtag$ denotes a virtual tag reference, π a protocol instance, $distr$

a polynomially bounded sampling algorithm, m and m' messages and ID a tag ID. For a complete definition of the oracles the reader is referred to [32].

The Vaudenay model divides adversaries into different classes, depending on restrictions regarding their use of the above the oracles. In particular, a *strong* adversary may use all eight oracles without any restrictions. A *destructive* adversary is not allowed to use a tag after it has been corrupted. This models situations where corrupting a tag leads to the destruction of the tag. A *forward* adversary can only do other corruptions after the first corruption. That is, no protocol interactions are allowed after the first corrupt. A *weak* adversary does not have the ability to corrupt tags. Orthogonal to these four attacker classes there is the notion of *wide* and *narrow* adversary. A *wide* adversary has access to the result of the verification by the server while a *narrow* adversary does not.

Due to their generality, the above restrictions can be used perfectly in other privacy models. Throughout the paper we will frequently refer to strong, destructive, forward, weak and wide/narrow adversaries.

The equations below show the most important relations between the above privacy notions:

$$\begin{tikzcd}
\text{Wide Strong} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Wide Destructive} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Wide Forward} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Wide Weak} \arrow[d, Rightarrow] \\
\text{Narrow Strong} \arrow[r, Rightarrow] & \text{Narrow Destructive} \arrow[r, Rightarrow] & \text{Narrow Forward} \arrow[r, Rightarrow] & \text{Narrow Weak}
\end{tikzcd}$$

In this case $A \Rightarrow B$ means that if the protocol is A-private it implies that the protocol is B-private. A protocol that is *Wide Strong* private, for example, obviously also belongs to all other privacy classes, that only allow weaker adversaries.

Privacy, security and correctness. In general, an RFID protocol should satisfy (a) correctness (a 'real' tag is always accepted), (b) security (fake tags are rejected) and (c) privacy (tags cannot be identified or traced). Privacy is defined by means of the notion of a 'trivial' adversary. Intuitively, a trivial adversary does not 'use' the communication captured during the protocol run to determine its output.

Definition 2 (Blinder, trivial adversary - Simplified version of Definition 7 from [32]). *A Blinder $\mathcal{B}$ for an adversary $\mathcal{A}$ is a polynomial-time algorithm which sees the messages that $\mathcal{A}$ sends and receives, and simulates the* `Launch`*,* `SendReader`*,* `SendTag` *and* `Result` *oracles to $\mathcal{A}$. The blinder does not have access to the reader tapes. A blinded adversary $\mathcal{A}^{\mathcal{B}}$ is an adversary who does not use the* `Launch`*,* `SendReader`*,* `SendTag` *and* `Result` *oracles.*

An adversary $\mathcal{A}$ is trivial if there exists a blinder $\mathcal{B}$ such that $|\Pr(\mathcal{A}\,\textit{wins}) - \Pr(\mathcal{A}^{\mathcal{B}}\,\textit{wins})|$ *is negligible.*

Intuitively, an adversary is called trivial if, even when blinded, it still produces the same output. Such an adversary does not 'use' the communication captured during the protocol run in order to determine its output. Note that a blinded adversary is not the same as a simulator typically found in security proofs: the blinder is separate from the adversary and has no access to the adversary's tape. The blinder just receives incoming queries from the adversary and has to respond either by itself or by forwarding the queries to the system.

We are now ready to present the privacy definition.

Definition 3 (Privacy - Simplified version of Definition 6 from [32]). *The privacy game between the challenger and the adversary consists of two phases:*

1. *Attack phase: the adversary issues oracle queries according to applicable restrictions*
2. *Analysis phase: the adversary receives the table that maps every vtag to a real tag ID. Then it outputs* `true` *or* `false`*.*

The adversary wins if it outputs `true`*. A protocol is called P-private, where P is an adversary class (strong, destructive, ...), if and only if all winning adversaries that belong to the class P are trivial.*

Besides privacy the protocol should also offer authentication of the tag. We refer to this property as the *security* of the protocol.

Definition 4 (Security - Simplified version of Definition 4 from [32]). *We consider any adversary in the class* strong*. The adversary wins if the reader identifies an uncorrupted legitimate tag, but the tag and the reader did not have a matching conversation. The RFID scheme is called secure if the success probability of any such adversary is negligible.*

Definition 5 (Correctness - Definition 1 from [32]). *An RFID scheme is correct if its output is correct except with negligible probability for any polynomial-time experiment which can be described as follows:*

1. *set up the reader*
2. *create a number of tags including a subject one named ID*
3. *execute a complete protocol between reader and tag ID*

The output is correct if and only if $Output = \perp$ *and tag* ID *is not legitimate or* $Output = ID$ *and tag* ID *is legitimate.*

In a follow-up paper [25] to the Vaudenay paper, the concept of mutual authentication for RFID is defined. The tag simply outputs a boolean, indicating whether or not the reader was accepted. The authors extend the security definition by adding a criterion for reader authentication.

Discussion. The paper of Vaudenay inspired many authors to formulate derived RFID privacy models or to evaluate the (Paise-)Vaudenay model [6,8,12,13,23, 24,25,28,29]. Although Vaudenay's privacy model is perhaps the strongest and most complete, it contains some flaws with respect to strong privacy.

Vaudenay's proof of the statement that 'strong privacy is impossible' uncovers some of these flaws. This proof assumes a destructive private protocol. By definition, for every destructive adversary, there exists a blinder. This includes the adversary that (a) creates one real tag, (b) corrupts this tag right away, (c) starts a protocol using either the state from the corrupted tag or from another fake tag. In the end, the blinder has to answer the `Result` oracle. Obviously, the adversary knows which tag was selected and knows which result to expect. However, since the blinder has no access to this random coin of the adversary, it must be able to distinguish a real and a fake tag just by looking at the protocol run from the side of the reader. The proof then uses this blinder to construct a strong adversary. Since all strong adversaries are also destructive, this proves the impossibility of strong privacy.

Obviously, this proof only works because the blinder is separated from the adversary. In later work [33], Vaudenay corrects the inconsistency in the model and shows that strong privacy is indeed possible. In this new approach, the blinder is given access to the random coin flips of the adversary. The issue with a separate blinder is exploited multiple times by Armknecht et al. in [1]. Using this property the authors show the impossibility of reader authentication combined with respectively narrow forward privacy (if `Corrupt` reveals the temporary state of tags) and narrow strong privacy (if `Corrupt` only reveals the permanent state of tags).

Independent from this correction, Ng et al. [23] also identified the problems with strong privacy. They propose a solution, based on the concept of a 'wise' adversary that does not make any 'irrelevant' queries to the oracles i.e. queries to which it already knows the answer. The authors claim that, if the protocol does not generate false negatives, then a wise adversary never calls the `Result` oracle. Given the vague definition of wise adversaries it is hard to verify these claims. The existence of attacks which exploit false positives [4] however, suggests that the general claim that `Result` is not used by a wise adversary is incorrect. Based on this questionable general claim, the authors further identify an IND-CPA-based protocol as being strong private, without giving a formal proof.[2]

3.2 Canard et al.

Model. The model of Canard et al. [8] builds on the work of Vaudenay, so the definition of oracles is quite similar. For the privacy definition the model requires the adversary to produce a non-obvious link between virtual tags.

Definition 6. *$(vtag_i, vtag_j)$ is a non-obvious link if $vtag_i$ and $vtag_j$ refer to the same ID and if a 'dummy' adversary, who only has access to `CreateTag`, `Draw`, `Free`, `Corrupt`, is not able to output this link with a probability better than* $^1/_2$.[3]

[2] Note that the original security proof (i.e. no false positives) by Vaudenay requires IND-CCA2 encryption, so using only IND-CPA encryption would require a new security proof. The `Result` may therefore serve as a decryption oracle.

[3] It is unclear why the authors use the probability threshold $^1/_2$, since one would expect some dependency on the total number of non-obvious links. One slightly different interpretation is that a 'dummy' adversary cannot determine if a given non-obvious candidate link $vtag_i$, $vtag_j$ is a link in reality or not.

One major difference with respect to Vaudenay's model is that a 'dummy' adversary is used instead of a blinded adversary. This avoids some of the issues surrounding the use of a blinder, because a 'dummy' adversary can also access its own random tape, while a blinder cannot access the adversary's random tape.

The definition requires the adversary to output a non-obvious link. A protocol is said to be untraceable if, for every adversary $\mathcal{A}$, it is possible to construct a 'dummy' adversary $\mathcal{A}_d$ such that $|\mathbf{Succ}_{\mathcal{A}}^{Unt}(1^k) - \mathbf{Succ}_{\mathcal{A}_d}^{Unt}(1^k)| \leq \epsilon(k)$.

Discussion. While the work certainly has its merit in formalizing and fixing the Vaudenay model (by using a dummy adversary instead of a blinder), the model of Canard et al. lacks generality because it focuses on non-trivial links. Other relevant properties, which do not imply the leakage of a non-trivial link, are not considered a privacy breach. For example, the cardinality of the set of active tags can be leaked without leaking a non-trivial link. Because of the limited scope of untraceability, we are not using this model.

3.3 Deng, Li, Yung and Zhao

Model. Deng et al. presented their RFID Privacy Framework in [14].

The correctness ('adaptive completeness') definition used by Deng et al. is more elaborate than Vaudenay's definition. In particular, it allows the adversary to execute multiple complete protocol runs. This captures 'desynchronization' attacks where the adversary communicates a number of times with a tag (without involvement of the reader), in order to desynchronize the tag's state such that it will no longer be recognised by the reader.

The security definition considers both tag-to-reader and reader-to-tag authentication. The definition is similar to Vaudenay's since it requires matching sessions at reader and tag side. In Deng et al.'s model the last message is always sent by the reader, so an adversary could just prevent the tag from finishing the protocol by dropping this last message. Deng et al. therefore define the notion of 'matching sessions' such that last message attacks do not breach security. Vaudenay omits an exact definition of 'matching sessions', and therefore issues like the last message attack are not captured.

While the correctness and security definitions of Vaudenay and Deng et al. appear to be, to a large extent, equivalent, there is a significant discrepancy in the privacy definitions. Firstly, there is no notion of virtual tags in Deng et al.'s model; instead the adversary can refer to all tags using their real identifiers. Secondly, the adversary cannot create new tags. Thirdly, Deng et al. apply a zero-knowledge proof instead of Vaudenay's blinder construction. Informally stated, in the zero-knowledge experiment, the adversary (in the real world) consists of these phases:

1. Standard interaction using the oracles.
2. Select one tag at random (the 'challenge' tag) from the set of clean (non-corrupted and non-active) tags.

3. Interaction using the oracles, except that the adversary can only interact with the non-clean tags and the challenge tag. Moreover, the challenge tag cannot be corrupted.
4. Output a view from the previous step and the index of the challenge tag.

The simulated world is the same, except that, in the third phase, the adversary cannot access the challenge tag. If all PPT adversaries can be simulated such that the output of the adversary and simulator are computationally/statistically indistinguishable, then the protocol is considered zk-private. This implies that for all adversaries the output can actually be derived without interacting with the challenge tag (as the simulator does).

Discussion. Because of the very specific restrictions imposed in the third phase, this model is significantly weaker than Vaudenay's. Firstly, the model focuses on deriving information about a specific challenge tag (selected by the adversary), while in Vaudenay's model *any* statement that reveals information on the underlying identity of any of the tags is considered a privacy breach. Secondly, the adversary's ability to corrupt tags is limited. In Vaudenay's (corrected) strong privacy model one could prove that a protocol satisfies the privacy definition even if the 'challenge' tag is corrupted. The restriction that the challenge tag must be clean is, according to the authors, introduced to ensure that the tag is not stuck halfway a protocol run. Otherwise one can trivially distinguish the challenge tag by checking whether or not it responds to the remainder of the protocol run. Since a protocol run takes only a short timespan, obviously linking two protocol messages from the same run to the same tag should not be considered a privacy breach. However, we believe that, for the purposes of excluding this as a privacy breach, the concept of virtual tags is more suitable than overly limiting the adversary's corruption abilities in this manner.

The zero-knowledge private protocol proposed in [14] uses a counter as the tag state. The value of this counter is incremented after each protocol run completed by the tag. Obviously, this protocol does not satisfy the privacy definition if the adversary can corrupt the targeted tag, because the adversary learns the value of the counter (and the key) and, by decrementing the value of the counter, it can identify previous protocol runs of the targeted tag. The model in [14] has however been specifically tuned to disallow corruption of the challenge tag, which is a rather unrealistic assumption and thus undermines the significance of the claims that follow from its application.

The security and correctness definitions are more rigorous than Vaudenay's, so they can be a valuable alternative to them.

3.4 Juels-Weis

Model. The Juels-Weis model [20] is based on the notion of indistinguishability. The model does not feature a `DrawTag` query and the `Corrupt` query is replaced by a `SetKey` query, which returns the current secret of the tag and allows the adversary to set a new secret. Figure 3.4 shows a simplified version of the privacy

Experiment $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}$:

1. Setup:
   - Generate n random keys key_i.
   - Initialize the reader with the random key_i.
   - Create n tags, each with a key_i.
2. Phase (1): Learning
   - $\mathcal{A}$ can interact with a polynomial number of calls to the system, but can only issue `SetKey` on $n-2$ tags, leaving at least 2 uncorrupted tags
3. Phase (2): Challenge
   - $\mathcal{A}$ selects two uncorrupted tags T_0 and T_1. Both are removed from the set of tags.
   - One of these tags (T_b, the challenge tag) will be selected at random by the challenger.
   - $\mathcal{A}$ can make a polynomial number of calls to the system, but cannot corrupt the challenge tag T_b.
   - $\mathcal{A}$ outputs a guess bit $g \in \{0,1\}$.

Fig. 1. Privacy experiment from [20]

game. The protocol is considered private if $\forall \mathcal{A}, \Pr\left[\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}} \text{guesses } b \text{ correctly}\right] \leq \frac{1}{2} + \epsilon$

Discussion. The Juels-Weis model is one of the few models that are based on a simple indistinguishability game instead of the notion of simulatability. The model is limited by the fact that the challenge tags cannot be corrupted. In terms of the model in [32] it would be a Weak adversary with regard to the challenge tags. For example, attacks in which the adversary links together executions of a tag that have taken place prior to its corruption are not possible in the Juels-Weis model because of this.

The model from [16] is very similar, with the difference that the privacy is defined as distinguishing the reply from a real tag from a random reply.

3.5 Bohli-Pashalidis

Model. Unlike the previous models, the Bohli-Pashalidis model [5] is not an RFID-specific model. Unfortunately, it captures only privacy properties; properties like security and correctness are not covered. The model considers a set of users (with unique identifiers) $\mathcal{U}$, whose size is at least polynomial in a security parameter. There is no formal difference between different types of player, like there is with tag and reader in most RFID models. The system $\mathcal{S}$ can be invoked with input batches $(u_1, \alpha_1), (u_2, \alpha_2), \ldots, (u_c, \alpha_c) \in (\mathcal{U}, A)^c$, consisting of pairs of user identifiers and 'parameters' and will output a batch $((e_1, \ldots e_c), \beta)$, with the outputs e_i from each system invocation and a general output β, applying to the batch as a whole. Users can also be corrupted, revealing their internal state to the adversary.

The authors investigate the properties of the function $f \in \mathcal{F}$, where $\mathcal{F} = \{f : \{1, 2, \ldots, n\} \rightarrow \mathcal{U}\}$ is the space of functions that map the serial number of each output element to the user it corresponds to. In the Strong Anonymity (SA) setting, no information should be revealed to the adversary about the function

f, guaranteeing the highest level of privacy. Several weaker notions (which reveal *some* information on f) are defined and the relations among notions are examined.

In the RFID setting the batch properties are currently not considered, although this would be an interesting extension, since some localization protocols are based on batch invocations of a large set of RFID tags. For simplicity we restrict ourselves to the Bohli-Pashalidis model for online systems. For these systems, where all batches have size one (i.e. the system never waits for multiple inputs until it produces some output), the only two applicable distinct notions are Strong Anonymity (SA) and Pseudonymity (PS).

The adversarial model is based on indistinguishability. The adversary can cause different users to invoke the system using different parameters (e.g. messages) in both a left and right world with the $\texttt{Input}((u_0, \alpha_0), (u_1, \alpha_1))$ oracle. Based on a bit b, selected by the challenger, the system will be invoked with the user-data pair (u_b, α_b). That is, the adversary itself defines the functions $f_0, f_1 \in \mathcal{F}$, for respectively the left and right world. The adversary can also corrupt users. At the end of the game the adversary has to output a guess bit g. The adversary wins the game if $g = b$. By imposing restrictions on f_0 and f_1, the authors investigate different levels of privacy.

Definition 7. *A privacy protecting system $\mathcal{S}$ is said to unconditionally provide privacy notion X, if and only if the adversary $\mathcal{A}$ is restricted to invocations (u_0, α_0) and (u_1, α_1) such that f_0 and f_1 are X-indistinguishable for all invocations and for all such adversaries $\mathcal{A}$, it holds that $\mathbf{Adv}^X_{\mathcal{S},\mathcal{A}}(k) = 0$.*

Similar definitions for computational ($\mathcal{A}$ is polytime in k and $\mathbf{Adv}^X_{\mathcal{S},\mathcal{A}}(k) \leq \epsilon(k)$) and statistical privacy are available.

Discussion. Due to its generality, and due to the fact that it is not meant to cover security properties, the Bohli-Pashalidis model needs non-trivial adaptations in order to apply to RFID setting. In its current form, the model does not support multi-pass protocols, where linking two messages from the same protocol run is not a privacy breach. Moreover there is no distinction between tags that need to be protected, and the reader for which privacy is not an issue. An interesting question is whether the strictly binary distinguishing game (only one bit of randomness in the challenge) provides enough flexibility compared to other models, like Vaudenay's, where there are multiple bits of randomness that are to be guessed.

4 Our Model

4.1 Adversarial Model and Privacy

We use the setup from Definition 1. We assume a central reader R and a set of tags $\mathcal{T} = \{T_1, T_2, \ldots, T_i\}$. $\mathcal{T}$ is initially empty, and tags are added dynamically by the adversary. The reader maintains a database of tuples (ID_i, K_i), one for every tag $T_i \in \mathcal{T}$. Moreover, every tag T_i stores an internal state S_i.

Let $\mathcal{A}$ denote the adversary, which can adaptively control the system $\mathcal{S}$. $\mathcal{A}$ interacts with $\mathcal{S}$ through a set of oracles. The experiment that the challenger sets up for $\mathcal{A}$ (after the security parameter k is fixed) proceeds as follows:

$\mathbf{Exp}^{b}_{\mathcal{S},\mathcal{A}}(k)$:

1. $b \in_R \{0,1\}$
2. $SetupReader(1^k)$
3. $g \leftarrow \mathcal{A}^{\texttt{CreateTag},\texttt{Launch},\texttt{DrawTag},\texttt{Free},\texttt{SendTag},\texttt{SendReader},\texttt{Result},\texttt{Corrupt}}()$
4. Return $g == b$.

At the beginning of the experiment, the challenger picks a random bit b. The adversary $\mathcal{A}$ subsequently interacts with the challenger by means of the following oracles:

- `CreateTag(ID)` $\rightarrow T_i$: on input a tag identifier ID, this oracle calls $SetupTag(ID)$ and registers the new tag with the server. A reference T_i to the new tag is returned. Note that this does not reject duplicate IDs.
- `Launch()` $\rightarrow \pi, m$: this oracle launches a new protocol run, according to the protocol specification. It returns a session identifier π, generated by the reader, together with the first message m that the reader sends. Note that this implies that our model does not support tag-initiated protocols.
- `DrawTag(`T_i`,`T_j`)` $\rightarrow vtag$: on input a pair of tag references, this oracle generates a virtual tag reference, as a monotonic counter, $vtag$ and stores the triple $(vtag, T_i, T_j)$ in a table $\mathcal{D}$. Depending on the value of b, $vtag$ either refers to T_i or T_j. If one of the two tags T_i or T_j is already referenced in the table (i.e. is already passed to a `DrawTag` without being released with a `Free`), then this oracle returns $\perp$. Otherwise, it returns $vtag$.
- `Free(vtag)`$_b$: on input $vtag$, this oracle retrieves the triple $(vtag, T_i, T_j)$ from the table $\mathcal{D}$. If $b = 0$, it resets the tag T_i. Otherwise, it resets the tag T_j. Then it removes the entry $(vtag, T_i, T_j)$ from $\mathcal{D}$. When a tag is reset, its volatile memory is erased. The non-volatile memory, which contains the state S, is preserved.
- `SendTag(vtag,`m`)`$_b$ $\rightarrow m'$: on input $vtag$, this oracle retrieves the triple $(vtag, T_i, T_j)$ from the table $\mathcal{D}$ and sends the message m to either T_i (if $b = 0$) or T_j (if $b = 1$). It returns the reply from the tag (m'). If the above triple is not found in $\mathcal{D}$, it returns $\perp$.
- `SendReader(`π`,` m`)` $\rightarrow m'$: on input π, m this oracle sends the message m to the reader in session π and returns the reply m' from the reader (if any) is returned by the oracle.[4]
- `Result(`π`)`: on input π, this oracle returns a bit indicating whether or not the reader accepted session π as a protocol run that resulted in successful authentication of a tag. If the session with identifier π is not finished yet, or there exists no session with identifier π, $\perp$ is returned.

[4] If no active session π exists, the reader is likely to return $\perp$.

- `Corrupt`(T_i): on input a tag reference T_i, this oracle returns the complete internal state of T_i.[5] Note that the adversary is not given control over T_i.

According to the above experiment description, the challenger presents to the adversary the system where either the 'left' tags T_i (if $b = 0$) or the 'right' tags T_j (if $b = 1$) are selected when returning a virtual tag reference in `DrawTag`. The function $f_0 \in \mathcal{F}$ (where $\mathcal{F} = \{f : \{1, 2, \ldots, n\} \rightarrow \mathcal{T}\}$, see Section 3.5) maps the `DrawTag` invocations (referenced by an index k) to the tag T_i, which was passed as first argument to `DrawTag`. Similarly, f_1 maps invocation serial numbers to the second argument to `DrawTag`. f_0 and f_1 therefore describe the 'left' and the 'right' world, respectively.

$\mathcal{A}$ queries the oracles a number of times and, subsequently, outputs a guess bit g. We say that $\mathcal{A}$ wins the privacy game if and only if $g = b$, i.e. if it correctly identifies which of the worlds was active. The advantage of the adversary is defined as

$$\mathbf{Adv}_{\mathcal{S},\mathcal{A}}(k) = \left| Pr\left[\mathbf{Exp}^0_{\mathcal{S},\mathcal{A}}(k) = 1\right] + Pr\left[\mathbf{Exp}^1_{\mathcal{S},\mathcal{A}}(k) = 1\right] - 1 \right| \qquad (1)$$

4.2 Security, Correctness, Privacy

Since our model focuses on privacy, the correctness and security property are not discusses further. Both the Vaudenay and Deng et al. security and correctness definition can be used combined with the new privacy definition, without compatibility issues (also see Section 3.1 and Section 3.3).

The adversary restrictions, as defined in Section 3.1, also apply to our privacy definition. Depending on the acceptable usage of the `Corrupt` oracle, an adversary in our model is either Strong, Destructive (`Corrupt` destroys a tag), Forward (after the first `Corrupt` only further corruptions are allowed), or Weak (no `Corrupt` oracle) adversaries. Depending on the allowed usage of the `Result` oracle, there exist Narrow (no `Result` oracle) and Wide adversaries. X is used to denote one of these privacy notions.

Definition 8 (Privacy). *An RFID system $\mathcal{S}$, is said to unconditionally provide privacy notion X, if and only if for all adversaries $\mathcal{A}$ of type X, it holds that $\mathbf{Adv}^X_{\mathcal{S},\mathcal{A}}(k) = 0$. Similarly, we speak of computational privacy if for all polynomial time adversaries, $\mathbf{Adv}^X_{\mathcal{S},\mathcal{A}}(k) \leq \epsilon(k)$*

We also define X^+ privacy notion variants, where X refers to the basic privacy notion and $+$ to the notion that arises when the corruption abilities of the adversary are further restricted (see [5]). Formally, an RFID system is said to be X^+ private if it is X private and if, for all adversaries, $f_0 \approx_{\hat{T}} f_1$. Here, $f_0 \approx_{\hat{T}} f_1$ means that $\forall i$ such that $f_0(i) \in \hat{T}$ or $f_1(i) \in \hat{T}$, it holds that $f_0(i) = f_1(i)$,

[5] Both the volatile and non-volatile state is returned. For multi-pass protocols it might be necessary to relax this to only the non-volatile state; to force the adversary to only corrupt tags T_i that are currently not drawn; or to use the concept of X^+ privacy, as discussed in Section 4.3.

where $\hat{T}$ denotes the set of corrupted tags. This implies that, whenever a tag is corrupted at some point during the privacy game, it always has to be drawn simultaniously in both the left and the right world using a `DrawTag(`T_i, T_i`)` query with identical arguments.

4.3 Motivation and Comparison

Our proposed model is based on the well-studied notion of (left-or-right) indistinguishability. This avoids the issues with less well-studied concepts such as blinders that the Vaudenay model suffers from (see Section 3.1). Moreover, since several cryptographic schemes have proven security properties based on indistinguishability games (e.g. IND-CPA, IND-CCA, IND-CCA2...), this is likely to simplify the proofs using our model when using these schemes as building blocks.

Note that the Juels-Weis model from Section 3.4 also uses a traditional indistinguishability setup. However, the model requires the adversary to distinguish one out of two selected tags in the final phase. The disadvantage of this approach is that it does not take into account other properties that might leak privacy (e.g. cardinality) and that it limits the use of tag corruption. The Vaudenay model did introduce some crucial tools like virtual tag references and the corruption types that are still required.

Modelling details. There are certain notable differences of our model when compared to the Bohli-Pashalidis model [5] and the other models discussed in Sect. 3:

- The introduction of `CreateTag(·)`: since the set of tags is not predefined we allow the adversary to dynamically create new tags.
- `DrawTag(·, ·)` and `Free(·)` are used to introduce the concept of virtual tags. This concept is needed since otherwise `SendTag(·, ·)` would have to accept two tag/message pairs (and select one of them based on the value of b). In this case it would be trivial to determine the bit b for multi-pass protocols, simply by using different tags for each pass of the protocol if $b = 0$ and the same tag if $b = 1$. The protocol would only succeed if $b = 1$, thus allowing detection of b. Hence, it is crucial that the same tag is always used within a certain protocol run, which can be ensured by using virtual tag identifiers.
- `Free(·)` clears the volatile memory of tag, in order to avoid attacks that depend on leaving a tag hanging in a temporary state. Such an attack is described in [25].
- A separate communication oracle for tags and reader is used, since the reader is not considered as an entity whose privacy can be compromised.
- `Corrupt(·)`: corruption is done with respect to a tag, not a virtual tag. If `Corrupt(·)` would accept a vtag, then determining the bit b becomes trivial by performing the following attack:
  - $vtag_a \leftarrow$ `DrawTag(`T_1`,`T_2`)`
  - $C_a \leftarrow$ `Corrupt(`$vtag_a$`)`
  - `Free(`$vtag_a$`)`
  - $vtag_b \leftarrow$ `DrawTag(`T_1`,`T_3`)`
  - $C_b \leftarrow$ `Corrupt(`$vtag_b$`)`

If $C_a = C_b$ then $b = 0$, otherwise $b = 1$.
We believe that it is realistic to assume that one has the tag identifier T_i when corrupting a tag, since corruption implies having physical access to a tag.
Note that stateful protocols (which update their state after a protocol run) do not satisfy our privacy definition. By issuing a `Corrupt(`T_i`)` query before and after a protocol run, one can always identify whether or not the tag has been active. For such protocols, one could use the significantly weaker X^+ privacy notions.

- In the current setup `Corrupt(`T_i`)` reveals the full internal state of the tag, i.e. both its volatile and non-volatile parts. This follows [1] where it is shown that, if corruptions reveal the volatile state, then the resulting privacy notions are stronger. Single-pass protocols (e.g. challenge-response) do not suffer from any issues, since the volatile memory is typically erased after sending the reply, and hence all computations are confined to the invocation of the `SendTag` oracle. Multi-pass protocols on the contrary, typically require storage of data in between `SendTag` invocations. Because corruption yields the entire internal state, one could make additional assumptions on the corruption abilities of the adversary by restricting corruption to the non-volatile state. An even stronger restriction would be to allow only corruption of tags that are not drawn in either the left or right world; or use the X^+ privacy notions.

5 Evaluating Existing Protocols

This section evaluates several protocols (or classes of protocols) using our privacy model. For security and correctness results we refer to the original papers.

Several protocol 'prototypes' based on symmetric cryptography are evaluated by Ng et al. in [24] with respect to Vaudenay's privacy model. Since none of these protocols attain wide-forward privacy, we expect them to behave the same in our model. For this reason, these protocols are not discussed further.

5.1 Vaudenay's Public Key Protocol

Figure 2 shows the public key protocol presented by Vaudenay. The reader sends out a random number a and the tag encrypts this challenge, combined with the shared secret K and tag ID under the public key K_P of the reader. The reader can decrypt the tag's reply and verify the shared secret K in its database. The protocol relies on the encryption being IND-CPA to achieve narrow-strong Vaudenay-privacy and IND-CCA2 to achieve security and forward privacy. However, this protocol is wide-strong private under our model, if the underlying encryption is IND-CCA2.

Theorem 1. *If the encryption used in the protocol from Figure 2 is IND-CPA, then the protocol is strong private for narrow adversaries (i.e. adversaries that do not use the* `Result` *query).*

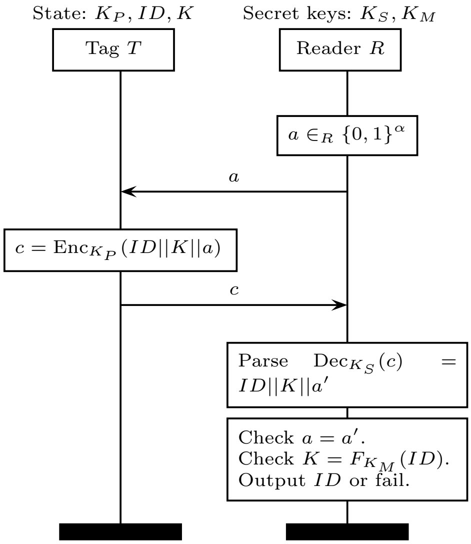


Fig. 2. Public key RFID protocol from [32]

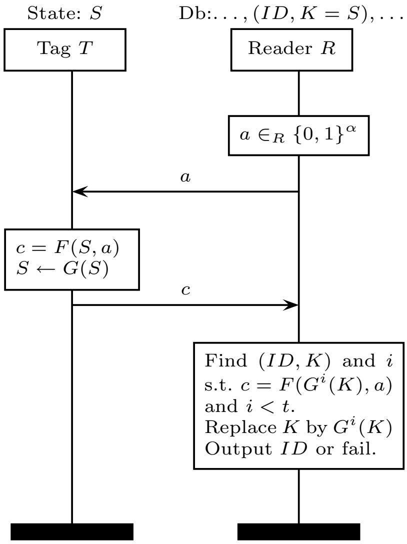


Fig. 3. RO protocol from [32]

Proof. Given an adversary $\mathcal{A}$ that wins the privacy game with non-negligible advantage, we show how to create an adversary $\mathcal{A}'$ that wins the IND-CPA game with non-negligible advantage.

The adversary $\mathcal{A}'$ runs the adversary $\mathcal{A}$ and answers all oracle queries from $\mathcal{A}$ by simply simulating the system $\mathcal{S}$, with the following exceptions:

- The public key K_P of the reader is the public key of the IND-CPA game.
- `SendTag`: retrieve the tag references T_i and T_j from the table using the virtual tag identity *vtag*. For these two tags, it generates the messages $m_0 = ID_i||K_i||a$ and $m_1 = ID_j||K_j||a$. The two messages m_0, m_1 are forwarded to the IND-CPA oracle, which returns the encryption under K_P of one of the messages.

At the end of the game $\mathcal{A}'$ outputs whatever guess $\mathcal{A}$ outputs. The privacy game is perfectly simulated for the inner adversary $\mathcal{A}$.

Assume that $\mathcal{A}$ breaks privacy, i.e. it can distinguish the left and right world, then $\mathcal{A}'$ wins the IND-CPA game. Since IND-CPA with only one call to the encryption oracle is equivalent to IND-CPA with multiple calls to the encryption oracle, this proves the (narrow) privacy of the protocol. □

The results from Lemma 8 in [32] still hold, provided the security and correctness definitions from Vaudenay are used. So, based on these results, the protocol above is also wide forward private.

Theorem 2. *If the encryption used in the protocol from Figure 2 is IND-CCA2, then the protocol is strong private for wide adversaries.*

Proof. The proof is similar to the proof for Theorem 1 above. When receiving a `Result` query, the adversary proceeds as follows. It first compares the ciphertext c to a list of outputs generated by the encryption oracle from the IND-CPA game (which are used in the `SendTag` oracle). If it matches one of these, `true` is returned. Otherwise, the result oracle forwards the ciphertext to the IND-CCA decryption oracle and receives the matching plaintext m. The plaintext is then parsed and verified, just as the reader would do. This game gives the same result as the IND-CPA game described in Theorem 1. □

5.2 RO-Based Protocol

Another (weaker) protocol from [32], shown in Figure 3, makes use of two random oracles F and G. The protocol uses an updating state S, which is shared by both tag and reader. The reader sends out a random number a and the tag computes a reply by applying F on the state S and a. The state is afterwards updated using G. Obviously, such a protocol cannot be (narrow) strong private, since the tag can trivially be traced after being corrupted.

Theorem 3. *The protocol shown in Figure 3 is narrow-destructive private.*

Proof. Assume that the challenge bit $b = 0$. We simulate the `SendTag` oracle by returning a random value c. There will never be a `SendTag` query to a corrupted tag, since tags are destroyed after corruption. This way we obtain a 'random' world that is indistinguishable from the 'left' world obtained when $b = 0$, provided the adversary makes no calls to F and G identical to the queries inside the `SendTag` oracle when $b = 0$. The probability of this happening is however negligible. By applying the same argument to the adversary execution when $b = 1$, we show that the adversary cannot distinguish between the two worlds. □

6 Conclusion

Several RFID privacy models were critically examined with respect to their assumptions, practical usability and other issues that arise when applying their privacy definition to concrete protocols. We have shown that, while some models are based on unrealistic assumptions, others are impractical to apply. We presented a new RFID privacy model, that, based on the classic notion of indistinguishability, combines the benefits of existing models while avoiding their identified drawbacks. By proving it for a concrete protocol, we show that the notion of (wide) strong privacy can be achieved under our model. Since the privacy model is based on an indistinguishability game, we can fall back on a wide range of existing proof techniques, making the model quite straightforward to use in practice.

Acknowledgements. The authors would like to thank Elena Andreeva, Junfeng Fan, Sebastian Faust, and Roel Peeters for the frequent meetings and discussions; and the anonymous reviewers for their comments and suggestions.

References

1. Armknecht, F., Sadeghi, A.-R., Scafuro, A., Visconti, I., Wachsmann, C.: Impossibility Results for RFID Privacy Notions. Transactions on Computational Science 11, 39–63 (2010)
2. Avoine, G., Dysli, E., Oechslin, P.: Reducing Time Complexity in RFID Systems. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 291–306. Springer, Heidelberg (2006)
3. Bellare, M., Fischlin, M., Goldwasser, S., Micali, S.: Identification Protocols Secure against Reset Attacks. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 495–511. Springer, Heidelberg (2001)
4. Bleichenbacher, D.: Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
5. Bohli, J.-M., Pashalidis, A.: Relations Among Privacy Notions. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 362–380. Springer, Heidelberg (2009)
6. Bringer, J., Chabanne, H., Icart, T.: Efficient zero-knowledge identification schemes which respect privacy. In: Li, W., Susilo, W., Tupakula, U.K., Safavi-Naini, R., Varadharajan, V. (eds.) ASIACCS, pp. 195–205. ACM, New York (2009)
7. Burmester, M., Le, T., Medeiros, B.: Provably secure ubiquitous systems: Universally composable RFID authentication protocols. In: Proceedings of the 2nd IEEE/CreateNet International Conference on Security and Privacy in Communication Networks (SECURECOMM). IEEE Press, Los Alamitos (2006)
8. Canard, S., Coisel, I., Etrog, J., Girault, M.: Privacy-preserving rfid systems: Model and constructions. Cryptology ePrint Archive, Report 2010/405 (2010), http://eprint.iacr.org/
9. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC, pp. 235–244 (2000)
10. Atmel Corporation. Innovative Silicon IDIC solutions (2007), http://www.atmel.com/dyn/resources/prod_documents/doc4602.pdf
11. Damgård, I., Østergaard, M.: RFID Security: Tradeoffs between Security and Efficiency. Cryptology ePrint Archive, Report 2006/234 (2006), http://eprint.iacr.org/
12. D'Arco, P., Scafuro, A., Visconti, I.: Revisiting DoS Attacks and Privacy in RFID-Enabled Networks. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 76–87. Springer, Heidelberg (2009)
13. D'Arco, P., Scafuro, A., Visconti, I.: Semi-Destructive Privacy in DoS-Enabled RFID systems. In: RFIDSec (2009)
14. Deng, R.H., Li, Y., Yung, M., Zhao, Y.: A New Framework for RFID Privacy. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 1–18. Springer, Heidelberg (2010)
15. Goyal, V., Sahai, A.: Resettably Secure Computation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 54–71. Springer, Heidelberg (2009)
16. Ha, J., Moon, S.-J., Zhou, J., Ha, J.: A New Formal Proof Model for RFID Location Privacy. In: Jajodia, S., López, J. (eds.) [19], pp. 267–281
17. Hutter, M., Schmidt, J.-M., Plos, T.: RFID and Its Vulnerability to Faults. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 363–379. Springer, Heidelberg (2008)
18. I.C.A. Organization. Machine Readable Travel Documents, Doc 9303, Part 1 Machine Readable Passports, 5th edn. (2003)

19. Nali, D., van Oorschot, P.C.: CROO: A Universal Infrastructure and Protocol to Detect Identity Fraud. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 130–145. Springer, Heidelberg (2008)
20. Juels, A., Weis, S.A.: Defining Strong Privacy for RFID. In: PerCom Workshops, pp. 342–347. IEEE Computer Society, Los Alamitos (2007)
21. Kasper, T., Oswald, D., Paar, C.: New Methods for Cost-Effective Side-Channel Attacks on Cryptographic RFIDs. In: RFIDSec (2009)
22. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer, Heidelberg (2007)
23. Ng, C.Y., Susilo, W., Mu, Y., Safavi-Naini, R.: RFID Privacy Models Revisited. In: Jajodia, S., López, J. (eds.) [19], pp. 251–266
24. Ng, C.Y., Susilo, W., Mu, Y., Safavi-Naini, R.: New Privacy Results on Synchronized RFID Authentication Protocols against Tag Tracing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 321–336. Springer, Heidelberg (2009)
25. Paise, R.-I., Vaudenay, S.: Mutual Authentication in RFID: Security and Privacy. In: ASIACCS 2008, pp. 292–299. ACM Press, New York (2008)
26. Plos, T.: Evaluation of the Detached Power Supply as Side-Channel Analysis Countermeasure for Passive UHF RFID Tags. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 444–458. Springer, Heidelberg (2009)
27. Sadeghi, A.-R., Visconti, I., Wachsmann, C.: User Privacy in Transport Systems Based on RFID E-Tickets. In: Bettini, C., Jajodia, S., Samarati, P., Wang, X.S. (eds.) PiLBA. CEUR Workshop Proceedings, vol. 397 (2008), CEUR-WS.org
28. Sadeghi, A.-R., Visconti, I., Wachsmann, C.: Anonymizer-Enabled Security and Privacy for RFID. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 134–153. Springer, Heidelberg (2009)
29. Sadeghi, A.-R., Visconti, I., Wachsmann, C.: Efficient RFID security and privacy with anonymizers. In: RFIDSec (2009)
30. NXP Semiconductors. MIFARE, http://www.mifare.net/
31. Van Le, T., Burmester, M., de Medeiros, B.: Universally composable and forward-secure RFID authentication and authenticated key exchange. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, ASIACCS 2007, pp. 242–252. ACM Press, New York (2007)
32. Vaudenay, S.: On Privacy Models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)
33. Vaudenay, S.: Invited talk at RFIDSec 2010 (2010)
34. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In: Hutter, D., Müller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing. LNCS, vol. 2802, pp. 201–212. Springer, Heidelberg (2004)

A Extending the Model

In a typical indistinguishability-based security/privacy definition, a challenger picks a random bit b and then offers a set of well-defined interfaces over which an adversary $\mathcal{A}$ can interact with the challenger. In ‘left-or-right’ security/privacy definitions, in particular, the interface specification requires that $\mathcal{A}$ provides *a pair* of identically formatted inputs to the challenger. The value of b can be interpreted as indicating in which of two possible configurations the challenger operates, namely the ‘left’ or the ‘right’ configuration, and $\mathcal{A}$’s job is to determine this configuration.

It is possible to generalise left-or-right indistinguishability such that, the challenger picks one out of 2^n possible configurations, giving us an n- indistinguishability game, with adversary $\mathcal{A}_n$. Suppose there is a system $\mathcal{S}$ that, if invoked with some parameter α (taken from a system-specific parameter space A), produces an output $\mathcal{S}(\alpha)$. The challenger chooses a positive number n, such that n is polynomial in k and generates an n-bit vector $\hat{b} = (\hat{b}_1, \ldots, \hat{b}_n)$ uniformly at random. Finally, it offers an interface over which $\mathcal{A}_n$ may query the challenger with triplets of the form $(i, \alpha_0, \alpha_1) \in \{1, \ldots, n\} \times A \times A$. On input such a triple, the challenger outputs $\mathcal{S}(\alpha_{\hat{b}_i})$.

At the end of the game, $\mathcal{A}_n$ outputs a guess $\hat{g}$ for $\hat{b}$, and we say that it wins the game if $\hat{g} = \hat{b}$. If there exists some $\mathcal{A}_n$ such that $\Pr(\mathcal{A}_n \text{wins}) > {}^1\!/\!{}_{2^n} + \epsilon$, where ϵ is any function that is non-negligible in k, then we say that $\mathcal{A}_n$ has ‘non-negligible advantage’ and that $\mathcal{S}$ is not secure.

In general, it is unclear whether or not n-indistinguishability implies 1-indistinguishability. In principle, a system could be secure if the adversary has to identify a string from a space that is exponentially large in k, but may fail security if the adversary just needs to identify a single hidden bit.

Lemma 1 (1-indistinguishability implies n-indistinguishability). *If a system $\mathcal{S}$ satisfies 1-indistinguishability then $\mathcal{S}$ also satisfies n-indistinguishability.*

Proof. We construct an 1-indistinguishability adversary $\mathcal{A}$ that uses an n- indistinguishability adversary $\mathcal{A}_n$ as a black box. $\mathcal{A}$ proceeds as follows. First, it uniformly at random chooses two n-bit vector κ and λ such that $\kappa \neq \lambda$. Then it offers the interface (i, α_0, α_1) to $\mathcal{A}_n$. For each (i, α_0, α_1) received from $\mathcal{A}_n$, $\mathcal{A}$ forwards the query $(\alpha_{\kappa_i}, \alpha_{\lambda_i})$ to the challenger, and returns the challenger’s output. By forwarding the queries this way, $\mathcal{A}$ simulates $\hat{b} = \kappa$ if $b = 0$, and $\hat{b} = \lambda$ if $b = 1$ for $\mathcal{A}_n$. In the rest of the proof $\hat{b}$ will denote the κ if $b = 0$ and λ if $b = 1$, $\bar{\hat{b}}$ will denote the κ if $b = 1$ and λ if $b = 0$. Accordingly, and given $\mathcal{A}_n$’s guess $\hat{g}$, $\mathcal{A}$ outputs the guess $b = 0$ if $\hat{g} = \kappa$, $b = 1$ if $\hat{g} = \lambda$, or simply a uniformly at random selected bit otherwise.

Consider the $2^n \times 2^n$ matrix P with elements $p_{i,j} = \Pr(\mathcal{A}_n \text{ outputs } j \mid \hat{b} = i)$. That is, P contains the probabilities that $\mathcal{A}_n$ outputs any possible value $\hat{g}$, conditional on the value of $\hat{b}$; the element at row number i and column number j is the probability that $\mathcal{A}_n$ outputs $\hat{g} = j$ (encoded as a bit vector), given the challenge bit vector has the value $\hat{b} = i$ (encoded as a bit vector). Note that, for all $0 \leq i \leq 2^n$, $\sum_j p_{i,j} = 1$.

For any given choice of a pair (κ, λ), the probability that $\mathcal{A}_n$ wins (i.e. that it outputs $\hat{g} = \hat{b}$) is ${}^1\!/\!{}_2(p_{\kappa,\kappa} + p_{\lambda,\lambda})$. Similarly, the probability that it ouputs $\hat{g} = \bar{\hat{b}}$ is ${}^1\!/\!{}_2(p_{\kappa,\lambda} + p_{\lambda,\kappa})$. Averaging over all possible choices of (κ, λ) we obtain

$$\Pr(\mathcal{A}_n \text{ wins}) = \frac{1}{2^n(2^n-1)} \sum_{\substack{\kappa,\lambda \in \{0,1\}^n \\ \kappa \neq \lambda}} \frac{1}{2}(p_{\kappa,\kappa} + p_{\lambda,\lambda}) = \frac{\mathcal{D}}{2^n} \tag{2}$$

$$\Pr(\mathrm{err}) = \frac{1}{2^n(2^n-1)} \sum_{\substack{\kappa,\lambda\in\{0,1\}^n\\ \kappa\neq\lambda}} \frac{1}{2}(p_{\kappa,\lambda}+p_{\lambda,\kappa}) = \frac{2^n-\mathcal{D}}{2^n(2^n-1)}, \tag{3}$$

where $\mathcal{D} = \sum_{i=1}^{2^n} p_{i,i}$ is the trace of P. By construction of our $\mathcal{A}$, we have

$$\Pr(\mathcal{A}\text{ wins}) = \Pr(\mathcal{A}_n\text{ wins}) + {}^1\!/\!{}_2(1-\Pr(\mathcal{A}_n\text{ wins}) - \Pr(\mathrm{err})) \tag{4}$$

and substituting Equations 2 and 3 into Equation 4, we obtain

$$\Pr(\mathcal{A}\text{ wins}) = \frac{1}{2} + \frac{2^n(\mathcal{D}-1)}{2^{n+1}(2^n-1)}. \tag{5}$$

By assumption we have that $\Pr(\mathcal{A}_n\text{ wins}) > {}^1\!/\!{}_{2^n} + \epsilon$ for all functions ϵ that are negligible in k. Hence, $\Pr(\mathcal{A}_n\text{ wins}) = {}^1\!/\!{}_{2^n} + \delta$ for some non-negligible positive $\delta \leq 1 - {}^1\!/\!{}_{2^n}$. In terms of the elements in P, we have $\mathcal{D} = 1 + 2^n\delta$ and when substituting this into Equation 5 we obtain $\Pr(\mathcal{A}\text{ wins}) = \frac{1}{2} + \frac{2^n\delta}{2(2^n-1)} > {}^1\!/\!{}_2 + {}^\delta\!/\!{}_2$. Hence, $\mathcal{A}$'s advantage is non-negligible. □

Unlike standard hybrid arguments, the advantage δ is at most divided by 2, when going from an n-bit distinguisher to a 1-bit distinguisher.

B Mutual Authentication

Since our model is not based anymore on the blinder construction of Paise-Vaudenay [25], none of the impossibility results of [1] apply. It is straightforward to modify the proof from Section 5.1 to the mutual authentication protocol based on IND-CCA encryption from Section 6.3 in [25].

Quantitative Information Flow, with a View*

Michele Boreale[1], Francesca Pampaloni[2], and Michela Paolini[2]

[1] Università di Firenze, Italy
[2] IMT - Institute for Advanced Studies, Lucca, Italy

Abstract. We put forward a general model intended for assessment of system security against passive eavesdroppers, both quantitatively (*how much* information is leaked) and qualitatively (*what* properties are leaked). To this purpose, we extend information hiding systems (IHS), a model where the secret-observable relation is represented as a noisy channel, with *views*: basically, partitions of the state-space. Given a view W and n independent observations of the system, one is interested in the probability that a Bayesian adversary wrongly predicts the class of W the underlying secret belongs to. We offer results that allow one to easily characterise the behaviour of this error probability as a function of the number of observations, in terms of the channel matrices defining the IHS and the view W. In particular, we provide expressions for the limit value as $n \to \infty$, show by tight bounds that convergence is exponential, and also characterise the rate of convergence to predefined error thresholds. We then show a few instances of statistical attacks that can be assessed by a direct application of our model: attacks against modular exponentiation that exploit timing leaks, against anonymity in mix-nets and against privacy in sparse datasets.

Keywords: quantitative information flow, statistical attacks, anonymity, privacy, information theory.

1 Introduction

Statistical attacks against secrecy, anonymity, privacy and other confidentiality properties in systems that handle sensitive data abound in the literature. In these attacks, the adversary gets to know a sample of observations of a target system – such as timing or power consumption traces of a smart-card [14], attribute values in a dataset [19], etc. – and, exploiting some form of correlation existing between the secret and the observables, tries to infer the secret – the private key, the identity of an individual, etc. Many of these attacks seem to exploit very specific features of the target system. This fact makes assessing the security of a system against this form of threat a difficult task in general. A major motivation of the present paper is to put forward a general Bayesian model where this kind of assessment can be conducted rigorously. One of our objectives is to characterise the *information leakage* of a system, both quantitatively and qualitatively, as the number of observations of the attacker increases.

* Work partially supported by the EU funded project ASCENS. Corresponding author: Michele Boreale, Università di Firenze, Dipartimento di Sistemi e Informatica, Viale Morgagni 65, I-50134 Firenze, Italy. E-mail: boreale@dsi.unifi.it

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 588–606, 2011.

It has been recently argued [8] that, for the purpose of quantifying the amount of sensitive information that is leaked by a system, it is useful to model the system itself as a *channel* in the sense of Information Theory: inputs to the channel represent the secret information, outputs represent the observable information, and the two sets are related by a conditional probability matrix. We collectively designate systems amenable to this kind of analysis as *information hiding systems (*IHS*)*. Initial works on IHS's concentrated on Shannon entropy and capacity as measures of information leakage [8,9]. More recently, it has been argued [21] that min-entropy based metrics, taking into account the success probability of an optimal attacker, provide a more operational and sensible formalization of leakage. Analysis of IHS's in the case of min-entropy and repeated independent observations, which encompasses several forms of statistical attacks, has been carried out in our previous paper [5].

A drawback of the IHS approach so far is that it focuses exclusively on the quantitative aspect of the analysis (*how much* is leaked), while ignoring the qualitative aspect (*what* is leaked) at all. In [5] it is shown that, when a uniform distribution on the secrets is assumed, the asymptotic information leakage of a system corresponds to the log of the number of indistinguishability classes in the system – where two states are indistinguishable if they induce the same probability distribution on the observables. For instance, an anonymity protocol in which users are grouped into a small number of classes is considered as globally secure. However, it might well be the case that, while the vast majority of users belong indeed to large classes, few individual users belong to a singleton classes, hence being totally exposed to eavesdropping. To make another, extreme example, consider the two small imperative procedures `P1` and `P2` below. Both of them receive as an argument a confidential variable `h` that can take on a value in the set $\mathcal{S} = \{0, ..., 15\}$, possibly corresponding to user identifiers or other sensitive information. Part of the information about `h` is disclosed by the procedures through the public variable `l`.

```
P1(h): l=-1; if (h==0) then l=0;        P2(h): l=h mod 4;
```

In the case of `P1`, there are two possible observables, -1 and 0, hence $\mathcal{S}$ is partitioned into two indistinguishability classes: thus, assuming `h` is uniformly distributed, `P1` leaks 1 bit of information about `h`. In the case of `P2` there are four classes, hence `P2` leaks two bits. From a global point of view, `P1` is therefore more secure than `P2`. Needless to say, though, *from the point of view of user 0*, `P2` is preferable over `P1`. One would like to conduct the analysis both at a quantitative and at a qualitative level, revealing not only how much is leaked, but also what. This is particularly relevant in relation to the privacy of individuals or groups.

In this paper, we propose a framework to deal with this issue by extending the IHS's considered in [5] and elsewhere with *views*. A view is, in short, a partition of the states, representing perhaps a subdivision in "buckets" of a large population (in fact, we are more general and also admit probabilistic partitions). In the example above, the view of interest to user 0 is the partition of $\mathcal{S}$ into $(\{0\}, \mathcal{S} \setminus \{0\})$. Given a view W, one is interested in the adversary's probability of wrongly predicting the class of W the secret belongs to, after observing n independent executions of the system, throughout which the secret state is kept fixed: call this quantity $P_e^W(n)$. In the example above, the involved systems are deterministic, hence a single observation is all the attacker needs. One easily finds that $P_e^W(1)$ equals 0 in the case of P1, and $\frac{1}{16}$ in the case of P2. In the general case of probabilistic systems, computation of the limit value of $P_e^W(n)$ is not as obvious. Nevertheless, we offer results that allow one to easily characterise the behaviour of $P_e^W(n)$ from the channel matrices defining the IHS and the view W. In particular, we show how to determine the limit value of $P_e^W(n)$ and its *rate*. In fact, the security of a system (w.r.t. W) depends not only on the limit in question, but also on the shape of $P_e^W(n)$ as a function of n. We show that the convergence is exponential, and provide bounds for the rate of convergence. More generally, we give bounds on the rate at which a chosen probability threshold can be reached[1].

We then give a few examples of statistical attacks that can be assessed as a direct application of our results: timing attacks against exponentiation with blinding [14,17], attacks against anonymity in mix-nets [13] and attacks against privacy in sparse datasets [19]. In the last case, we show that the condition of (ϵ, δ)-sparsity directly translates into a rate $-\log \epsilon$ for the threshold δ in our framework. In all cases, we highlight the role played by views.

In summary, we offer a unifying model for assessing a variety of statistical attacks, both at the global level and at the level of specific partitions of the secrets. We believe that this model can gain us a qualitative insight about the security of IHS's. Whenever the system is found to be insecure, an attack can be explicitly described, usually with little effort, as an instantiation of the Bayesian attacker underlying our framework. We are *not* claiming, of course, that our bounds always match the performance of existing attacks, tailored against specific, real-world systems.

The rest of the paper is organized as follows. In Section 2 some terminology and notation are introduced. Section 3 introduces the formal set up. Section 4 discusses the main results on asymptotic error probability. Section 5 presents an application to mix-nets, while Section 6 discusses sparse datasets. Some concluding remarks and discussion of related work are found in Section 7. Some technical material has been confined to a separate Appendix.

2 Notations and Preliminary Notions

Let $\mathcal{A}$ be a finite nonempty set. A probability distribution on a $\mathcal{A}$ is a function $p : \mathcal{A} \to [0,1]$ such that $\sum_{a\in\mathcal{A}} p(a) = 1$. We let $\mathrm{supp}(p)$ denote $\{a \in \mathcal{A} : p(a) > 0\}$. For any $A \subseteq \mathcal{A}$ we let $p(A)$ denote $\sum_{a\in A} p(a)$. Given $n \geq 0$, we let $p^n : \mathcal{A}^n \to [0,1]$ be the n-th extension of p, defined as $p^n(a_1,\ldots,a_n) \stackrel{\triangle}{=} \Pi_{i=1}^n p(a_i)$; this is in turn a probability distribution on $\mathcal{A}^n$. For $n = 0$, we set $p^0(\epsilon) = 1$, where ϵ denotes here the empty string. Given $A \subseteq \mathcal{A}^n$, we will often write $p^n(A)$ as just $p(A)$, if n is clear from the context.

Given two distributions p and q on $\mathcal{A}$, the *Kullback-Leibler (KL) divergence* of p and q is defined as (all the log's are taken with base 2)

$$D(p\|q) \stackrel{\triangle}{=} \sum_{a\in\mathcal{A}} p(a) \cdot \log \frac{p(a)}{q(a)}$$

[1] Indeed, it may well be the case that, even if the asymptotic rate of convergence to the limit value is extremely slow, convergence to the chosen threshold is very fast, leading to consider the system insecure.

with the proviso that $0 \cdot \log \frac{0}{q(a)} = 0$ and that $p(a) \cdot \log \frac{p(a)}{0} = +\infty$ if $p(a) > 0$. It can be shown that $D(p\|q) \geq 0$, with equality if and only if $p = q$ (*Gibbs inequality*). KL-divergence can be thought of as a sort of distance between p and q, although strictly speaking it is not – it is not symmetric, nor satisfies the triangle inequality.

$\Pr(\cdot)$ will generally denote a probability measure. Given a random variable X taking values in $\mathcal{A}$, we write $X \sim p$ if X is distributed according to p, that is for each $a \in \mathcal{A}$, $\Pr(X = a) = p(a)$.

3 Formal Set Up

3.1 Basic Definitions

We recall from [5] that an *information hiding system* (IHS for short) is a quadruple $\mathcal{H} = (\mathcal{S}, \mathcal{O}, p(\cdot), p(\cdot|\cdot))$, composed by a finite set of *states* $\mathcal{S} = \{s_1, ..., s_m\}$ representing the secret information, a finite set of *observables* $\mathcal{O} = \{o_1, ..., o_l\}$, an a priori probability distribution on $\mathcal{S}$, $p(\cdot)$, and a *conditional probability matrix*, $p(\cdot|\cdot) \in [0,1]^{\mathcal{S}\times\mathcal{O}}$, where each row sums up to 1. The entry of row s and column o of this matrix will be written as $p(o|s)$, and represents the probability of observing o given that s is the (secret) input of the system. For each s, the s-th row of the matrix is identified with the probability distribution $o \mapsto p(o|s)$ on $\mathcal{O}$, denoted by $p(\cdot|s)$.

Definition 1 (views). *Let $\mathcal{H} = (\mathcal{S}, \mathcal{O}, p(\cdot), p(\cdot|\cdot))$ be a* IHS. *A* view *of $\mathcal{H}$ is a pair $(\mathcal{W}, q(\cdot|\cdot))$, where $\mathcal{W}$ is a finite alphabet and $q(\cdot|\cdot) \in [0,1]^{\mathcal{S}\times\mathcal{W}}$ is a matrix where all rows sum to 1.*

Informally, $q(w|s)$ is the probability that the property w holds when in state s. The probability distribution p on $\mathcal{S}$ and the conditional probability matrices $p(o|s)$ and $q(w|s)$ induce a probability distribution r on $\mathcal{W}\times\mathcal{S}\times\mathcal{O}$, defined as $r(w,s,o) \overset{\triangle}{=} p(s)\cdot p(o|s)\cdot q(w|s)$. This distribution induce a triple of discrete random variables $(W, S, O) \sim r$, taking values in $\mathcal{W} \times \mathcal{S} \times \mathcal{O}$. We shall denote the marginal probability distributions of this triple for S, W and O by p_S, p_W and p_O, respectively. Of course, $p_S(\cdot)$ coincides with the prior $p(\cdot)$ given in the IHS, while the marginal distributions p_W and p_O can be computed from the given data, $p(\cdot)$, $p(\cdot|\cdot)$ and $q(\cdot|\cdot)$.

Let us now discuss the observation scenario. Given any $n \geq 0$, we assume the adversary is a passive eavesdropper that gets to know the observations corresponding to n independent executions of the system, $o^n = (o_1, ..., o_n) \in \mathcal{O}^n$, throughout which both the secret state s and the corresponding view w are kept fixed. Formally, the adversary knows a random vector of observations $O^n = (O_1, ..., O_n)$ such that, for each $i = 1, ..., n$, O_i is distributed like O. Moreover, the individual O_i and the view W are *conditionally independent* given S. This means that the following equality holds true for each $o^n \in \mathcal{O}^n$, $w \in \mathcal{W}$ and $s \in \mathcal{S}$ s.t. $p(s) > 0$

$$\Pr\left(O^n = (o_1, \ldots, o_n),\ W = w \,|\, S = s\right) = \Pi_{i=1}^n \Pr(O_i = o_i|S = s) \Pr(W = w|S = s)\,.$$

Note that the right-hand side of the above equality can be equivalently written as $\Pi_{i=1}^n p(o_i|s)q(w|s)$. Concerning the goals of the attacker, there are two cases, which we examine in the following subsections.

Notation: We shall drop the subscripts from the above defined (conditional) probability distributions when no ambiguity can arise. We will often abbreviate $\Pi_{i=1}^n p(o_i|s)$

as $p(o^n|s)$. Moreover, by slightly abusing notation, we will freely identify a view $(\mathcal{W}, q(\cdot|\cdot))$ of $\mathcal{H}$ with the induced random variable W.

3.2 Attacker Targets S

We first discuss the case when the attacker targets the states, like in [5]. In this case, his strategy, for any fixed length n of observations, is modeled by a *guessing function* $g : \mathcal{O}^n \to \mathcal{S}$, which represents the single guess the attacker is allowed to make about the secret state s, after observing o^n. In this case, one is interested in the *probability of error after n observations* (relative to g), given by

$$P_e^{(g)}(n) \overset{\triangle}{=} \Pr(g(O^n) \neq S)\,.$$

It is well-known (see e.g. [12]) that the optimal strategy for the adversary, that is the one that minimizes the error probability, is the Maximum A Posteriori (MAP) rule. A function $g : \mathcal{O}^n \to \mathcal{S}$ satisfies the *Maximum A Posteriori (*MAP*) criterion*[2] if for each $o^n \in \mathcal{O}^n$ and $s \in \mathcal{S}$

$$g(o^n) = s \text{ implies } p(o^n|s)p(s) \geq p(o^n|s')p(s') \text{ for each } s' \in \mathcal{S}\,.$$

In the above definition, for the case $n = 0$ it is convenient to stipulate that $p(\epsilon|s) = 1$: that is, with no observations at all, it is selected some s maximizing the prior distribution. With this choice, $P_e^{(g)}(0)$ denotes $1 - \max_s p(s)$. Once n and $p(s)$ are fixed, $P_e^{(g)}(n)$ does *not* depend on the specific MAP function g that is chosen. Unless otherwise stated, throughout the paper we assume the underlying guessing function is MAP and shall normally omit the superscript $^{(g)}$.

In [5], it is proven that $P_e(n)$ converges exponentially fast to a quantity that depends on an *indistinguishability* relation on states. This relation is defined as follows: $s \equiv s'$ if $p(\cdot|s) = p(\cdot|s')$. Concretely, two states are indistinguishable if the corresponding rows in the conditional probability matrix $p(\cdot|\cdot)$ are equal. This intuitively says that there is no way for the adversary to tell s and s' apart, no matter how many observations he performs. Let us stress that this definition does not depend on the prior distribution on states, nor on the number n of observations. Assume $\equiv$ partitions $\mathcal{S}$ into K equivalence classes $C_1, ..., C_K$. For each i, let $s_i^* \in C_i$ be a state that $p_S(s_i^*) = \max_{s \in C_i} p_S(s)$. Let

$$\pi_i \overset{\triangle}{=} p_S(s_i^*) \quad \text{and} \quad p_i(\cdot) \overset{\triangle}{=} p(\cdot|s_i^*)\,. \tag{1}$$

We can assume w.l.o.g. that $\pi_i > 0$ for each i. In [5], it is shown that as $n \to \infty$, then exponentially fast

$$P_e(n) \to 1 - \sum_{i=1}^{K} \pi_i\,. \tag{2}$$

Note that the case $|\mathcal{S}| = 2$ with a nontrivial indistinguishability corresponds to the Bayesian version of the classical binary Hypothesis Testing; in this case, the Chernoff information is known to be the optimal exponent (see [12, Ch.11] and Section 4).

[2] Another widely used criterion for guessing functions is *Maximum Likelihood (*ML*)*, which requires no knowledge of the prior distribution. Our main results can be extended to the ML rule, although we will not discuss this issue in the present paper. See [5, Remark 2].

3.3 Attacker Targets *W*

We discuss now the case when the attacker targets a property of states represented by a view W. Similarly to the previous case, the attacker's strategy corresponds to a guessing function, which this time is of the form $g : \mathcal{O}^n \rightarrow \mathcal{W}$. The corresponding error probability (after n observations, relative to g) is

$$P_e^{g,W}(n) \triangleq \Pr\left(g(O^n) \neq W \right) . \qquad (3)$$

A function g minimizes this quantity if it is W-MAP, that is if satisfies the following condition. For each $o^n \in \mathcal{O}^n$ and $w \in \mathcal{W}$

$$g(o^n) = w \text{ implies } p(o^n|w)p(w) \geq p(o^n|w')p(w'), \text{ for each } w' \in \mathcal{W}.$$

Unless otherwise stated, given a view of $\mathcal{H}$, we shall assume an underlying guessing function that is W-map. Consequently, we shall normally omit the indication of g from $P_e^{g,W}(n)$.

In many systems, the practically important views are those that partition the state-space into equivalence classes. A view W is called a *partition* of $\mathcal{H}$ if W is a function of S, that is $W = f(S)$ for some function $f : \mathcal{S} \rightarrow \mathcal{W}$. Equivalently, the matrix $q(\cdot|\cdot)$ has a single entry '1' for each row. Let $\mathcal{W} = \{w_1, ..., w_L\}$, and let $E_i \triangleq f^{-1}(w_i)$ for $1 \leq i \leq L$. Of course $E_1, ..., E_L$ forms a partition of $\mathcal{S}$, in the set-theoretic sense.

3.4 Information Leakage

Information leakage aims at measuring, typically in bits, the information leaked by a system, by comparing the prior to the posterior (to the observations) adversary's *success* probability. Below, we follow Smith [21] and define information leakage as the difference between the min-entropies of the prior and posterior probability distributions. In what follows, we pose $P_{succ}(n) \triangleq 1 - P_e(n)$; similarly for P^W_{succ}. The intuition underlying this definition is that gaining 1 bit of information corresponds to doubling the success probability.

Definition 2 (Information leakage [21]). *The* information leakage *of $\mathcal{H}$ after n observations is defined as*

$$\mathcal{L}(n) \ \triangleq\ \log\left(\frac{P_{succ}(n)}{\max_s p_S(s)} \right).$$

Similarly, information leakage after n observations relative to a view *W is defined as* $\mathcal{L}^W(n) \triangleq \log(\frac{P^W_{succ}(n)}{\max_w p_W(w)})$.

4 Asymptotic Error Probability

Throughout the section $\mathcal{H}$ denotes a generic IHS $(\mathcal{S}, \mathcal{O}, p(\cdot), p(\cdot|\cdot))$. We begin with a few preliminary definitions concerning the rate of convergence. Then prove a result giving strong bounds for $P_e(n)$ and its rate of convergence, Theorem 1. This result greatly improves on the bounds in [5] and is the key to the results for $P^W_e(n)$.

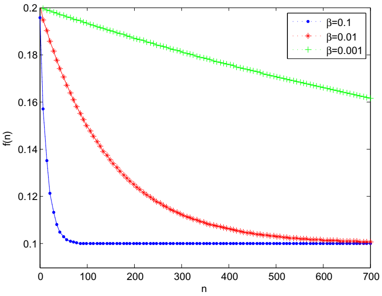
Definition 3 (rate). *Let $f : \mathbb{N} \to \mathbb{R}^+$ be a nonnegative, monotonically non-increasing function. Let $\gamma = \lim_{n\to\infty} f(n)$. The* rate *of f is defined as the nonnegative quantity*

$$\rho(f) \overset{\triangle}{=} -\lim_{n\to\infty} \frac{1}{n} \log(f(n) - \gamma) . \tag{4}$$

We further say that f reaches δ at rate ϵ *if there is a nonnegative, monotonically non-increasing function h s.t.* $\lim_{n\to\infty} h(n) \le \delta$, $\rho(h) \ge \epsilon$ *and $f(n) \le h(n)$ for each n large enough.*

Note that we admit rates of 0, as well as of $+\infty$.

Example 1. Consider $f(n) = \alpha + \beta 2^{-n\lambda_1} + \gamma 2^{-n\lambda_2}$, for some nonnegative α, β and γ, and $0 < \lambda_1 < \lambda_2$. Then $f(n) \to \alpha$ and $\rho(f) = \lambda_1$. On the other hand, since $f(n) \le h(n) = \alpha + \beta + \gamma 2^{-n\lambda_2}$, one has that f reaches $\alpha + \beta$ at a rate of λ_2. The picture on the right displays a plot of three functions, characterised by identical values of $\alpha = 0.1$, $\gamma = 0.01$, $\lambda_1 = 0.01$, and $\lambda_2 = 2$, and by three different values of β: $\beta = 0.1$ (top curve), 0.01 (middle curve) and 0.001 (bottom curve).



One can see that although the convergence to the limit value, 0.1 for all of them, is extremely slow, convergence to the value 0.11, which is only slightly higher, in the third case is very fast. A system with an error probability function of this shape would not be considered as secure.

Recall from [12] that given two probability distributions p and q on O, the *Chernoff Information* between p and q is the nonnegative quantity

$$C(p,q) \overset{\triangle}{=} -\min_{0\le\lambda\le1} \log \sum_{o\in \mathrm{supp}(p)\cap\mathrm{supp}(q)} p^{\lambda}(o) q^{1-\lambda}(o) \tag{5}$$

with the convention that $C(p,q) = +\infty$ if $\mathrm{supp}(p) \cap \mathrm{supp}(q) = \emptyset$. Recall that, in our notation, $p_1(\cdot), ..., p_K(\cdot)$ are the representative probability distributions of $\mathcal{H}$, defined in (1). By adapting the proof for the case $|\mathcal{S}| = 2$ that is given in [12] (see also [18]), it is not difficult to prove the following result, which gives the exact rate of convergence for $P_e(n)$, in the case where the distributions $p_1(\cdot), ..., p_K(\cdot)$ *all have the same support*[3].

Proposition 1. *Suppose that* $\mathrm{supp}(p_1) = \cdots = \mathrm{supp}(p_K)$. *Then* $\rho(P_e) = \min_{i\neq j} C(p_i, p_j)$.

The next result provides tight bounds on the error probability $P_e(n)$ and its rate in the general case, although in general not the *exact* rate. More generally, the result below provides a means to tradeoff bounds on error probability with bounds on the rate of convergence. We make use of the following notations. For all $i, j = 1, ..., K$, define

$$c_{ij} \overset{\triangle}{=} C(p_i, p_j) .$$

[3] In the case where the distributions have different supports, the argument of [12] does not apply. The ultimate reason is that that $D(p||q)$ is not continuous in the first argument if q has not full support; see also [2] for a discussion on this issue.

We also stipulate that $2^{-\infty} = 0$. The next theorem has the following interpretation. The attacker focuses on a subset of the representative states, $\{s^*_i | i \in I\}$, and tries to identify one of them as S. This strategy can fail for two reasons: either S is not in the target subset (first term in the error expression), or it is, but the attacker mistakes one state in the subset for another (second term in the error expression). The latter probability decreases exponentially fast with n, at a rate that is at least as big as the minimum "distance" ρ_I between the distributions $p_i(\cdot)$, for $i \in I$. The proof can be found in the Appendix.

Theorem 1. *Let I be a nonempty subset of $\{1, ..., K\}$. Let $\rho_I \overset{\triangle}{=} \min_{i,j \in I, i \neq j} c_{ij}$. Let $\pi_{\max} = \max_{i \in I} \pi_i$. Then, for all $n \geq 1$*

$$P_e(n) \ \leq \ (1 - \sum_{i \in I} \pi_i) \ + \ \frac{|I|^2}{2} \pi_{\max} 2^{-n\rho_I} \, . \tag{6}$$

As a consequence, $P_e(n)$ reaches $(1 - \sum_{i \in I} \pi_i)$ at a rate of ρ_I. In particular, by taking $I = \{1, ..., K\}$, we obtain that $\rho(P_e) \geq \rho_I$.

Remark 1. (a) In the practically important case where the prior p_S on $\mathcal{S}$ is uniform, the term $\frac{|I|^2}{2} \pi_{\max} 2^{-n\rho_I}$ is bounded above by $\frac{K}{2} 2^{-n\rho_I}$.

(b) Computation of the Chernoff Information (5) is an optimization problem that may be difficult to solve exactly. In practice, setting $\lambda = \frac{1}{2}$ in the argument of the min often yields a good lower bound of $C(p, q)$, known as *Bhattacharyya distance*. Another lower bound that we will find useful in the case of distributions with sparse support (see Section 6), is obtained by taking the min limited to the cases $\lambda = 0$ and $\lambda = 1$. Letting $\sigma = \mathrm{supp}(p) \cap \mathrm{supp}(q)$, this quantity amounts to $- \min\{\log p(\sigma) \, , \, \log q(\sigma)\}$.

We analyse now the case of P^W_e, where W is a generic view of an IHS $\mathcal{H}$. We follow the notation and terminology established in the previous section. It would be tempting to proceed as follows: build a new IHS, say $\mathcal{H}^W$, where the states are $\mathcal{W}$ and the channel matrix is $p_{O|W}$. The error probability function for $\mathcal{H}^W$ would then coincide with $P^W_e(n)$. It would then be enough to apply Theorem 1 to $\mathcal{H}^W$. This approach however is doomed to failure. In fact, the assumption that the observations O_i are conditionally independent given W is in general false:

$$p(o_1 \cdots o_n | w) \neq p(o_1 | w) \cdots p(o_n | w) \, .$$

As a consequence, the IHS $\mathcal{H}^W$ is meaningless for what concerns our purposes. However, conditional independence of the O_i's given W is guaranteed, and the approach outlined above *does* work, in the special case where W is a partition finer than $\equiv$. This intuition leads us to develop to the method illustrated below for P^W_e in the general case.

Some more notation first. For notational simplicity, assume $\mathcal{W}$ is a set of integers $\{1, ..., |\mathcal{W}|\}$. Let $q(\cdot|\cdot)$ be the matrix defining the view W. We denote by $\sim_W$ the equivalence relation on $\mathcal{S}$ induced by $q(\cdot|\cdot)$, that is

$$s \sim_W s' \text{ iff } \text{ for each } o \in \mathcal{O} : \ q(o|s) = q(o|s') \, . \tag{7}$$

In other words, two states are $\sim_W$-equivalent if the corresponding rows of $q(\cdot|\cdot)$ are equal. Let $\mathcal{S}/{\sim_W}$ be $\{E_1, ..., E_L\}$, the equivalence classes of $\sim_W$. The intersection $\equiv \cap \sim_W$

is still an equivalence relation on $\mathcal{S}$, that is finer than both $\equiv$ and $\sim_W$. Recall that $\mathcal{S}/\equiv$ is $\{C_1, ..., C_K\}$. For $1 \leq i \leq K$ and $1 \leq j \leq L$, we let the equivalence classes of $\equiv \cap \sim_W$ be denoted as

$$F_{ij} \triangleq C_i \cap E_j \tag{8}$$

and furthermore

$$F_i^* \triangleq \max_j p_S(F_{ij}) \quad \text{and} \quad q_j^* \triangleq \max_w q(w|s), \text{ for an arbitrary } s \in E_j\,. \tag{9}$$

The next theorem has the following interpretation. The attacker focuses on a subset of the representative states, $\{s_i^* | i \in I\}$. He tries to identify first the class C_i of S, then guesses the class F_{ij} – this is given by the j that maximizes $p_S(F_{ij})$. Finally he guesses the view w that is most likely in E_j. This strategy can fail for two reasons: either w is wrong (first term in the expression), or F_{ij} is wrong (second + third term). We report a proof of this result in the Appendix.

Theorem 2. *Let I and ρ_I be chosen as in Theorem 1. Let W be a view of $\mathcal{H}$. Let $\Pi_{\max} = \max_{i \in I} F_i^*$. Then*

$$P_e^W(n) \;\leq\; \sum_{j=1}^{L}(1 - q_j^*) \;+\; (1 - \sum_{i \in I} F_i^*\,) \;+\; \frac{|I|^2}{2}\Pi_{\max}2^{-n\rho_I}\,. \tag{10}$$

Note that the determination of the upper-bound in (10) is computationally practical: the partitions induced by $\equiv \cap \sim_W$ can be directly computed by inspection of the matrices $p(\cdot|\cdot)$ and $q(\cdot|\cdot)$. Their intersection (8), and the probability mass of the corresponding classes $p_S(F_{ij})$, are then straightforward to compute. Theorem 2 only provides an (exponential) upper bound to $P_e^W(n)$. The following theorem provides the exact limit of $P_e^W(n)$ in the special, but important case when W is a partition.

We introduce quickly a few concepts of the *method of types* from Information Theory [12, Ch11] that will be used in the proof. Fix $n \geq 1$. Given a sequence $o^n \in \mathcal{O}^n$ and $o \in \mathcal{O}$, denote by $\mathrm{n}(o, o^n)$ the number of occurrences of o inside o^n. The empirical distribution or *type* of o^n is the distribution on $\mathcal{O}$ defined as $t_{o^n}(o) \triangleq \mathrm{n}(o, o^n)/n$, for each $o \in \mathcal{O}$. The "balls" of center $p_i(\cdot)$ and radius $\epsilon > 0$ in $\mathcal{O}^n$ are defined as $U_i^n(\epsilon) \triangleq \{o^n : D(t_{o^n} \| p_i) \leq \epsilon\}$. It is a result from the method of types that, as $n \to +\infty$, $p_i(U_i^n(\epsilon)) \to 1$, while, for any $p \neq p_i$ there is $\epsilon > 0$ small enough s.t. $p(U_i^n(\epsilon)) \to 0$. Moreover, the convergence is exponential in both cases.

Theorem 3. *Let W be a partition of $\mathcal{H}$. Then $P_e^W(n)$ converges exponentially fast to $1 - \sum_{i=1}^K F_i^*$. More precisely, with the same notation of Theorem 2, for each $n \geq 1$, $1 - \sum_{i=1}^K F_i^* \leq P_e^W(n) \leq (1 - \sum_{i=1}^K F_i^*) + \frac{K^2}{2}\Pi_{\max}2^{-n\rho_I}$, where $I = \{1, ..., K\}$.*

Proof. (Outline) First, note that for W a partition, the first term in (10) vanishes, as each q_j^* equals 1. The upper bound is then a consequence of Theorem 2 with $I = \{1, ..., K\}$. We now seek for a lower bound of $P_e^W(n)$. We equivalently focus on an upper bound of $P_{succ}^W(n)$. Assume without loss of generality that $\mathcal{W} = \{1, ..., L\}$. For any $n \geq 1$, let

$g : \mathcal{O}^n \to \{1, ..., L\}$ be a W-MAP guessing function, and let $A_j = g^{-1}(j)$, for $j \in \{1, ..., L\}$, be the acceptance region in $\mathcal{O}^n$ for j. It is a routine task to check that

$$P^W_{succ}(n) = \sum_{i=1}^{K} \sum_{j=1}^{L} p_i(A_j) p_S(F_{ij}) \,. \tag{11}$$

Now, fix any $i \in \{1, ..., K\}$, and let $j_i = \text{argmax}_{j=1,...,L} p_S(F_{ij})$, that is $p_S(F_{ij_i}) = F^*_i$. We claim that $p_i(A_{j_i}) \to 1$ as $n \to +\infty$. In fact, fixed $\epsilon > 0$ small enough, for any n large enough A_{j_i} contains the "ball" $U^n_i(\epsilon)$ of center $p_i(\cdot)$ and radius ϵ in $\mathcal{O}^n$. To see that this is true, note that a sufficient condition for $o^n \in A_{j_i}$ is that for each $j \neq j_i$

$$p_{O^n|W}(o^n|j_i)p_W(j_i) = \sum_{l=1}^{K} p_l(o^n) p_S(F_{lj_i}) \;>\; \sum_{l=1}^{K} p_l(o^n) p_S(F_{lj}) = p_{O^n|W}(o^n|j)p_W(j) \,. \tag{12}$$

Now from results of the method of types it follows that, for $o^n \in U^n_i(\epsilon)$, we have that all the $p_l(o^n)$ with $l \neq i$ go exponentially fast to 0 as n grows. Thus the condition (12) reduces, for n large enough, to $F^*_i = p_S(F_{ij_i}) > p_S(F_{ij})$: this is satisfied by definition of j_i[4]. Now $A_{j_i} \supseteq U^n_i(\epsilon)$ implies that $p_i(A_{j_i})$ goes to 1 exponentially fast as n grows; for the same reason, $p_i(A_j)$ goes to 0 for each $j \neq j_i$ as n grows (recall that the A_j's form a partition of $\mathcal{O}^n$). This way, and taking (11) into account, we have proved that

$$\lim_{n\to\infty} P^W_{succ}(n) \;=\; \sum_{i=1}^{K} F^*_i \,.$$

Since $P^W_{succ}(n)$ is monotonically non-decreasing, we have proved that $P^W_{succ}(n) \leq \sum_{i=1}^{K} F^*_i$ holds true for each $n \geq 1$. This implies in turn the wanted statement.

Example 2 (modular exponentiation). We consider timing attacks against implementations of the modular exponentiation algorithm with blinding, used in public-key cryptography – see e.g. [14,16,17,5] and references therein. A typical implementation of modular exponentiation works as follows. The bits of the secret exponent are scanned from right to left, or vice-versa. When the ith bit is considered ($0 \leq i < N$), either one or two modular multiplications are performed, depending on whether the i-th bit is 0 or 1. In timing attacks, the attacker tries to reconstruct the secret key by sampling the duration of several independent executions of the algorithm. To an implementation as described above there corresponds an IHS where: $\mathcal{S} = \{0, 1\}^N$ is the set of secret keys, i.e. the possible exponents of the algorithm, over which we assume a uniform distribution can be assumed; $\mathcal{O} = \{t_1, t_2, ...\}$ is the finite set of possible execution times; $p(t|s)$ is the probability that, depending on the deciphered message, the execution of the algorithm takes times t given that the secret key is k. As argued in [5], it is sensible to assume that any two keys having the same Hamming weights are indistinguishable in $\mathcal{H}$. Therefore,

[4] If there is more than one index j maximizing $p_i(F_{ij})$, then the choice of j_i gets more involved: among those j's that maximize $p_S(F_{ij})$, one chooses the one that maximizes $p_S(F_{i'j})$, where $p_{i'}(\cdot)$ is the distribution closest to $p_i(\cdot)$ in terms of KL-distance, if this j is unique; otherwise one must look at the second closest distribution $p_{i''}(\cdot)$, and so on. We omit the details here.

we have $N+1$ indistinguishability classes. From each of them we choose a representative s_i^* of probability $\pi_i = \frac{1}{2^N}$. Applying Theorem 1, we find $P_e(n) \to 1 - \frac{N+1}{2^N}$, which for realistic values of N, is very close to 1. E.g., for $N = 1024$, the attacker gets on the limit $\log(1025) \approx 10.01$ bits of information leakage out of 1024.

One would then like to prove that this small leakage is not concentrated in few individual bits of the exponent, which would make them potentially vulnerable. For instance, let us examine the error probability of guessing the least two significant bits of the exponent. Let W be the partition of $\mathcal{S}$ s.t. $s \sim_W s'$ iff $s \bmod 4 = s' \bmod 4$. We apply Theorem 3 to P_e^W. We have four $\sim_W$-classes $E_0, ..., E_3$, that intersect with the $N+1$ classes C_i to form $4(N+1)$ classes F_{ij}. Assume N even. For all $i = 0, ..., \frac{N-2}{2}$, the class F_{ij} that has more elements, hence determines the probability F_i^*, is F_{i0}; by symmetry, for $i = \frac{N-2}{2} + 1, ..., N$ the class with more elements is F_{i3}. For $i = \frac{N}{2}$, instead, we can choose between F_{i1} and F_{i2}. According to Theorem 3 then

$$P^W_{succ} \;\to\; \sum_{i=0}^{N} F_i^* \;\approx\; \frac{1}{2^N}\Big(\sum_{i=0}^{N-2}\binom{N-2}{i}\Big) \;=\; \frac{1}{4}\,.$$

Thus, asymptotically the observations do not increase the prior probability of success, which is already $\frac{1}{4}$. In terms of information leakage, one gets $\mathcal{L}^W(n) \to\approx 0$. One can generalize this reasoning to the case where W represent the least m significant bits, and arrive at similar conclusions.

5 Example 1: Unlinkability in Threshold Mix-Nets

Statistical attacks against anonymity protocols may take advantage of sender-receiver relationships that remain fixed through repeated rounds of the protocol. In this section, we consider the case of a *mix network*, a concept due to Chaum [10]. In a mix-network, messages are relayed through a sequence of trusted intermediary nodes, called *mixes*, in order to hide sender-receiver relationships (*unlinkability*). In the scenario we consider, a single mix is used by a number of senders and receivers. The *threshold* of the mix is $b+1$: at each round, the mix waits for $b+1$ messages from the senders and then distributes the messages to the corresponding receivers. We consider the situation where one of the senders is always Alice, with her receiver being always a node Bob, initially unknown to the attacker. The recipients of the remaining b messages are assumed be chosen at random in a set of nodes $R_1, ..., R_N$. A similar scenario is at the basis of the statistical disclosure attack by Danezis [13]. We analyse the situation of a local eavesdropper that observes one fixed receiver, say R_j, and after each round is able to tell whether at least one message has reached R_j. More sophisticated forms of eavesdropping could be easily accommodated (e.g. attacker observing all the nodes), but would not change significantly the outcome of the analysis. The task of the attacker is to discover which node is Bob; or at least, to tell if Bob is or not the observed node, R_j.

We can model the scenario described above by an IHS $\mathcal{H}$ where: the set of states is given by all possible nodes (potential receivers of Alice's messages), that is $\mathcal{S} = \{R_1, \ldots, R_N\}$, with $p_S(R_i) = \frac{1}{N}$ for each $i = 1, ..., N$; the set of observations is $\mathcal{O} = \{0, 1\}$, where $o = 1$ iff R_j has received at least one message at the end of the round.

The conditional probability matrix $p(\cdot|\cdot)$ is then given by the following equalities:

$$\begin{aligned} &p(0|R_j) = 0 & &p(1|R_j) = 1 \\ &p(0|R_i) = (1 - \tfrac{1}{N})^b & &p(1|R_i) = 1 - (1 - \tfrac{1}{N})^b \text{ for all } i \neq j. \end{aligned}$$

Here, the first row means that, if Bob=R_j, then the attacker will observe at least one message with certainty. The second row means that, in case Bob is any node different from R_j, then the attacker will observe 0 messages only if all the b messages – other than the one sent to Bob – are not sent to R_j (Alice surely does not send to R_j). In other words, except for a permutation of the rows, we have the matrix below. Here the last row refers to R_j. This means that there are only two classes of indistinguishability: $\mathcal{S}/\equiv$ is $\{C_1, C_2\}$, with $C_1 = \{R_j\}$ and $C_2 = \mathcal{S} \setminus \{R_j\}$.

$$\begin{bmatrix} (1 - \frac{1}{N})^b & 1 - (1 - \frac{1}{N})^b \\ \vdots & \vdots \\ (1 - \frac{1}{N})^b & 1 - (1 - \frac{1}{N})^b \\ 0 & 1 \end{bmatrix}$$

We first apply apply Theorem 1 to $\mathcal{H}$, which will tell us what is the error probability in case the attacker wishes to know exactly who is Bob. We can set $I = \{i, j\}$, for any $i \neq j$, and get the following bound:

$$P_e(n) \leq \left(1 - \frac{2}{N}\right) + \frac{2}{N}\left(1 - \left(1 - \frac{1}{N}\right)^b\right)^n .$$

As expected, the limit value $1 - \frac{2}{N}$ is > 0, and the security of the system increases as N increases. The corresponding asymptotic information leakage is $\log(N \cdot \frac{2}{N}) = 1$, that is, the attacker gains 1 bit of min-entropy on the limit about the identity of Bob.

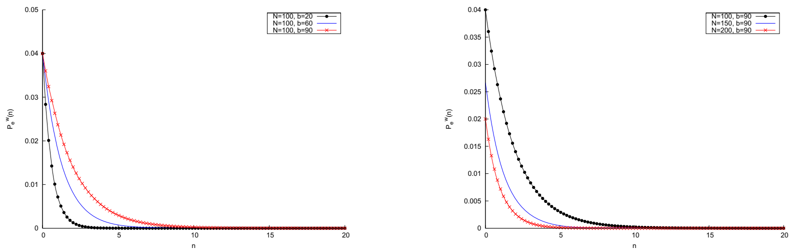
To see qualitatively *what* the single bit gained by the attacker corresponds to, we analyse the error probability with respect to the view $W \in \{0, 1\}$ given by:

$$W = 1 \text{ iff } S = R_j .$$

That is, W yields 1 iff Bob is R_j. The partition induced on $\mathcal{S}$ by W coincides with $\equiv$, hence its classes are C_1, C_2. Concerning the sets F_{ij}, we note that: $F_{11} = \{R_j\}$, $F_{12} = F_{21} = \emptyset$ and $F_{22} = \mathcal{S} \setminus \{R_j\}$. Since the distribution on the states is uniform, we have: $F_1^* = \frac{1}{N}$ and $F_2^* = 1 - \frac{1}{N}$. Take $I = \{i, j\}$ as defined as above. According to Theorem 2, the limit of $P_e^W(n)$ vanishes, moreover

$$P_e^W(n) \leq \frac{2}{N}\left(1 - \left(1 - \frac{1}{N}\right)^b\right)^n .$$

The attacker's success probability of guessing whether R_j =Bob or not approaches very fast 1. It is also interesting to study the behaviour of the rate $\rho_I = -\log\left(1 - (1 - \frac{1}{N})^b\right)$ depending on b and N. It is easy to see that as b increases, ρ_I decreases; on the contrary, as N increases and b is kept fixed, ρ_I increases. The shape of $P_e^W(n)$ is illustrated qualitatively by the plots in the figures below: very few rounds of the protocols ($n < 10$) are sufficient to achieve $P_e^W \approx 0$.



(a) Plots of $P_e^W(n)$ depending on parameter b

(b) Plots of $P_e^W(n)$ depending on parameter N

As mentioned above, it is easy to repeat this kind of analysis with more sophisticated observations on the part of the attacker: we do not do so here for lack of space. On the other hand, note that just repeating this simple attacks for each of the potential Alice's receivers (that is, setting $R_j = R_1, R_2, ..., R_{N-1}$ in turn), would lead the attacker to uncover the identity of Bob after a low number of rounds. This is sufficient to show that the single threshold mix system is totally insecure.

6 Example 2: Privacy in Sparse Datasets

We consider datasets collecting *micro-data* – preferences, recommendations, transaction records, health histories and so on – about a large number of individuals. Datasets of this kind are sometimes published for commercial or research purposes. Making micro-data public poses serious threats to the privacy of individuals, even when the data are released in anonymized form – that is with personal identifiers, such as SSN's, removed. The risk is that an attacker, using a little of background information about a given individual and cross-correlation of attributes, might *re-identify* the individual within the dataset, leading to the disclosure of the whole set of her/his attributes. An example of this technique is the spectacular de-anonymization attack of Narayanan and Shmatikov against the Netflix Prize dataset [19][5].

In this section, we show that (sparse) datasets naturally arise as instances of IHS, and that assessment of statistical attacks against dataset privacy is easily accomplished using the general results of Section 4.

We view a dataset as a table $\mathcal{D}$, with rows and columns corresponding to individuals (or more generally, records) and attributes, respectively. Formally, $\mathcal{D} \in \mathcal{V}^{R \times A}$, where $\mathcal{V}$, R and A are finite nonempty sets of values, records and attributes, respectively. One can view any dataset $\mathcal{D}$ as an IHS $\mathcal{H}_{\mathcal{D}}$, as follows. Records are equiprobable states, that is we set $\mathcal{S} = R$ and let $p_S(\cdot)$ be the uniform distribution on R. Concerning observables, there is a variety of sensible choices, depending on the observation power one wishes to grant the attacker with. For instance, a sensible choice is $O \triangleq A \times \mathcal{V}$. Another choice,

[5] The Netflix Prize dataset collects anonymous movie ratings of 500,000 subscribers. Using background information publicly available from the Internet Movie Database, Narayanan and Shmatikov successfully re-identified known users within the Netflix dataset.

if $\mathcal{V}$ is a totally ordered, is to observe attributes and *ranges* of values. The last choice is more robust than the former in case the dataset is published in a perturbed form. In fact, even setting $\mathcal{O} \triangleq A$ is sensible, as just knowledge of non-null attributes of a record provides a great deal of information[6]. In any case, the technical development presented below does not depend on the specific choice of $\mathcal{O}$. Finally, the conditional probability matrix models the process of acquiring background information about the individuals in the dataset. Depending on its exact nature, this information might come from various sources, e.g. personal blogs, Google searches, or even a water-cooler conversation with a colleague (see [19]). For example, if $\mathcal{O} = A$, then it is sensible to assume that the background knowledge consists of randomly chosen attributes and set, for each record r and attribute a

$$p(a|r) \triangleq \begin{cases} \frac{1}{n_r} & \text{if } a \text{ is a non-null attribute of } r \\ 0 & \text{otherwise} \end{cases}$$

where n_r is the number of non-null attributes in the row of the dataset corresponding to r. Of course, non-uniform distributions can be equally accommodated, e.g. if it is felt that certain attributes are more likely to be publicly released than others.

Having shown how to model a dataset as a IHS, we have to point out that, in the formal development below, there is no need to restrict to IHS's of the form $\mathcal{H}_{\mathcal{D}}$. To work in full generality, we will just assume a dataset is simply an IHS.

In a sparse dataset, most of the entries in the table are null. Specifically, we consider a dataset sparse if, except possibly for a small fraction of records, for no record there is another "similar" record in the dataset. To make the notion of sparsity precise, we have first to make precise the notion of similarity between records. We will work with a similarity function $\mathrm{Sim} : \mathcal{S} \times \mathcal{S} \to [0, 1]$. The intuition underlying the following definition, which is different from that proposed in [19], is that the similarity of s' to s is related to the fraction of non-null attributes they share. More precisely, it is the fraction of non-null attributes that can be inferred on any of the two by looking at the other.

Definition 4 (similarity). *Given an* IHS $\mathcal{H}$*, for any* $s, s' \in \mathcal{S}$*, let* $\sigma_{ss'} = \mathrm{supp}(p(\cdot|s)) \cap \mathrm{supp}(p(\cdot|s'))$*. We set*

$$\mathrm{Sim}(s, s') \triangleq \min\{\, p(\sigma_{ss'} \mid s)\,,\; p(\sigma_{ss'} \mid s')\,\}\,.$$

Note that $\mathrm{Sim}(s, s') = 1$ iff $\mathrm{supp}(p(\cdot|s)) = \mathrm{supp}(p(\cdot|s'))$. The following notion of sparsity is not related to - not weaker nor stronger than - the one considered in [19]. It seems to be satisfied by typical sparse datasets, like the Neflix Prize [19,20]. In fact, our results extend, although in a different form, to the notion of sparsity of [19], but we shall not give any detail here for lack of space.

Definition 5 (sparsity). *Let* $\mathcal{H}$ *be a* IHS *with* $p_S(\cdot)$ *the uniform distribution. Let* $\epsilon > 0$ *and* $\delta > 0$*. We say* $\mathcal{H}$ *is* (ϵ, δ)-sparse *if*

$$\Pr\big(\max_{s:s\neq S} \mathrm{Sim}(S, s) \geq \epsilon\big) < \delta\,. \tag{13}$$

[6] See [19] for further considerations on the structure of sparse datasets.

Our results apply to a situation where the attacker gets to know an entire copy of the dataset.We begin with a result on error probability.

Theorem 4. *Let $\mathcal{H}$ be (ϵ, δ)-sparse, with $|\mathcal{S}| = N$. Then $P_e(n)$ reaches δ at a rate $-\log\epsilon$. More precisely, $P_e(n) \leq \delta + \frac{1}{2}[(1-\delta)^2 N + 2(1-\delta) + \frac{1}{N}]\epsilon^n$.*

Proof. By definition of sparsity, it is possible to find a subset of the records, say $\mathcal{R} = \{s_i^* | i \in I\}$, s.t. for each $s \in \mathcal{R}$, there is no other record in $\mathcal{S}$ which is ϵ-similar to s, and such that $p_S(\mathcal{R}) \geq 1 - \delta$. Moreover, by uniform distribution of the probability mass on records, we can choose the size of I satisfying $\frac{|I|-1}{N} < (1-\delta) \leq \frac{|I|}{N}$, which means $(1-\delta)N \leq |I| < (1-\delta)N + 1$. Next note that, with the notation introduced in Section 4 and by virtue of Remark 1(b), for any $i, j \in I$ with $i \neq j$, the Chernoff information c_{ij} satisfies: $c_{ij} \geq -\log \mathrm{Sim}(s_i^*, s_j^*) \geq -\log\epsilon$. Applying Theorem 1 we get the thesis.

In some cases, all the adversary needs to determine about a record its "similarity class". In fact, knowledge of this class already provides him with almost all the information about the record. If this class is disclosed then a privacy breach has occurred. The next definition formalizes this intuition. Recall from (7) that $\sim_W$ is the equivalence relation induced on $\mathcal{S}$ by W.

Definition 6 **((ϵ, δ, ρ)-breach).** *Let $\mathcal{H}$ be a* IHS. *Consider a partition W of $\mathcal{H}$ such that whenever $s \sim_W s'$ then $\mathrm{Sim}(s, s') \geq \epsilon$. We say W is an* (ϵ, δ, ρ)-breach *if $P_e^W(n)$ reaches δ at rate ρ.*

The following result establishes strong upper bounds on the resistance to privacy breaches in sparse datasets (the proof is reported in the Appendix).

Theorem 5. *Any (ϵ, δ)-sparse* IHS *has an $(\epsilon, \delta, -\log\epsilon)$-breach W. In particular, to $P_e^W(n)$ the same bound applies as given for $P_e(n)$ in Theorem 4.*

Example 3. Real-world datasets tend to be extremely sparse. For instance, $(0.15, 0.2)$-sparsity in a dataset containing $N = 5 \times 10^5$ records should not be considered as exceptional (cf. [19, Fig.1], referring to the Netflix Prize dataset). Applying the bound of Theorem 4 to these figures, we see that already after coming across $n = 10$ randomly chosen attribute values of a target individual, the probability of uncorrect re-identification in the dataset is < 0.201. This may still seem quite high in absolute terms. Consider, however, that the success probability prior to the observations was $\frac{1}{5\times10^5}$. In terms of information leakage, this means that the attacker has obtained $\mathcal{L}(10) \approx 18.6$ bits of min-entropy, out of $\log N \approx 18.9$. The privacy breach is therefore absolutely relevant. Note that attacks against real-world datasets can exploit specific features of the target and get more impressive success probabilities [19].

7 Conclusion

We have put forward a model to analyse a variety of statistical attacks in a uniform fashion. This permits the assessment of systems security against passive eavesdroppers both at the global level and at the level of specific partitions of the secrets. In particular, we give precise bounds for the probability of misclassification on the part of the attacker, characterising both the limit value and the rate of convergence of the error probability as a function of the number independent observations.

The last few years have seen a flourishing of research on quantitative models of information leakage. In the context of language-based security, Clark et al. [11] first motivated the use of mutual information to quantify information leakage in a setting of imperative programs. Boreale [4] extended this study to the setting of process calculi, and introduced a notion of rate of leakage, albeit with a different technical meaning than that considered in the present paper. Chatzikokolakis, Palamidessi and their collaborators have studied IHS's from the point of view of both capacity and error probability, but mainly confining to the case of a single observation [8,9,6,7]. The min-entropy based information leakage has been proposed by Smith [21], originally in the case of a single observation.

Backes and Köpf in [1] too consider a scenario of repeated independent observations, but from the point of view of Shannon entropy, rather than of error probability. An application of their setting to the modular exponentiation algorithm is the subject of [16], where the effect of *bucketing* on security of RSA is examined. This study has recently been extended to the case of min-entropy by Köpf and Smith in [17]. Earlier, Köpf and Basin had considered a scenario of adaptive chosen-message attacks [15]. Our previous paper [5] studies the asymptotic behaviour of information leakage. The bounds obtained there for the asymptotic rates are much looser than those we obtain here, though. Moreover, considerations on views are absent.

Our work is also related, at least conceptually, to the notion of probabilistic *opacity* as studied by Bérard, Mullins and Sassolas [3]. Indeed, although their setting is different – they work with finite-state machines – our partitions could be viewed as a generalization of the binary predicates they consider. Note however that [3] is based on Shannon entropy, and considers observations consisting of a single run of the system, rather than repeated observations, hence not statistical attacks. The Bayesian traffic analysis framework of Troncoso and Danezis [22] is tailored to the analysis of mix-networks, but mostly focuses on simulation rather than on formal models and analytical results.

As for future work, it would be natural to generalize the present scenario to the case where the attacker is given k tries for guessing the secret, with $k \geq 2$, rather than just one. Finally, the application to sparse datasets prompts a connection to databases privacy issues that deserves further attention.

Acknowledgments. The first author wishes to thank V. Shmatikov for a stimulating discussion on the notion of sparsity in datasets. Three anonymous ESORICS 2011 referees provided valuable comments.

References

1. Backes, M., Köpf, B.: Formally Bounding the Side-Channel Leakage in Unknown-Message Attacks. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 517–532. Springer, Heidelberg (2008)
2. Baignères, T., Vaudenay, S.: The Complexity of Distinguishing Distributions (Invited Talk). In: Safavi-Naini, R. (ed.) ICITS 2008. LNCS, vol. 5155, pp. 210–222. Springer, Heidelberg (2008)

3. Bérard, B., Mullins, J., Sassolas, M.: Quantifying Opacity. In: Proc. of QEST 2010, pp. 263–272. IEEE Society, Los Alamitos (2010)
4. Boreale, M.: Quantifying information leakage in process calculi. Information and Computation 207(6), 699–725 (2009)
5. Boreale, M., Pampaloni, F., Paolini, M.: Asymptotic information leakage under one-try attacks. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 396–410. Springer, Heidelberg (2011)
6. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Compositional Methods for Information-Hiding. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 443–457. Springer, Heidelberg (2008)
7. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative Notions of Leakage for One-try Attacks. In: Proc. of MFPS 2009. Electr. Notes Theor. Comput. Sci, vol. 249, pp. 75–91 (2009)
8. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Information and Computation 206(2-4), 378–401 (2008)
9. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the Bayes risk in information-hiding protocols. Journal of Computer Security 16(5), 531–571 (2008)
10. Chaum, D.: Untraceable electronic mail, return address, and digital pseudonyms. Communications of the ACM 24(2) (1981)
11. Clark, D., Hunt, S., Malacaria, P.: Quantitative Analysis of the Leakage of Confidential Data. Electr. Notes Theor. Comput. Sci. 59(3) (2001)
12. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2/e edn. John Wiley & Sons, Chichester (2006)
13. Danezis, G.: Statistical Disclosure Attacks. In: SEC 2003. IFIP Conference Proceedings, vol. 250, pp. 421–426 (2003)
14. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
15. Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In: ACM Conference on Computer and Communications Security, pp. 286–296 (2007)
16. Köpf, B., Dürmuth, M.: A Provably Secure and Efficient Countermeasure against Timing Attacks. In: CSF 2009, pp. 324–335 (2009)
17. Köpf, B., Smith, G.: Vulnerability Bounds and Leakage Resilience of Blinded Cryptography under Timing Attacks. In: CSF 2010, pp. 44–56 (2010)
18. Leang, C.C., Johnson, D.H.: On the asymptotics of M-hypothesis Bayesian detection. IEEE Transactions on Information Theory 43, 280–282 (1997)
19. Narayanan, A., Shmatikov, V.: Robust De-anonymization of Large Sparse Datasets. In: IEEE Symposium on Security and Privacy 2008, pp. 111–125. IEEE Computer Society, Los Alamitos (2008)
20. Shmatikov, V.: Personal communication (2011)
21. Smith, G.: On the Foundations of Quantitative Information Flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
22. Troncoso, C., Danezis, G.: The bayesian traffic analysis of mix networks. In: ACM Conference on Computer and Communications Security, pp. 369–379 (2009)

A Appendix

Unless otherwise stated, we use the notation and conventions introduced in Section 4.

Theorem 6 (Theorem 1). *Let I be a nonempty subset of $\{1, ..., K\}$. Let $\rho_I \stackrel{\triangle}{=} \min_{i,j\in I, i\neq j} c_{ij}$. Let $\pi_{\max} = \max_{i\in I} \pi_i$. Then, for all $n \geq 1$*

$$P_e(n) \;\leq\; (1 - \sum_{i\in I} \pi_i) \;+\; \frac{|I|^2}{2}\pi_{\max} 2^{-n\rho_I}\,. \tag{14}$$

As a consequence, $P_e(n)$ reaches $(1 - \sum_{i\in I} \pi_i)$ at a rate of ρ_I. In particular, by taking $I = \{1, ..., K\}$, we obtain that $\rho(P_e) \geq \rho_I$.

Proof. Fix $n \geq 1$. Let $\mathcal{R} = \{s_i^* | i \in I\}$ and $g : \mathcal{O}^n \to \mathcal{R}$ be a function satisfying: $g(o^n) = s_i^*$ implies $p(o^n|s_i^*)\pi_i \geq p(o^n|s_j^*)\pi_j$ for each $j \in I$. Note that g need not be MAP for $\mathcal{H}$, and that $g^{-1}(s) = \emptyset$ for $s \notin \mathcal{R}$. For each $i \in I$, let $A_i = g^{-1}(s_i^*)$ be the acceptance region for s_i^*. Then we have (the sums below run over s's s.t. $p_S(s) > 0$)

$$\begin{aligned}
P_e^g(n) &= \sum_{s\in\mathcal{S}} \Pr(g(O^n) \neq s | S = s) p_S(s) \\
&= \sum_{s\notin\mathcal{R}} \Pr(g(O^n) \neq s | S = s) p_S(s) + \sum_{i\in I} \Pr(g(O^n) \neq s_i^* | S = s_i^*)\pi_i \\
&= (1 - \sum_{i\in I} \pi_i) + \sum_{i\in I} p_i(A_i^c)\pi_i \\
&\leq (1 - \sum_{i\in I} \pi_i) + \sum_{i\in I} \sum_{j\in I, j\neq i} p_i(A_j)\pi_i \\
&= (1 - \sum_{i\in I} \pi_i) \;+\; \sum_{i\in I} \sum_{j\in I, j>i} p_i(A_j)\pi_i + p_j(A_i)\pi_j
\end{aligned} \tag{15}$$

where the inequality follows from $A_i^c = \cup_{j\in I\setminus\{i\}} A_j$ and a simple union bound, while the last equality is simply a rearrangement of summands. Now, we evaluate $p_i(A_j)\pi_i + p_j(A_i)\pi_j$ for each $i, j \in I$ and $i \neq j$.

Essentially by the same derivation given in [12, eqn.(11.239)–(11.251)], one finds that $p_i(A_j)\pi_i + p_j(A_i)\pi_j \leq \pi_i^\lambda \pi_j^{1-\lambda} 2^{-nc_{ij}}$, for a suitable $\lambda \in [0, 1]$. Since $\pi_i^\lambda \pi_j^{1-\lambda} \leq \pi_{\max}^\lambda \pi_{\max}^{1-\lambda} = \pi_{\max}$ and $c_{ij} \geq \rho_I$, we obtain

$$p_i(A_j)\pi_i + p_j(A_i)\pi_j \;\leq\; \pi_{\max} 2^{-n\rho_I} \tag{16}$$

Now, if we plug the bound (16) in (15), and then factor out $\pi_{\max} 2^{-n\rho_I}$ and reorder the summands, we get

$$P_e^g(n) \;\leq\; (1 - \sum_{i\in I} \pi_i) \;+\; (\sum_{i\in I} \sum_{j\in I, j>i} 1)\,\pi_{\max} 2^{-n\rho_I}\,.$$

Now, use the fact that $(\sum_{i\in I} \sum_{j\in I, j>i} 1) = \frac{|I|(|I|-1)}{2} \leq \frac{|I|^2}{2}$, which shows that the wanted inequality holds for $P_e^g(n)$. But, from optimality of MAP, $P_e(n) \leq P_e^g(n)$, which completes the proof.

Theorem 7 (Theorem 2). *Let I and ρ_I be chosen as in Theorem 1. Let W be a view of $\mathcal{H}$. Let $\Pi_{\max} = \max_{i\in I} F_i^*$. Then*

$$P_e^W(n) \;\leq\; \sum_{j=1}^{L}(1-q_j^*) \;+\; (1-\sum_{i\in I} F_i^*\,) \;+\; \frac{|I|^2}{2}\Pi_{\max}2^{-n\rho_I}\,. \tag{17}$$

Proof. Denote a pair of indices $(i,j) \in \{1,...,K\}\times\{1,...,L\}$ as ij. For each $s \in \mathcal{S}$, define $\mathrm{ind}(s) = ij$ iff $s \in F_{ij}$. Fix $n \geq 1$ and any function $g' : \mathcal{O}^n \to \{1,...,K\}\times\{1,...,L\}$, and let $Succ'$ be the event $(g'(O^n) = \mathrm{ind}(S))$. That is, $Succ'$ is the event that g' correctly classifies the index (of the equivalence class F_{ij}) of S. Now define a guessing function for $\mathcal{H}$, $g : \mathcal{O}^n \to \mathcal{W}$, as $g(o^n) \overset{\triangle}{=} w$, where $g'(o^n) = ij$ and $w = \mathrm{argmax}_w q(w|s)$ for any $s \in E_j$ (note that the information about i provided by g' is ignored by g). Let Err be the event $(g(O^n) \neq W)$. We have

$$\begin{aligned} P_e^W(n) &= \Pr(Err, Succ') + \Pr(Err|\neg Succ')\Pr(\neg Succ') &(18)\\ &\leq \Pr(Err, Succ') + \Pr(\neg Succ')\,. &(19)\end{aligned}$$

Let us estimate $\Pr(Err, Succ')$ and $\Pr(\neg Succ')$ separately. It is an easy matter to prove that

$$\begin{aligned}\Pr(Err, Succ') &= \sum_{j=1}^{L}(1-q_j^*)\Pr(S\in E_j, Succ')\\ &\leq \sum_{j=1}^{L}(1-q_j^*)\,. \end{aligned}\tag{20}$$

We now estimate $\Pr(\neg Succ')$. Consider the new IHS $\mathcal{H}' \overset{\triangle}{=} (\{1,...,K\}\times\{1,...,L\}, \mathcal{O}, p'(\cdot), p'(\cdot|\cdot)\,)$, where $p'(ij) \overset{\triangle}{=} p_S(F_{ij})$ and $p'(o|ij) \overset{\triangle}{=} p_i(o)$. Note that $ij \equiv i'j'$ iff $i = i'$. Hence we have K distinct classes in this system, whose representatives are elements $s'_1 = 1j_1, ..., s'_K = Kj_K$ such that $j_i = \mathrm{argmax}_j p_S(F_{ij})$, hence $p'(s'_i) = F_i^*$, for $i = 1,...,K$. The corresponding representative distributions (rows of the matrix $p'(\cdot|\cdot)$) are $p'_1(\cdot) = p_1(\cdot), ..., p'_K(\cdot) = p_K(\cdot)$.

Now take the function g' above to be a MAP guessing function for $\mathcal{H}'$. Call $P'_e(n)$ the error probability of $\mathcal{H}'$: clearly, $\Pr(\neg Succ') = P'_e(n)$. Take $I \subseteq \{1,...,K\}$ and apply Theorem 1 to $\mathcal{H}'$ and I to get

$$\Pr(\neg Succ') \leq 1 - \sum_{i\in I} F_i^* \;+\; \frac{|I|^2}{2}\Pi_{\max}2^{-n\rho_I}\,. \tag{21}$$

When we plug the bounds (20) and (21) into (19), we get the wanted result.

Theorem 8 (Theorem 5). *Any (ϵ,δ)-sparse* IHS *has an $(\epsilon,\delta,-\log\epsilon)$-breach W. In particular, to P_e^W the same bound applies as given for P_e in Theorem 4.*

Proof. The proof is similar to that of Theorem 4. Consider the set $\mathcal{R} = \{s_i^* \mid i \in I\}$. Build the partition W as follows: take as blocks the singletons $\{s_i^*\}$, for $i \in I$, plus the blocks obtained by breaking $\mathcal{S}\setminus\mathcal{R}$ in such a way that any two records in the same block are ϵ-similar. Then apply Theorem 2 with W and I, taking into account the bounds for $|I|$ given in the proof of Theorem 4.

To Release or Not to Release: Evaluating Information Leaks in Aggregate Human-Genome Data

Xiaoyong Zhou, Bo Peng, Yong Fuga Li, Yangyi Chen, Haixu Tang, and XiaoFeng Wang

Indiana University, Bloomington

Abstract. The rapid progress of human genome studies leads to a strong demand of aggregate human DNA data (e.g, allele frequencies, test statistics, etc.), whose public dissemination, however, has been impeded by privacy concerns. Prior research shows that it is possible to identify the presence of some participants in a study from such data, and in some cases, even fully recover their DNA sequences. A critical issue, therefore, becomes how to evaluate such a risk on individual data-sets and determine when they are safe to release. In this paper, we report our research that makes the first attempt to address this issue. We first identified the space of the aggregate-data-release problem, through examining common types of aggregate data and the typical threats they are facing. Then, we performed an in-depth study on different scenarios of attacks on different types of data, which sheds light on several fundamental questions in this problem domain. Particularly, we found that attacks on aggregate data are difficult in general, as the adversary often does not have enough information and needs to solve NP-complete or NP-hard problems. On the other hand, we acknowledge that the attacks can succeed under some circumstances, particularly, when the solution space of the problem is small. Based upon such an understanding, we propose a risk-scale system and a methodology to determine when to release an aggregate data-set and when not to. We also used real human-genome data to verify our findings.

1 Introduction

With rapid advancement in genome sequencing technologies, human genomic data has been extensively collected and disseminated to facilitate human genome studies (HGS). A prominent example is genome-wide association study (GWAS) [4], a research technique that has been demonstrated to be highly valuable for identifying the genetic factors underlying common diseases. In a GWAS study, a group of participants with a disease/phenotype of interest (cases) are genotyped to compare the statistical features of their single-nucleotide polymorphisms (SNPs)[1] to those of the individuals without the disease/phenotype (controls). It is highly desired that the DNA data gathered during this process can be conveniently accessed by other researchers, which will greatly benefit the HGS community. Such data dissemination, however, needs to be balanced with the protection of participants' privacy, which is of paramount importance to this kind of research: for example, revealing the identity of a case individual in a GWAS relates her to the disease under the study, which can have serious consequences such as denial of access to health/life insurance, education, and employment. Prior research

[1] Common terminologies of genomics are summarized in Appendix A.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 607–627, 2011.

shows that raw DNA data (genotypes) is often too risky to publish even after removal of explicit identifiers (such as name, social security number, etc.), as de-anonymization of a participant's identity can happen through examining the genetic markers related to her observable features (a.k.a. phenotypes) [8][2]. What has been thought to be of low risk is *aggregate genome data*, such as *allele frequencies*, i.e., the frequencies of different SNP values, because such data covers an individual's information with that of others. As an example, the NHGRI/NIH used to make allele frequencies publicly available.

Aggregate Data Releases. A recent development in inference technologies, however, has completely changed the risk perception associated with the aggregate data. Particularly, Homer et al [39] discovered that the presence of an individual in a case group can be reliably determined from allele frequencies using the victim's DNA profile, which can be acquired, for example, from a single hair or a drop of blood. In response to this finding, the NIH swiftly removed all aggregate genome data from the public domain to protect the participants of HGS and avoid legal troubles [2]. Today, those who want to access the data have to file an application and sign an agreement, a complicated procedure that is time consuming. This becomes a hurdle to the dissemination of the data critical to HGS, and as a result, provokes intensive debates [10]: some researchers pointed out that the NIH may have overreacted, as the attack power achievable over at least some data-sets can be very limited [51,21]. On the other hand, such agreement-based protection has been found to be insufficient, as confidential user information can still be derived from other public sources: a recent study [52] shows that even the test statistics (e.g., p-values, r-squares) calculated from allele frequencies and published in HGS papers give away a significant amount of information, in some cases enough for identifying participants or even recovering portions of their DNA sequences. To make things worse, HGS researchers typically receive little guidance on what they are not supposed to share. Oftentimes, fine-grained allele frequencies/test statistics can be directly acquired from the authors of HGS papers.

Our Work. The current way aggregate human DNA data is handled indicates a disturbing lack of understanding of its privacy implication: such data have been both overprotected, which unnecessarily restricts their availability to the HGS researchers, and underprotected, which exposes the HGS participants to privacy threats. Crucial to the progress of the human genome research, therefore, becomes an in-depth study on how to evaluate the information leaks in the aggregate data and determine when they are safe to release, which also poses a challenge to the privacy researchers. This paper reports our research that makes a first step toward this end. We consider two types of common aggregate data, the allele frequencies for both individual SNPs and SNP pairs, and the test statistics derived from the frequencies. Such data is studied under two typical threats, *identification attack* that uses an individual's DNA profile to determine her relation with an aggregate data-set [39,52,46], and *recovery attack* that re-constructs individuals' SNP sequences from such data. Our paper investigated the feasibility of these attacks on different data based on information-theoretic and computational analyses. We further explored the potential to build a risk scale system.

[2] The NIH's guideline for sharing GWAS data [8] explicitly states "the NIH takes the position that technologies available within the public domain today, and technological advances expected over the next few years, make the identification of specific individuals from raw genotype-phenotype data feasible and increasingly straightforward".

Contributions. We summarize the contributions of this paper as follows:

•*Fundamental studies on information leaks in aggregate data.* We performed both information-theoretic and complexity analyses on the common threats to different types of aggregate data. Our research sheds light on the fundamental questions on whether an attack on a specific data-set is feasible and how difficult it can be. Of particular importance here is our consideration of the special features of human genomes, which, as we show in the paper, can have significant impacts on the answers to these questions.

•*Preliminary research on a risk-scale system.* We propose a risk-scale system to classify aggregate data and guide the release of such data. Our research, though preliminary, is the first attempt to evaluate the risk of information leaks in a broad spectrum of aggregate data, including both single and pair-wise allele frequencies and different test statistics.

Roadmap. The rest of the paper has been organized as follows: Section 2 introduces background knowledge; Section 3 and 4 elaborate our research on the data release problems; Section 5 surveys the related research and Section 6 concludes the paper and discusses the future research.

2 Backgrounds and Assumptions

2.1 Aggregate Human-Genome Data

Our research has been conducted on two types of aggregate genomic data, *allele frequencies* and *test statistics*. Both are among the most valuable data to human genome research and are also most widely disseminated: for example, the former has been published by the NIH [7] and the latter are elaborated in every GWAS paper [53,47,44,25].

Each SNP has two alleles, encoded as 0 (major) or 1 (minor). Using this encoding scheme, the DNA profiles (containing the nucleotide sequences of the participants) of N individuals L SNPs, could be simply represented as a $N \times L$ matrix. Figure 1 gives an extremely small sample of encoded SNP profiles of 5 participants and 8 SNPs. The *single-allele frequencies* f_i^p of a SNP site are the frequencies of the site's 'alleles, and the *pair-wise allele frequencies* f_{ij}^{pq} of a SNP pair represent the frequencies of site i and j of the four allele combinations: $pq \in \{00, 01, 10, 11\}$. Note that allele frequencies can be simply calculated from allele counts by dividing N (e.g. $f_{ij}^{pq} = C_{ij}^{pq}/N$).

From the allele frequencies, test statistics are often computed in different human-genome studies. Particularly, GWAS researchers utilize *association tests* to detect the SNPs related to the disease under the study. These tests compare the single-allele frequencies of the case population with those of the control population, in the hope of

$$\begin{array}{c|cccccccc} & \multicolumn{8}{c}{L} \\ \hline & 0&0&0&0&0&1&0&0 \\ & 0&1&1&0&1&0&0&0 \\ N & 1&0&0&1&0&0&0&0 \\ & 1&0&0&0&0&0&0&1 \\ & 0&1&0&1&1&1&1&1 \end{array}$$

Fig. 1. A 0-1 encoded SNP profiles of $N = 5$ individuals and $L = 8$ SNPs

Data	Name	Sample
C_i	single allele count for SNP i (major)	$C_1 = 3, C_3 = 4$
C_{ij}^{pq}	pair wise allele counts for SNP i and j	$C_{12}^{10} = 2, C_{13}^{00} = 2$
C_{ij}^{p*}	single allele count for SNP i	$C_{12}^{1*} = 2$
r_{ij}^2	r-square, measures association and LD	$\frac{(C_{ij}^{00}C_{ij}^{11} - C_{ij}^{01}C_{ij}^{10})^2}{C_{ij}^{0*}C_{ij}^{1*}C_{ij}^{*0}C_{ij}^{*1}}$

Fig. 2. Routinely published data (single allele counts without superscript means major counts, e.g. $C_i = C_i^0$)

identifying the genetic marker of the disease. The significance of each SNP (i.e., the strength of its tie to the disease) is measured by a p-value. Typically, those with p-values below 10^{-7} are selected as putative markers. Such marker-disease associations can also be quantified using other test statistics such as odds ratios.

In addition to analyzing individual SNPs, a GWAS also examines the putative marker's associations with other SNPs in the same genetic locus, called *linkage disequilibrium* (LD) [45], which could also have a connection with the disease. LD of a locus is typically measured by the test statistics such as D' and r-square, which are calculated from pairwise allele frequencies of the locus. Sometimes, researchers further analyzed the allele combinations involving multiple correlated SNPs, i.e., *haplotypes*, which are inferred from *genotypes* through a class of *phasing algorithms* [1,50,49].

Figure 2 shows how to calculate these test statistics and some sample values for Figure 1, which are routinely published in HGS papers [28,48,47,53]. Oftentimes, these papers include the p-values of hundreds of SNPs and figures that illustrate their LDs. More detailed information can also be acquired from the authors. In our research, we focused on p-values and r-squares, the two most-commonly reported test statistics.

2.2 Threats

The threats studied in our research include *identification attack* and *recovery attack*, two major privacy concerns in human genome research. The first identification attack on aggregate data has been proposed by Homer, et al [39], which requires availability of a SNP profile from the victim. The objective here is to determine the presence of an individual in the case group, so as to relate her to a disease. To this end, the attacker runs a statistic test that evaluates whether the victim's SNP profile is independent from the single-allele frequencies of the case population. Let $Y_j \in \{0, 1\}$ be the allele of SNP j in the profile, and $\hat{f}_j^0$ and f_i^0 be the major allele frequencies of that SNP in the case population and a reference population, respectively. Homer's attack measures the following distance:

$$D(Y_j) = |Y_j - f_j^0| - |Y_j - \hat{f}_j^0| \qquad (1)$$

Under the assumption that the distributions of individual allele frequencies are identical in the case and the reference, the sum of $D(Y_j)$ across a large number of SNPs follows a normal distribution with a zero mean if the victim is not present in the case group. Otherwise, the sum becomes positive and significantly deviates from the mean. In their paper, the authors report identification of a case individual with a extremely low false positive rate, given 25,000 SNPs of the victim. This line of research has been followed by multiple research groups [46,21,51,40,52]. Particularly, Sankararaman, et al [46] utilized the likelihood ratio test to estimate the upper-bound of the identification power achievable on single-allele frequencies. They also built a tool called SecureGenome [11] to evaluate such a threat on different data sets.

Besides single-allele frequencies, pair-wise allele frequencies and test statistics were also found to leak out a substantial amount of information. In prior research [52] , it was found that the identification attack can happen to even the test statistics published in GWAS papers, through a statistical test based upon *signed* r values. Given N sequences of L neighboring SNPs in the genome, the signed r_{ij} between two SNPs i and j ($1 \leq i < j \leq L$) is defined as $r_{ij} = \frac{C^{11}C^{00} - C^{01}C^{10}}{\sqrt{C^{1*}C^{0*}C^{*1}C^{*0}}}$, where C^{pq} is the pair-wise allele counts, i.e. the number of the sequences with allele p ($p \in \{0,1\}$) at SNP i and allele q ($q \in \{0,1\}$) at SNP j, and C^{p*} and C^{*q} are single allele counts. r_{ij} can be computed from r_{ij}^2 (Figure 2) except its sign. Like Homer's approach, the attack needs a reference population whose r values are denoted by r^R, in addition to the case population (r^C), and a SNP profile from the victim in which $Y_{ij}^{pq} \in \{0,1\}$ indicates whether her SNP pair ij has a pair-wise allele pq. A test statistic T_r is thus constructed as follows:

$$T_r = \sum_{1 \le i < j \le N} (r_{ij}^C - r_{ij}^R) \cdot (Y_{ij}^{00} + Y_{ij}^{11} - Y_{ij}^{01} - Y_{ij}^{10}) \tag{2}$$

T_r is much more powerful than the statistical attacks on single allele frequencies [52], as it makes use of the relations among SNPs, the linkage disequilibrium, which contain much more information than individual SNPs. A problem here, however, is the need to know the signs, which is not typically released. They are determined in the prior work [52] by taking advantage of *integer constraints*, base upon the assumption that the published r-squares are calculated from allele counts (integers) and are not perturbed by noise.

The recovery attack aims at re-constructing the SNP sequences (i.e., haplotypes) used in an HGS: prior research [52] reports a successful restoration of 100 sequences involving 174 SNPs on a locus from their single and pair-wise allele frequencies. Note that these frequencies can be estimated through reverse engineering the test statistics published in GWAS papers [52]. Compared with the identification attack, such an attack can be more difficult to succeed and consume much more computing resources. However, it does not rely on the DNA profile from the victim.

An ideal privacy goal here is *differential privacy* [29], which ensures that two aggregated datasets differing from each other by one individual's data have indistinguishable statistical features. An example when this happens is that the data from a very large number of participants is aggregated so that the contribution of an individual becomes negligible. This privacy goal, once achieved, can defeat inference attacks using all kinds of background knowledge. However, this condition is known to be hard to satisfy in a practical system. For genomic data, the knowledge about the victim's DNA profile and a good reference population is deemed as a strong assumption in the adversary's favor [51, 21]. Based on such an assumption, we thoroughly studied the feasibility and complexity of these two types of attacks on the two types of datasets, and the methodology to determine whether a specific set of data is safe to release. Due to the space limit, this paper focuses on two most interesting scenarios where allele frequencies face the recovery attacks and test statistics are under the identification threat. The other two cases, i.e., the identification threat to allele frequencies and the recovery threat to test statistics, are much simpler: for example, the former has already been preliminarily explored by the prior research [11]. Our new findings on these cases can be found in a longer version of the paper [55].

2.3 Adversary Model

We consider a probabilistic polynomial time adversary who can not accomplish the task that needs exponential computing power, for instance, sampling an exponential space to determine a probability distribution over this space. Other than that, we assume the adversary has sufficient resources and perfect information at her disposal for individual attacks. Specifically, for the identification attack, we consider that the adversary has access to the victim's DNA profile and a good reference population with an allele distribution identical to that of the case population. This is the best resource such an attack can expect [52, 39]. For the attack involving test statistics, we assume that high-precision data is available, which affects the outcome of such an attack, as indicated in the prior research [52].

3 Recovery Threats to Allele Frequencies

Given a set of pairwise allele frequencies, a recovery attack aims at partially recovering the haplotype sequences of HGS participants, which is completely realistic according to prior research [52]. These sequences, once restored, can be used to re-identify these participants, a threat well recognized by the NIH (see Footnote 1 and [8]). This section reports a new methodology for determining the susceptibility of different allele-frequency data to such an attack.

3.1 The Problem

Figure 3 illustrates the recovery attack, in which the adversary attempts to recover a matrix, with each of its row vectors being a haplotype sequence, from the constraints of pairwise allele frequencies[3]. This problem can be formulated as a *haplotype matrix recovery problem* below:

Haplotype Matrix Recovery Problem. Consider an $N \times L$ haplotype matrix M that represents N haplotype sequences over L SNP sites. The set of pairwise allele frequencies of M is denoted by $d = \{f_{ij}^{pq}\}$, where p and q are the allele types at SNP sites i and j, respectively. Note that there are in total $\binom{L}{2}$ such pairs among L SNPs. Let S be the space of M (the matrix), and D be the space of d (the pairwise allele frequency). Given d and N, the adversary intends to recover the haplotype matrix, that is, to find an M' in S, which is equivalent to M ignoring the order of their row vectors.

It is conceivable that in some cases (some pairwise allele frequency d) the problem has unique *solution*: that is, there exists a unique matrix M, disregarding the ordering of its rows, that satisfies the constraints imposed by d, whereas in some other cases, the problem has no solution (i.e. the pairwise allele frequencies are not *satisfiable*), and in the remaining cases, the problem has multiple solutions. Figure 4 illustrates an example that multiple solutions exists for a given d. If there are multiple solutions and the intersection of all the solutions is small, when an attacker gets one solution, she has low confidence if any of the sequence in his solution is indeed in the original haplotype matrix.

Challenges in Risk Classification. To determine the risk scale of a given frequency set d, we first find out whether it has multiple solutions. If this is true and the overlap among these solutions is sufficiently small, we can comfortably put the set in the Green zone. Unfortunately, this decision turns out to be extremely difficult to make, because several problems on the haplotype matrix recovery are computationally hard. Specifically, we found that:

Theorem 1. *Determining if there is a haplotype matrix for a given pairwise allele frequency set is NP-complete.*

[3] Note that the pairwise allele frequencies contain all the information of single allele frequencies.

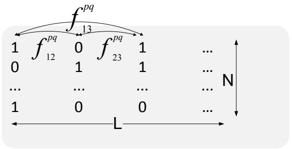


Fig. 3. Recovering a matrix from pairwise allele frequencies. Given a pairwise allele frequency set $d = \{f^{pq}_{ij}\}$, the attacker tries to recover the matrix satisfying d.

$$\begin{array}{ccc|ccc}
\hline
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 \\
\hline
\end{array}$$

Fig. 4. The left matrix and the right matrix have exactly the same single allele frequencies and pairwise allele frequencies, but do not share any single haplotype sequence

Corollary 1. *Determining the number of haplotype matrices for a given pairwise allele frequency set is NP-hard.*

Conjecture 1. Determining if a solution is unique for a given pairwise frequency set is Co-NP-complete.

Corollary 2. *Recovering one haplotype matrix for a given pairwise allele frequency set is NP-hard.*

Corollary 3. *Determining if there exists a solution for a given pairwise allele frequency set that does not contain a given row vector is NP-complete.*

Corollary 4. *Recovering one haplotype matrix for a given pairwise allele frequency set that does not contain a given row vector is NP-hard.*

Proofs are provided in Appendix B. Theorem 1 to Corollary 4 show that determining the existence of unique or multiple solutions for a given allele frequency set and recovering even single one of them are all hard problems. Note that proving *average-case* complexity is well known to be difficult [34]. Nevertheless, our empirical study using IBM Cplex [5] with parallel enabled suggests that at least the decision problems here do not seem to be easy in the average time. We randomly sampled 10 matrices of size 100×80 and put them on a workstation with 4 Quad-Core Xeon 2.93GHz processors, none of them could be solved within one week.

Determination of Risk Scales. In spite of the difficulty in finding the number of solutions, it is still plausible to estimate whether a given frequency set is likely to have multiple haplotype matrix solutions, by considering solely the size of the recovery problem as determined by two parameters, i.e., the number of SNP sites L and the number of haplotype sequences N. We compare the solution space $\|S\|$ and the frequency set space $\|D\|$. When $\|D\| \approx \|S\|$, the corresponding frequency set is likely to have a unique haplotype matrix solution. Conversely, when $\|S\| \gg \|D\|$, a data-set d becomes very likely to have multiple solutions. Intuitively, the distribution of the solutions over the different d tends to have a very small deviation: that is, it is unlikely that only a few have many solutions while the others have unique ones. Furthermore, because the distribution is over a large number of variables (i.e. the elements in the haplotype matrix) and it is very complicated, the adversary cannot estimate the distribution without using exponential computing power. The adversary, who is unsure about the uniqueness of the solution, but, on the other hand, is aware of the strong indications that multiple

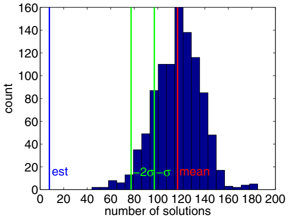


Fig. 5. Solution Distribution. (N = 40, L = 7, sample size = 1000, space ratio (estimated number of solutions) = 7.861, average = 116.855)

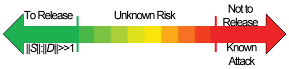


Fig. 6. Risk spectrum. When $\|S\|$: $\|D\| \gg 1$, data is placed in the Green zone. If there is a known attack, data must be placed in the Red zone. Otherwise further investigation is needed for the data (Yellow Zone).

solutions exist, will end up with little faith in any solution she is able to find. What is more, she may not even know how close to the real haplotype sequences her solution is, if $\|S\|$ becomes sufficiently large to ensure that many data-sets have multiple solutions.

Although it is difficult to rigorously define the distribution of solutions over d, we conducted an empirical study to verify our hypothesis that the solutions distribute near randomly. We randomly sampled 1000 haplotype matrices of size $N = 40$ and $L = 7$, and calculated their pairwise allele frequencies[4]. Using each set of these pairwise allele frequencies d as constraints, we computed for each instance all solutions that can be found by Cplex, a state-of-the-art NP solver [5] [5]. As expected, the distribution of the number of solutions is close to a normal distribution with a small standard deviation (Figure 5). The standard deviation (19) is on the same scale as the square root of the mean (116), indicating that it is unlikely that only a few d have many solutions while others have only a few or single solutions.

The above analysis indicates that we can have a shade-of-grey risk spectrum, as illustrated in Figure 6, which is approaching the Green end with the increase of the ratio $\|S\| : \|D\|$. Intuitively, this suggests that the larger the ratio, the less the adversary knows about the distance between her solution and the real one[6]. Upon the spectrum, we can use a distance threshold to determine when a frequency data-set can be designated to the Green zone. This research is elaborated in Section 3.2 and 3.3.

Towards the Red end of the spectrum, we proved that restoring a solution matrix from allele frequencies is NP-hard, even if the solution is known to be unique. However, we also acknowledge that the special features of human-genome data, particularly the LD relations among them, could make the problem tractable, as indicated in prior research [52]. Therefore, a conservative approach is to label a data-set "Red" only when

[4] We chose this problem scale because L and N met condition 3 which we will discuss shortly and the problem is small enough to be solved by Cplex in reasonable time.

[5] We did not enumerate all putative solutions. Instead, we set the populate limit of Cplex as 200 to save memory and time. Hence, the number of solutions shown here may be smaller than the actual number of solutions.

[6] An exception here is some special cases, for example, when the frequencies of the pairwise allele type 00 become 1 for all SNP pairs. Such cases, however, can be identified before the data being released.

it is found to be vulnerable to a known attack. Otherwise, the data-set is put in the Yellow zone, awaiting further investigation, if it is also not qualified for the Green zone. The details of this analysis are presented in Section 3.4.

3.2 When to Release

As discussed above, when the solution space becomes sufficiently larger than the space of allele-frequency sets, the threat of recovery attack can be mitigated, as the adversary cannot determine whether a given frequency data-set describes a unique set of SNP sequences. Here, we present an analysis on how large the solution space needs to be.

Solution-Space Analysis. Let us first consider the solution space S. For L SNPs, there are 2^L possible SNP sequences. The number of different solutions, each of which is an N by L haplotype matrix, is at least $\binom{2^L}{N}$, i.e., selecting N distinctive sequences from the 2^L sequences.

Then, we estimate the space of pairwise allele frequency sets D. Given N and a frequency set $d = \{f_{ij}^{pq}\}$, we can have a set of pairwise allele counts $\{C_{ij}^{pq}\}$, which directly determine the set of single allele counts $\{C_i\}$. Since for any SNP pair, the frequencies of one pairwise allele and one single allele are sufficient for inferring the frequencies of other alleles, pairwise or single, for the same SNP pair (see Inequality 3 in [52]), the set d is uniquely determined by $\{C_i\}$ and the set of pairwise major allele counts, which we denote by $\{C_{ij}\}$ for simplicity.

From the fact that C_{ij} and C_i can take any value in $[0, N]$ and there are $\binom{L}{2}$ SNP pairs and L single SNPs, we know that the number of different frequency sets d will not exceed $(N+1)^{\binom{L}{2}} \cdot (N+1)^L = (N+1)^{\binom{L}{2}+L}$. Comparing $\|S\|$ with $\|D\|$, we can get a necessary condition for the existence of multiple solutions: $\binom{2^L}{N} > (N+1)^{\binom{L}{2}+L}$. But it is too complex to use. Using Stirling's approximation, we get $\frac{2NL}{L(L-1)+2L}(1 - \frac{\log\frac{N}{e}}{L} - \frac{\log 2\pi N}{2NL}) > \log(N+1)$[7]. This gives us $\frac{2N}{L+1}(1 - \frac{\log\frac{N}{e}}{L} - \frac{\log 2\pi N}{2NL}) > \log(N+1)$. For $L > 200$, $1 - \frac{\log\frac{N}{e}}{L} - \frac{\log 2\pi N}{2NL} \approx 1$. Ignoring other constants, we get the following condition:

$$\frac{2N}{\log(N+1)} > L \tag{3}$$

Partial Recovery of Haplotype Matrix. The above analysis did not take into consideration the possibility that multiple solutions, although they exist, are close enough to each other for a given set of pairwise allele frequencies, e.g., there are a significant number of sequences shared between them. If this occurs and the attacker somehow recovered all the solutions (even though it is NP-hard, Corollary 1), and makes an intersection over these solutions, she knows the resulting common sequences must be in the case group. To defend against such attacks, we need stronger condition to assure the security of the pairwise allele frequency data to be released: for a specific haplotype sequence, there should exist another haplotype matrix solution that does not contain this sequence. When this happens, even if an attacker manages to obtain a solution (i.e. a set of haplotype sequences), she is not confident that *any* sequence in her solution is present in the actual haplotype matrix, because for any such sequence, there is always

[7] Unless otherwise specified, log means $\log_2$ in this paper.

another haplotype matrix that is equally likely to be the actual matrix and also does not contain this sequence (although to find this matrix is NP-hard according to Corollary 4). Similarly, even if the attacker obtained multiple solutions, the intersection of these solutions will not give her any confidence that the sequence in the intersection must be present in the actual matrix.

To get this stronger condition, we consider the solution space for a given instance d with N rows (sequences) and L columns (SNP sites), but one haplotype sequence in the original matrix is not in these solutions. This is equivalent to the entire matrix space, i.e., $\frac{2^{N \times L}}{N!}$, subtracted by the matrix space with one fewer row (set as the given haplotype sequence), i.e., $\frac{2^{(N-1) \times L}}{(N-1)!}$. By using the same analysis from above, we get the following condition:

$$\frac{2(N-1)}{\log(N+1)} > L \tag{4}$$

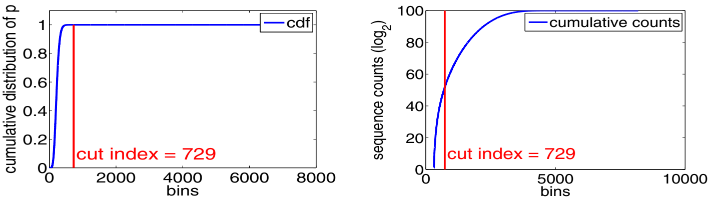
Once the size of a haplotype matrix (N and L) meets this condition, its solution space will become sufficiently large that the intersection of all of its solutions is unlikely to contain even one haplotype sequence. This condition is also very close to that of Condition 3.

Empirical Study. To verify whether the above privacy assurance is sufficient in practice, we conducted an empirical study on a number of small-scale problems. We randomly sampled 30 haplotype matrices that satisfy the condition (with $N = 40$ and $L = 8$), and for each haplotype sequence in the original matrix, we attempted to recover another haplotype matrix that did not contain this sequence but still has the same pairwise allele frequencies as those of the original matrix. Again, we used Cplex to search for all matrix solutions (with a populate limit of 200). In the end, for each of the haplotype sequences in the 30 matrices we sampled, at least 74 solutions were found that did not contain that sequence, indicating that given any haplotype sequence in a matrix, there likely exists an alternative solution (another haplotype matrix) associated with the pairwise frequency set of the original matrix, which does not include that sequence. This study shows that condition 4 can be used to estimate when a pairwise frequency set is unlikely to be vulnerable to an intersection attack.

3.3 The Impact of Human Genetic Structure

A critical pitfall in the analysis above is that it does not take into consideration the prominent features of human genome sequences. Instead of being random binary sequences (0 for major and 1 for minor allele) as assumed in our model, human genome sequences contain complex structures that are well studied in human genetics and can be inferred from publicly accessible human genome data [13, 6]. Thus, the adversary could simply examine a solution she finds to determine whether it looks like a human genome sequence. This leads to the further reduction of the solution space $||S||$. In this section, we present another analysis based upon a human genetic model.

Human Genetic Model. We model haplotype sequences with a Markov chain (MC), a standard approach extensively used in human genetic research for the modeling of the LD structure (single and pairwise allele frequencies) in a specific genetic locus [35, 42, 43]. Given L SNP sites, the model can be represented as a heterogeneous



(a) cdf

(b) counts

Fig. 7. The Markov Chain model for estimating the effective solution space. (a) Cumulative distributions of the probabilities of haplotype sequences, sorted in descending order of probabilities. Cutoff probability $\theta = 0.99999$. (b) Total number of most probable sequences vs. their cumulative probabilities. Vertical red lines represent the cutoff.

Markov chain with a sequence of L *states* $(X_1X_2...X_L)$, where $X_i \in 0, 1$, representing the major (0) or minor (1) allele, and an initial probability distribution (denoted by $P^0(X_1)$) as well as $L - 1$ different transition probability matrices (denoted by $P^i(X_{i+1}|X_i)$) are used to model the transition probabilities from the i-th state to the $(i + 1)$-th state, which are estimated from the single and pairwise allele frequencies using standard methods [35, 42, 43]. As a result, each of the 2^L haplotype sequences corresponds to a state sequence and the probability of observing it under the MC model can be computed by $P(X_1X_2...X_L) = P^0(X_1) \cdot \prod_{i=1}^{L-1} P^i(X_{i+1}|X_i)$. Once built from a group of haplotype sequences from human individuals (e.g. the case group or a reference group), the MC model can be used to evaluate the *effective* space of haplotype matrices that are likely sampled from real human individuals. Among totally 2^L possible haplotype sequences, the probabilities of observing some sequences are so low that they are deemed unlikely to appear in human genomes, owing to the strong associations among neighboring SNPs. These sequences should not be considered when estimating the solution space of haplotype matrices. Assume the probabilities of 2^L sequences and a threshold θ (close to 1, e.g. 0.99999) are given, the effective space of haplotype sequences can be estimated by the number of most probable sequences that have a cumulative probability greater than θ. This was achieved in our research through an approximation algorithm which is given in our technique report [55].

Evaluation. To estimate the solution space under a human genetic model, we phased 3008 sequences from WTCCC ch7 of 100 SNPs by using PHASE [3]. We chose $2T = 8192$ bins to estimate distribution of haplotype sequences under the MC model. As shown In Figure 7(a), with cutoff probability $\theta = 0.99999$, only 729 bins of $\approx 2^{52}$ most probable sequences are obtained, as compared to the entire space of 2^{100} haplotype sequences, which indicates that the incorporation of the human genetic model significantly reduces the effective space of haplotype sequences. Figure 8 shows the space comparison between $\|S\|$ and $\|D\|$. We could see that in the original analysis, we need about $5L$ sequences to ensure multiple solutions for the given pairwise allele frequencies. Defending against the intersection attack requires pretty much the same number of sequences as shown in the figure. To incorporate the human genetic model (the MC model), we need roughly $12L$ sequences.

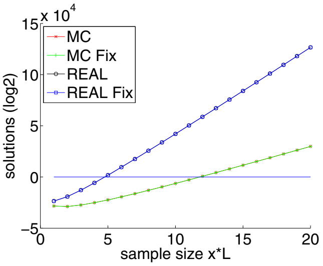


Fig. 8. Comparison between matrix space $\|S\|$ and constrain space $\|D\|$ for data fromWTCCC1 of SNP 100

3.4 When Not to Release

For the frequency set that cannot be put in the Green zone, its solution is likely to be unique. The adversary who finds the solution has reason to believe that it is the correct one. Here, we elaborate how to classify such a data-set.

Red-Zone Data. Although recovering SNPs sequences is NP-hard in general, the special features of human genome can enable the attack to succeed on at least some frequency sets. Prior research reports a successful attack on a data-set related to 100 SNP sequences and 174 SNPs from the FGFR2 locus [52]. The approach leverages the LD relations among these SNPs to break the matrix into small blocks in a way that preserves the strong inter-SNP relations within individual blocks. Such relations allow the adversary to first restore individual blocks and then use the aggregated relations between blocks to connect them together.

To avoid releasing the data vulnerable to the recovery attack as well as overprotecting those that can actually be disclosed, we suggest to test a frequency set with known attacks and assign it to the Red zone when it is exploited. If the attacks fail, we can label the data-set as “Yellow” to leave the decision on its release to the future research.

4 Identification Threat to Test Statistics

Besides allele frequencies, also widely disseminated by HGS are the test statistics computed from these frequencies. Particularly, HGS papers routinely report p-values and r-squares (r^2) over tens or even hundreds of SNP sites. Prior research [52] shows the key to an identification attack on such data is knowledge of the values of r or equivalently, their signs (given r^2). Once such information is given, we can use T_r [52] to decide whether a set of r-squares can be released, in the same way as SecureGenome [11] does to single and pairwise allele frequencies. Specifically, we can release such a data-set if given all correct signs, the achievable statistical power on it, as reported by T_r, is still below a threshold. However, when the power turns out to be high, a decision to keep the data off limit can be premature: after all, there we assume that all the signs are recovered, which is by no means easy in practice, as discussed later in this section. Therefore, a question becomes how to seek a “tighter bound”, allowing the statistics to be released when it is too difficult to recover a dangerous amount of information from them. This issue is addressed in this section.

The rest of the section presents our understanding of the problem: how sign recovery improves the chance of successful identifications and how difficult this can be done. Then, we come up with the yardsticks for releasing test statistics and describe a new potent attack that helps decide when data should be held from publication.

4.1 The Problem

An important question we are asking is how many correct signs a successful attack needs. The answer sheds light on the conditions under which the attack becomes ineffective. To find out the answer, we can analyze the relations between the rate of the correct signs used in an optimal test and the statistical power it can achieve on a particular data-set. Specifically, given a *rate* of correct signs α, we can randomly assign correct signs to the r of a fraction α of SNP pairs, and then run T_r under the assignment to determine its power, i.e., the rate of successful identifications. This test needs to be conducted repeatedly for each rate of correct signs, to get the maximum power under different sign assignments. In this way, we can obtain an estimated power-sign relation, and then use a threshold to determine the maximum rate of correct signs that will not pose a serious identification threat.

Complexity of Releasing Statistics. Given a threshold α ($\alpha \in [0, 1]$) of the correct sign rate, a set of test statistics (r-squares) can be placed in the Green zone if the adversary cannot correctly recover as many as α of all $\binom{L}{2}$ signs. This can be ensured if the set of r-squares is mapped to multiple sets of *valid* signed r values, and the overlap among these sets is below the threshold α. When this happens, the adversary, even if she can recover all these sets of signed r values, cannot identify enough signs with any confidence for a successful attack. Obviously, given $\binom{L}{2}$ r-squares over L SNP sites, there are totally $2^{\binom{L}{2}}$ possible *sign assignments*, with each of them corresponding to a different set of signed r values. However, not all of such assignments are valid: many of them do not correspond to any haplotype matrix, as those assignments lead to the r values inconsistent with each other.

We studied a *sign recovery problem*: given a set of r-square values r_{ij}^2 over L SNP sites, a set of single allele frequencies f_i ($i = 1, 2, ..., L$), which could be recovered from p-values [52], and the total number of sequences in the case group (N), find a set of signed r values $\hat{r}_{ij}$ so that (1) $r_{ij}^2 = \hat{r}_{ij}^2$; and (2) $\hat{r}_{ij}$ are *valid*, i.e. there exists a haplotype matrix whose pairwise allele counts C_{ij}^{pq} ($p, q \in 0, 1$) satisfy $N \cdot f_i = \sum_{q \in \{0,1\}} C_{ij}^{0q}$ for all i and j, and $r_{ij} = \frac{C_{ij}^{00} C_{ij}^{11} - C_{ij}^{01} C_{ij}^{10}}{C_i^0 C_i^1 C_j^0 C_j^1}$. Similar to the *haplotype matrix recovery problem*, several key problems related to the sign recovery problem are computationally hard if we assume the haplotype matrix has more than just a few rows (haplotype sequences). This can be satisfied by all real HGS studies, which typically contains hundreds of individuals. Specifically, under this condition, we show that:

Theorem 2. *Determining if there exists a set of sign assignments of r for a given set of r-squares and single allele frequencies is NP-complete.*

Corollary 5. *Recovering a valid sign assignment for a given set of r-squares and single allele frequencies is NP-hard.*

Corollary 6. *Finding the number of valid sign assignment for a given set of r-squares and single allele frequencies is NP-hard.*

The proofs are provided in technique report [55]. We note that these results have strong implications on classifying an r-square set into Green or Red zones. Briefly, an adversary faces the following computational difficulty: assume that she manages to recover some sets of signs from r-squares, which itself is NP-hard; she still has no clue whether there are any other valid sign assignment and how many correct signs have been discovered in her solution. In other words, she will not have any reasonable confidence in the identification she makes from the r-square data-set. There is an exception, though: if the solution space of valid sign assignments (or equivalently their corresponding haplotype matrices) is sufficiently small, for example, as small as the space of r-squares, then the adversary has a good reason to believe that every set of r-squares has a unique valid sign assignment. Here the situation is analogous with that in Case 2 (Section 3). Similarly, we need a solution-space analysis to ensure that the adversary cannot get any useful information from a data-set to be released.

4.2 When to Release

Before placing a data-set to the Green zone, we need to ensure that the adversary cannot recover enough signs from it to achieve any significant identification power. Consider that a polynomial-time adversary learns from the ratio between the space of r-squares $\|R^2\|$ and the space of matrices $\|S\|$ that an r-square set can have κ solutions. Given a specific set of r-squares, she has no reason to believe that the set has fewer solutions, because she can neither determine the exact number of solutions nor sample the exponential space S (when N and L are large) to estimate the distribution of matrices over r-square sets. Also, recovering all these matrices is NP-hard and therefore the adversary has no clue how many different sets of valid signs exist, except that the number will not exceed κ. When κ is sufficiently large, the adversary, even after she manages to get a set of signs, does not know whether the overlap among all sets (which can be as many as κ) goes above $1-\alpha$ of all $\binom{L}{2}$ signs.

Solution-Space Analysis. Therefore, the condition for the release of an r-square set is that $\|S\| : \|R^2\|$ should be sufficiently large to ensure that the adversary does not know whether she recovers enough correct signs. As described in Section 3, $\|S\| \approx 2^{LN}(\frac{N}{e})^{-N}(2\pi N)^{-1/2}$. Since the space size of the r values is approximately $(N+1)^{\binom{L}{2}+L}$, from r to r-squares, the space shrinks to $\|R^2\| \approx \frac{(N+1)^{\binom{L}{2}+L}}{2^{\binom{L}{2}}}$. To ensure multiple solutions, we need $\|S\| > \|R^2\|$, which gives:

$$\frac{2N}{\log(N+1)-1} > L \tag{5}$$

For example, for a locus involving 100 SNPs, at least 225 individuals (450 haplotype sequences) should be in the case group to ensure the existence of multiple solutions. Not surprisingly, this is less stringent than the condition of placing a set of pairwise allele frequency in the Green zone (where one needs to have at least 500 sequences for a 100-SNP locus), because r-squares contain less information than the pairwise allele frequencies. To further prevent the adversary from identifying more than $1-\alpha$ of the correct signs, we need to make it possible to have an element in R^2 be mapped by

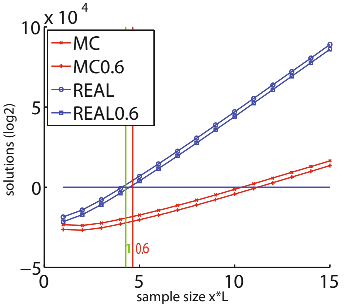


Fig. 9. Comparison between matrix space $\|S\|$ and $\|R^2\|$ for data from WTCCC1 of $L = 100$ SNP. Vertical line shows the required sample size estimated from formula 5 and 6 and then added by a buffer of $0.5L$.

at least $2^{(1-\alpha)\binom{L}{2}}$ elements[8] in S. To ensure this, we must have that $\|S\|$ is at least $2^{(1-\alpha)\binom{L}{2}}$ times as large as $\|R^2\|$. This ultimately gives us the following condition:

$$\frac{2N}{\log(N+1) - 1 + \alpha} > L \tag{6}$$

Considering Human Models. Again, when the special properties of human genomes are being considered, we need to re-assess the matrix space $\|S\|$ based upon a human genetic model, as described in Section 3.3. In our research, we ran the approximation algorithm (Section 3.3) to identify L and N that satisfy the above conditions (multiple sets of signs with a large distance), using the WTCCC1 data.

Figure 9 shows the result of the experiment involving 100 SNPs. As we can observe from the figure, in absence of a human model, a population with more than 250 individuals (500 sequences) are required to make sure that no more than 60% of signs can be identified. If we consider the human features, we need a population of at least 600 individuals ($N > 1200$).

4.3 When Not to Release

When the space of matrices S comes close to that of the r-squares, the adversary knows that once she acquires a set of valid r values, they are likely to be correct. Although we have shown that recovering signed r values from r-squares is NP-hard (Corollary 5), some instances of the sign recovery problem may be easy to solve, in particular when a human genetics model is employed to help solve the problem. Here we present a new attack technique that helps determine when this situation occurs, and thus a dataset should not be released. The new attack leverage on the LD structure of human genome and using haplotype recombination to efficiently recover the sign. For more detail, please read our technique report [55].

Evaluations. We ported the LD function, which is used in many GWAS papers for calculating MLE r^2, from the snp.plotter [12] package of R [9] to Matlab and implemented

[8] Note that the adversary has to consider the situation that all these elements (matrices) are associated with different r sets, as she has no computing power to estimate the relations between r and matrices.

the recombination attack using a stochastic hill climbing algorithm with multiple starting points. Then, we evaluated the attack on the data extracted from WTCCC1. We extracted 180 SNPs from chromosome 7. A case group and a reference group of 100 each were randomly sampled from the data-set. After that, the MLE-estimated r^2, together with single allele frequencies, was used as the optimization target for both inner block and inter block recombinations. On average, the sign agreement rate between the initial haplotype matrix (reference) and the target matrix (case) was 58.7%, which had very small power (identification rate 3.0% under a false positive rate 1%). After learning, the sign rate agreement was improved to 67.2% on average and the identification rate became 8.1%: that is, our approach enabled an adversary to identify about 8 participants from the aggregate data with a poor quality.

5 Related Work

The problem of releasing aggregate data while preserving their privacy has been extensively studied in privacy preserving data analysis [29,33], statistical disclosure control [18, 19, 32], inference control [24] and privacy-preserving data mining [14, 15]. However, the properties of human genome data make the problem special in this domain, which has not been well investigated. Especially, human individuals share about 99.9% genomic sequences, which makes it easy to find a reference group from public sources such as HapMap [6]. This enables both Homer's attack and the statistical attack proposed in [52], as elaborated in Section 2.2. Also remotely related to our research is the work on privacy preserving genome computing [16,41,22], which however does not focus on protecting the outcomes of a computation from being inferred.

The recent progress in human genome research [31,36] has made a great demand on convenient access to sensitive human genome data for research purpose. The problem of balancing privacy protection and data sharing in this domain, however, has not been seriously studied until Homer, et al. published their findings [39] a couple years ago. After that, several research groups, including us, have started working on this important issue [52, 46, 40, 51, 21]. As a prominent example, Sankararaman, et al [46] recently propose a technique (SecureGenome) for measuring the maximum statistical powers achievable on a set of single-allele frequencies. Most of these studies focus on single allele frequencies, which has been found in prior research to be insufficient [52], as sensitive information can also be inferred from other sources like test statistics. The research presented in this paper is the first attempt to understand and assess the risk in releasing different types of aggregate data, under typical inference threats.

Recovering SNP sequences is related to the research on contingency table release [20, 38, 23, 54, 27], and discrete tomography [37], which tries to reconstruct a matrix from a small number of projections. However, the specific problem of restoring a matrix from pair-wise allele counts is new, up to our knowledge, and the related complexity problems have not been studied before.

The Red-zone data identified by our techniques are not supposed to be released directly. However, they could still be published after proper sanitization and obfuscation. Such techniques have been studied in data-based privacy [17, 30, 19]. Particularly, the privacy policy based upon *Differential privacy* [29], once enforced, can make an identification impossible. Therefore, an important research direction is to develop effective techniques to achieve such a privacy objective on aggregate human genome data.

6 Conclusion

Availability of aggregate human DNA data is of great importance to human genome studies. Recent research shows that such data are vulnerable to different types of privacy threats, which could lead to identification of the participants of these studies and disclosure of their sensitive genetic markers. Therefore, a critical question becomes how to evaluate such a risk and determine when the data are safe to release. In our research, we make the first attempt to answer this question. We identified the problem space of aggregate data release, considering both different types of data available in the public domains (allele frequencies and test statistics) and common threats to such data (identification attack and recovery attack). Through a systematic exploration of the space, we gained an important new understanding of the problem. Specifically, we found that inferring useful information from such data is difficult in general: the adversary often does not have enough information and needs to solve NP-complete or NP-hard problems. On the other hand, we also show that an attack can still happen under some circumstances, particularly when the solution space of the problem is small. Based upon such an understanding, we propose a new risk-scale system that determines when data can be safely released, through analyzing their solution spaces.

Given the scale and the depth of this data-release problem, many open issues remain in the problem space. Particularly, a critical issue here is how to narrow the range of the Yellow zone, to get tighter bounds for releasing or not releasing an aggregate dataset. Also important is the study on new anonymization techniques that obfuscate the Red-zone data to achieve differential privacy without substantially compromising their scientific value.

References

1. Haplotype Estimation and Association (2005), http://slack.ser.man.ac.uk/theory/association_hap.html
2. NIH Background Fact Sheet on GWAS Policy Update (2008), http://grants.nih.gov/grants/gwas/background_fact_sheet_20080828.pdf
3. fastPHASE (2010), http://stephenslab.uchicago.edu/software.html
4. Genome-Wide Association Studies (2010), http://grants.nih.gov/grants/gwas/
5. Ibm ilog cplex optimizer (2010), http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
6. International HapMap Project (2010), http://www.hapmap.org/
7. National Institutes of Health (2010), http://www.nih.gov/
8. Policy for sharing of data obtained in nih supported or conducted genome-wide association studies (gwas) (2010), http://grants.nih.gov/grants/guide/notice-files/not-od-07-088.html
9. The R project for statistical computing (2010), http://www.r-project.org/
10. Re-identification and its discontents (2010), http://www.genomicslawreport.com/index.php/2009/10/13/re-identification-and-its-discontents/
11. SecureGenome (2010), http://securegenome.icsi.berkeley.edu/securegenome/
12. SNP.plotter (2010), http://cbdb.nimh.nih.gov/~kristin/snp.plotter.html

13. Wellcome Trust Case Control Consortium (WTCCC1) (2010), `https://www.wtccc.org.uk/ccc1/`
14. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: PODS 2001: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 247–255. ACM, New York (2001)
15. Agrawal, R., Srikant, R.: Privacy-preserving data mining. SIGMOD Rec. 29(2), 439–450 (2000)
16. Atallah, M.J., Kerschbaum, F., Du, W.: Secure and private sequence comparisons. In: WPES 20: Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, pp. 39–44. ACM, New York (2003)
17. Barak, B., Chaudhuri, K., Dwork, C., Kale, S., McSherry, F., Talwar, K.: Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In: PODS 2007: Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 273–282. ACM, New York (2007)
18. Beck, L.L.: A security machanism for statistical database. ACM Trans. Database Syst. 5(3), 316–3338 (1980)
19. Blum, A., Dwork, C., McSherry, F., Nissim, K.: Practical privacy: the sulq framework. In: PODS 2005: Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 128–138. ACM, New York (2005)
20. Bonferroni, C.E.: Teoria statistica delle classi e calcolo delle probability. Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze 8(1), 3–62 (1936)
21. Braun, R., Rowe, W., Schaefer, C., Zhang, J., Buetow, K.: Needles in the haystack: Identifying individuals present in pooled genomic data. PLoS Genet 5(10), e1000668 (2009)
22. Bruekers, F., Katzenbeisser, S., Kursawe, K., Tuyls, P.: Privacy-preserving matching of dna profiles. Technical Report Report 2008/203, ACR Cryptology ePrint Archive (2008)
23. Chen, Y., Diaconis, P., Holmes, S.P., Liu, J.S.: Sequential monte carlo methods for statistical analysis of tables. Journal of the American Statistical Association 100, 109–120 (2003)
24. Chin, F.Y., Ozsoyoglu, G.: Auditing and inference control in statistical databases. IEEE Trans. Softw. Eng. 8(6), 574–582 (1982)
25. Chiò, A., Schymick, J.C., et al.: A two-stage genome-wide association study of sporadic amyotrophic lateral sclerosis. Hum. Mol. Genet. 18(8), 1524–1532 (2009)
26. Chvatal, V.: Recognizing intersection patterns. In: Combinatorics 79, Part I, pp. 249–251. North-Holland Publishing Company, Amsterdam (1980)
27. Dobra, A., Fienberg, S.E.: Bounds for cell entries in contingency tables induced by fixed marginal totals. Statistical Journal of the United Nations ECE 18, 363–371 (2001)
28. Duerr, R.H.H., et al.: A genome-wide association study identifies il23r as an inflammatory bowel disease gene. Science (October 2006)
29. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
30. 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)
31. Edwards, A.O., Ritter, R., et al.: Complement factor H polymorphism and age-related macular degeneration. Science 308(5720), 421–424 (2005)
32. Fienberg, S.E.: Datamining and disclosure limitation for categorical statistical databases. In: Proceedings of Workshop on Privacy and Security Aspects of Data Mining, Fourth IEEE International Conference on Data Mining (ICDM 2004), pp. 1–12. Nova Science Publishing, Bombay (2004)
33. Gehrke, J.: Models and methods for privacy-preserving data analysis and publishing. In: ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering, p. 105. IEEE Computer Society, Washington, DC, USA (2006)

34. Goldreich, O., Vadhan, S.: Special issue on worst-case versus average-case complexity editors' foreword. Comput. Complex. 16, 325–330 (2007)
35. Greenspan, G., Geiger, D.: Modeling haplotype block variation using markov chains. Genetics 172(4), 2583–2599 (2006)
36. Haines, J.L., et al.: Complement factor H variant increases the risk of age-related macular degeneration. Science 308(5720), 419–421 (2005)
37. Herman, G.T., Kuba, A.: Advances in Discrete Tomography and Its Applications (Applied and Numerical Harmonic Analysis). Birkhauser, Basel (2007)
38. Hoeffding, W.: Scale-invariant correlation theory. Masstabinvariante Korrelationstheorie, Schriften des Matematischen Instituts und des Instituts fr Angewandte Mathematik der University 5, 179–233 (1940)
39. Homer, N., et al.: Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. PLoS Genet. 4(8), e1000167+ (2008)
40. Jacobs, K.B., et al.: A new statistic and its power to infer membership in a genome-wide association study using genotype frequencies. Nature Genetics 41(11), 1253–1257 (2009)
41. Jha, S., Kruger, L., Shmatikov, V.: Towards practical privacy for genomic computation. In: 2008 IEEE Symposium on Security and Privacy (2008)
42. Kim, Y., Feng, S., Zeng, Z.B.: Measuring and partitioning the high-order linkage disequilibrium by multiple order markov chains. Genet. Epidemiol. 32(4), 301–312 (2008)
43. Morris, A.P., Whittaker, J.C., Balding, D.J.: Little loss of information due to unknown phase for fine-scale linkage-disequilibrium mapping with single-nucleotide-polymorphism genotype data. Am. J. Hum. Genet. 74(5), 945–953 (2004)
44. Renström, F., et al.: Replication and extension of genome-wide association study results for obesity in 4,923 adults from northern sweden. Hum. Mol. Genet. (January 2009)
45. Robbins, R.: Some applications of mathematics to breeding problems iii. Genetics 3(4), 375–389 (1918)
46. Sankararaman, S., Obozinski, G., Jordan, M.I., Halperin, E.: Genomic privacy and limits of individual detection in a pool. Nat. Genet. 41(9), 965–967 (2009)
47. Scott, L., et al.: A genome-wide association study of type 2 diabetes in finns detects multiple susceptibility variants. Science (April 2007)
48. Sladek, R., et al.: A genome-wide association study identifies novel risk loci for type 2 diabetes. Nature (February 2007)
49. Stephens, M., Donnelly, P.: A comparison of bayesian methods for haplotype reconstruction from population genotype data. American Journal of Human Genetics 73(5), 1162–1169 (2003)
50. Stephens, M., Smith, N., Donnelly, P.: A new statistical method for haplotype reconstruction from population data. The American Journal of Human Genetics 68(4), 978–989 (2001)
51. Visscher, P.M., Hill, W.G.: The limits of individual identification from sample allele frequencies: Theory and statistical analysis. PLoS Genet. 5(10), e1000628 (2009)
52. Wang, R., Li, Y.F., Wang, X., Tang, H., Zhou, X.: Learning your identity and disease from research papers: Information leaks in genome wide association study. In: CCS 2009: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 534–544. ACM, New York (2009)
53. Yeager, M., et al.: Genome-wide association study of prostate cancer identifies a second risk locus at 8q24. Nature Genetics 39(5), 645–649 (2007)
54. Yuguo Chen, I.H.D., Sullivant, S.: Sequential importance sampling for multiway tables. The Annals of Statistics 34(1), 523–545 (2006)
55. Zhou, X., Peng, B., Li, Y.F., Chen, Y., Tang, H., Wang, X.: Technical report tr696: To release or not to release: Evaluating information leaks in aggregate human-genome data (2011), https://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR696

A Terminologies

Table 1. HGS Terminologies used in this paper

Terminologies	**Description**
Polymorphism	The occurrence of two or more genetic forms (e.g. alleles of SNPs) among individuals in the population of a species.
Single Nucleotide Polymorphism (SNP)	The smallest possible polymorphism, which involves two types of nucleotides out of four (A, T, C, G) at a single nucleotide site in the genome.
Allele	One of the two sets of DNAs in a human individual's genome. Most SNP sites have two common alleles in the human population: the major allele (denoted by 0), the one with higher frequency, and the minor allele (denoted by 1), the one with lower frequency.
Genotype	The combination of two set of alleles in a human individual. For a SNP site with two common alleles, there are three possible genotypes: two homozygotes, 00 and 11, and one heterozygote 01.
Locus(plural loci)	The surrounding regions of a SNP site in the genome .
Haplotype	Haplotype, also referred to as SNP sequence, is the specific combination of alleles across multiple neighboring SNP sites in a locus. Each individual has two haplotypes, each inherited from one parent. Some haplotypes are more common than others in the population.
Linkage disequilibrium(LD)	Non-random association of alleles among multiple neighboring SNP sites.

B Proofs of Theorem 1, Corollary 1, Corollary 2 and Corollary 4 Conjecture 1, Theorem 2, Corollary 5, Corollary 6

Proof of Theorem 1. This problem can be formalized as an existence problem EP(C_i, C_{ij}) which is to determine whether an $N \times L$ binary matrix M exists that satisfies the constraints of the sets of single allele counts $\{C_{i\in[1,L]}\}$ (e.g. the number of 0s in column i) and pairwise major allele counts $\{C_{ij\in[1,L]}\}$ (e.g. the number of 00 pairs of column i and column j). NOTE that these two sets are equivalent to the set of pairwise allele frequencies and may be used interchangeably in this paper. e.g. $C_i = 3$, termed as 3-EP(C_i, C_{ij}). Consider a special case of EP(C_i, C_{ij}), denoted as 3-EP, where all given single allele counts are 3 ($C_i = 3$). We prove 3-EP is NP-complete by reducing the 3-*Recognizing Intersection Patterns Problem*(3-RIPP(A)), a known NP-complete problem [26] to it. 3-RIPP(A) is described as: given $A = [a_{ij}]_{L\times L}$ in which $a_{ii} = 3$, is there an integer set collection $H = \{H_1, H_2, \cdots, H_L\}$ such that $a_{ij} = |H_i \cap H_j|$ for $1 \leq i, j \leq L$. Obviously, 3-EP $\in$ NP. Given an arbitrary instance of 3-RIPP(A), we construct an instance of 3-EP(C_i, C_{ij}) by setting $C_{ij} = a_{ij}$ for $1 \leq i \neq j \leq L$ and setting $C_i = a_{ii}$ for $1 \leq i \leq L$. Suppose $M_{N\times L}$ is a solution of EP. We can convert each column of $M_{N\times L}$ into a set, where the row indices of 1s in the i-th column form the elements in the set H_i. Therefore, We get $|H_i \cap H_j| = a_{ij} = C_{ij}$ for $1 \leq i, j \leq L, i \neq j$ and $|H_i \cap H_i| = a_{ii} = C_i = 3$. So $\{H_i\}$ represent a solution of 3-RIPP(A). Conversely, suppose $H = \{H_1, H_2, \cdots, H_L\}$ is a solution of 3-RIPP(A). We can construct a solution M of 3-EP by converting each set H_i into a column of length L where for each element $k \in H_i$, fill in the k-th position by 1 in the i-th column of $M_{N\times L}$, and all the other positions by 0. Clearly the resulting matrix $M_{N\times L}$ is consistent with (C_i, C_{ij}), and thus is a solution of 3-EP. Because the conversions described above can be done in polynomial time, 3-EP(C_i, C_{ij}) is NP-complete. Therefore, EP(C_i, C_{ij}) is also NP-complete since its special case 3-EP(C_i, C_{ij}) is NP-complete.

The rest of the proof is given in our technique report [55].

Don't Reveal My Intension: Protecting User Privacy Using Declarative Preferences during Distributed Query Processing

Nicholas L. Farnan[1], Adam J. Lee[1], Panos K. Chrysanthis[1], and Ting Yu[2]

[1] Department of Computer Science, University of Pittsburgh
[2] Department of Computer Science, North Carolina State University

Abstract. In a centralized setting, the declarative nature of SQL is a major strength: a user can simply describe *what* she wants to retrieve, and need not worry about *how* the resulting query plan is actually generated and executed. However, in a decentralized setting, two query plans that produce the same result might actually reveal vastly different information about the *intensional description* of a user's query to the servers participating its evaluation. In cases where a user considers portions of her query to be sensitive, this is clearly problematic. In this paper, we address the specification and enforcement of *querier* privacy constraints on the execution of distributed database queries. We formalize a notion of intensional query privacy called (I, A)-privacy, and extend the syntax of SQL to allow users to enforce strict (I, A)-privacy constraints or partially ordered privacy/performance preferences over the execution of their queries.

1 Introduction

Applications are increasingly becoming more decentralized, relying on data exchange between autonomous and distributed data stores. This reliance is in some cases a design decision motivated by the need for scalability; for example, when user demand for a service exceeds what a system at a single site would be able to provide and replication becomes a necessity. In other instances, it is a fact of life that must be dealt with. Such is the case when multiple partnering corporate entities wish to share data with one another, or when developers create "mashups" leveraging data gathered from multiple sources as a value-added service.

To date, the declarative nature of SQL has been one of its major strengths: a user can simply describe *what* her queries should retrieve and she need not worry about *how* they are converted into query execution plans and materialized. This has given rise to a rich literature concerning query optimization techniques that allow the system to explore relationally-equivalent representations of a user's query with the end goal of finding the *best possible* query execution plan (e.g., [14,10]). Traditionally, the term *best* has been defined in reference to minimizing some combination of overall query execution time or data transmission requirements [14]. Unfortunately, optimizing purely for query performance can

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 628–647, 2011.

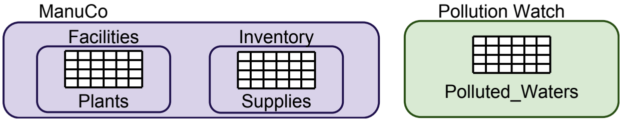


Fig. 1. Our example DB system. Three servers (`Facilities`, `Inventory`, and `Pollution Watch`), each store a single table (`Plants`, `Supplies`, and `Polluted_Waters`).

violate end-user privacy in a decentralized setting. Specifically, two query plans that produce the same result might actually reveal vastly different information about the intensional description of a user's query to the distributed servers participating in its evaluation.

Example 1. Consider Alice, a low ranking corporate executive, who wishes to investigate possible illegal pollution by her employer ManuCo. She could potentially do so by joining information stored in her employer's databases (e.g., records describing hazardous chemicals that are stored in the `Supplies` table on ManuCo's `Inventory` database server, and the locations of manufacturing plants stored in the `Plants` table on ManuCo's `Facilities` server, as in Fig. 1) with relevant information maintained by an environmental watchdog group (e.g., a database of polluted waterways). In issuing such a query, Alice would not want either party to become aware of the portion of her query that was issued to the other, or even that some part of her query was being evaluated by the other. Such a revelation could easily cost her her job, either because her employer felt that she "knew too much," or because the watchdog group applied external pressure to the company after learning of the intension of her query. ■

This type of scenario clearly points to the need to protect end-user privacy during distributed query evaluation. In this paper, we strive to protect the privacy of a user *querying* a set of distributed databases. This is in contrast to most existing work on user privacy in database systems, which has focused on the privacy of users whose data is *contained* in a database [5,16,17,22].

We believe that protecting the intension of user queries can be accomplished by modifying distributed query optimizers to optimize for query privacy in addition to query performance. We posit that through careful site selection and the use of a wide variety of query evaluation techniques (such as private information retrieval (PIR) [3,15,20]), query optimizers can be made to produce plans that effectively balance users' need for privacy with their desire for performance. In this paper, we work towards this goal by developing a formal framework for representing user preferences for privacy and performance on top of which such query optimizers can be built. As end-user privacy is an inherently personal and context-dependent notion that undoubtedly varies from user to user and query to query, our proposed approach allows users to express hard privacy constraints, establish preference relations between collections of possibly-competing privacy constraints, and explicitly balance the often competing interests of privacy-preservation and query efficiency. *It is our hypothesis that such a declarative model for preferences is a natural extension to declarative query* *languages such as SQL, and that a well defined semantics for balancing privacy and execution cost can be achieved using partially-ordered preference structures.* By investigating this hypothesis, this paper makes the following contributions:

- We develop a formal definition of intension-based user privacy called (I, A)-privacy, which allows users to specify that some subset of the intension of their query I (which we will refer to as an "intensional region") is kept hidden from some set of adversarial principals A. (Sect. 3)
- We propose a syntax for augmenting SQL with the capability of expressing partially-ordered (I, A)-privacy constraints, which future *privacy-aware* distributed query optmizers can utilize to optimize for both privacy and performance. We further demonstrate that (I, A)-privacy is flexible enough to express PIR privacy constraints. (Sects. 4 and 5)
- We develop a preference algebra capable of prioritizing and balancing competing (I, A)-privacy constraints, thereby allowing users to build complex privacy profiles that can be applied to their future queries. This preference algebra also allows users to explicitly balance the often competing goals of efficient query processing and intensional privacy. (Sect. 5)

It should be noted that the specific notion of (I, A)-privacy explored in this paper is syntactic, in that we define (I, A)-privacy to be violated only if some sensitive piece of query intension is directly revealed to another principal. However, the definition of (I, A)-privacy presented is sufficiently general to support more semantic notions of end user privacy. Such a semantic definition for the underpinnings of (I, A)-privacy is the subject of future work.

In the next section, we describe our system and threat model. In Sects. 3-6, we present our contributions. We overview related work in Sect. 6, and finally, discuss future work in Sect. 7.

2 Background and Assumptions

Here we will give a brief overview of query processing and relational algebra, and then present the assumptions we make for this work.

Query Processing and Relational Algebra. The basic processing of a user-input query involves the following steps: *parsing*, *reorganization*, *optimization*, *code generation*, and *plan execution*. The user query is first *parsed* based on whatever input query language was used (in this work we deal exclusively with SQL) and transformed to a representation that the query processor can operate on directly. In most query processing systems (and further what we will consider for the purposes of this work), this representation is a tree of *relational algebra* operators leafed by read operations on the source database relations for the query.

Although databases must deal with bags of tuples, relational algebra formally operates on sets of tuples. Relational algebra can be defined in terms of six primitive operators: *selection*, which returns only tuples from the input relation that match some given selection criteria; *projection*, which reduces the arity of the tuples it processes by eliminating unwanted attributes; *Cartesian product*, which returns all possible combinations of tuples from two input relations; *rename*, which changes the labels of the components of the tuples it processes; *set union*; and *set difference*. One notable operator that can be defined in terms these primitives operators is *join*. *Join* combines two input relations using selection criteria, as opposed to *Cartesian product* which does so exhaustively.

Once the query processor has converted the user query to an internal representation, the query is *reorganized* and *optimized* according to available meta-data to ensure its efficient evaluation by the database engine(s). After being sufficiently optimized, the query is transformed yet again to a representation that can be evaluated by the database engine through *code generation*. Finally, the result of the query is realized through *plan execution*. For a more in-depth review of query processing and relational algebra, we refer the reader to [6].

System Model. We tackle the problem of protecting the privacy of users who issue queries against a distributed system of autonomous relational database servers. In this paper, we assume the distributed system to be comprised of a countable set $\mathcal{P}$ of principals (i.e., servers and users) cooperating to process distributed queries. Each principal participating in the system as a *server* may host its own data tables locally, as well as export views defined in terms of queries over its own data tables, or tables/views hosted at other servers. When a server exports a view, it is not required to also export the definition of this view, which it may consider to be private [11]. Further, data tables may be replicated across multiple servers. Hence, in addition to exchanging meta-data (e.g., exported schemas and statistics) for query planning and optimization, servers inter-operate to maintain consistency across any replicated data tables.

Users are the subset of principals that issue queries within the system. A user is not assumed to have complete trust in every server that may see some part of her query, but is assumed to have (at least) one trusted server responsible for planning and executing queries on her behalf. Trivially, this trusted server—which we refer to as a *querier*—might be operated by the user herself on her local machine. Upon receiving an SQL query from a user, the querier uses its locally-available meta-data to generate an execution query plan derived from the user's query. It then schedules the execution of the query plan by decomposing it into sets of sub-plans by projecting on the servers selected to execute each operation in the query plan, while also making explicit the communication flow of tuples between sub-plans materialized by different servers. Finally, the querier dispatches the sub-plans to their corresponding servers, materializes its part of the query, and prepares the final result to return to the user. Note that a participating server may further expand and optimize their sub-plans, e.g., due to the expansion of remote views. As such, query evaluation is a recursive process that may traverse many servers.

Threat Model. We assume that each server controls access to the data stored within its tables and views using Role-Based Access Controls [9] defined at the table or view level, as implemented by most commercial database systems [24].

We further assume that any pair of principals $p_1, p_2 \in \mathcal{P}$ have the ability to set up a private and authenticated channel (e.g., a TLS tunnel [4]) to protect against eavesdropping and message modification, reordering, and replay attacks. We assume that all servers in the system (aside from the querier) to be *honest-but-curious* passive adversaries. Specifically, each server will correctly execute the query evaluation protocol, but may try to infer information regarding the user's original query from the sub-plan that it received. Furthermore, a set of colluding servers may work together to make inferences based upon their combined collection of sub-plans.

The honest-but-curious adversary model makes sense within the context of our work, given that we focus on preserving user privacy and not on ensuring the correctness of query results. Finally, although we assume a colluding adversary model, we do not consider the notion of a global passive adversary. That is, all information used by colluding servers to uncover querier intension must be gathered exclusively from the execution of distributed queries.

3 Querier Privacy

The vast majority of research in database privacy deals with preventing the disclosure or inference of sensitive tuples stored in database systems. By contrast, our focus lies in the protection of end-user privacy during the optimization and execution of distributed queries. In this section, we develop the notion of (I, A)-privacy, in which intensional regions of a user's query that the user considers to be sensitive are protected from disclosure to a set of colluding adversaries. While query intension is generally represented using a declarative language like SQL (and, indeed, knowledge of SQL is all that users would require in order to author protections over the privacy of such intension through the use of our model), we focus in this paper on how this knowledge is encoded in the actual query plan used during the distributed query evaluation process, as this is the information that would actually gleaned by adversaries according to our threat model.

Definition 1 (Query Plan). *A* query plan $Q = \langle N, E \rangle$ *is a directed, acyclic, and fully-connected graph with a single root where* $N \subseteq \mathcal{N}$ *and* $E \subseteq N \times N$. *An element* n *of the node set* $\mathcal{N}$ *is a ternary* $n = \langle op, params, p \rangle$ *that describes a relational operator (op), the parameters to this operator (params), and the principal at which this operator is scheduled to be executed (p). The edge set* E *describes the producer/consumer relation between relational operators.*

Definition 2 (Well-Formed Query Plan). *A query plan* Q *is* well-formed *iff (i)* Q *corresponds to a valid relational algebra expression and (ii) for each node* $n = \langle op, params, p \rangle \in Q$, *the principal* p *is capable of both executing the operation described by* n *and transmitting the result of that operation to the principal(s) annotated to execute the parent node(s) of* n *in the query plan* Q.

Assuming that $\mathcal{S}$ denotes the set of all SQL queries, we can then formally represent a query planner as a function $\mathsf{plan} : S \rightarrow \mathcal{Q}$ that generates query plan

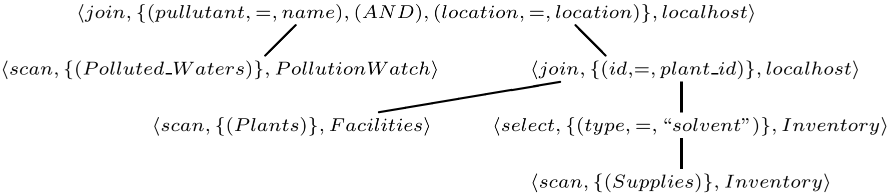

$\langle join, \{(pullutant, =, name), (AND), (location, =, location)\}, localhost\rangle$
$\langle scan, \{(Polluted\_Waters)\}, PollutionWatch\rangle$
$\langle join, \{(id,=, plant\_id)\}, localhost\rangle$
$\langle scan, \{(Plants)\}, Facilities\rangle$
$\langle select, \{(type, =, \text{“}solvent\text{”})\}, Inventory\rangle$
$\langle scan, \{(Supplies)\}, Inventory\rangle$

Fig. 2. Query plan for Alice's query. The text in each node is a comma-seperated list of the operation type of that node, the arguments to that operation, and the execution location of that node.

$Q \in \mathcal{Q}$ from an SQL query $q \in \mathcal{S}$. Fig. 2 is a graphical representation of a well-formed query plan corresponding to Alice's query from our motivating example in Sect. 1. We now look at two specific types of query plans:

Definition 3 (Locally-Expanded Query Plan). *A well-formed query plan Q is said to be* locally-expanded *with respect to a principal $p \in \mathcal{P}$ if every leaf node ℓ of Q represents either (i) an operation annotated for execution at some remote principal $p' \neq p$, or (ii) the scan of a table managed locally by the principal p. We denote the set of all locally-expanded query plans for a query $q \in \mathcal{S}$ as $\mathscr{L}(q)$.*

Note that the term *scan* is used to denote the retrieval of tuples from a database relation without specifying a specific access method (e.g., the use of a particular index). Given a query $q \in S$, the output of a query planner plan at a specific principal p is a locally-expanded query plan $Q \in \mathscr{L}(q)$ with respect to the principal p. The annotated leaves of the resulting query plan may be further expanded by other nodes in the network during query processing (e.g., during the expansion of a remote view). This leads to the following definition:

Definition 4 (Globally-Expanded Query Plan). *A well-formed query plan Q is said to be* globally-expanded *if every leaf node ℓ of Q represents the scan of a relational table managed by some principal $p \in \mathcal{P}$. We denote the set of all globally-expanded query plans for a query $q \in \mathcal{S}$ as $\mathscr{G}(q)$.*

Note that a globally-expanded query plan is trivially a locally-expanded query plan relative to every principal $p \in \mathcal{P}$. That is, no principal can further expand a globally-expanded query plan. Furthermore, given some initial query, a corresponding globally-expanded query plan may not ever be learned by any one principal in the system: each principal generates a locally-expanded query plan, but may perhaps be unaware of how its plan is further expanded by others.

In general, a user may only consider certain parts of her query to be sensitive. In our example query, Alice may not mind if people know that she is interested in data from the Facilities server's Supplies table, but she would certainly want the fact that it will be combined with data from Pollution Watch to be kept private. To formalize this notion, we introduce the following term:

Definition 5 (Intensional Region). *An* intensional region *is a countable subset I of the node set $\mathcal{N}$.*

Example 2. To illustrate this concept, we can refer back to Alice's query plan from Fig. 2. Consider the case in which Alice wants to keep the selection criteria (i.e., solvents) used over the `Supplies` table private from all principals involved in the processing of her query. An intensional region which contains the single select node in her query plan could then be defined to represent the specific portion of the intension of Alice's query that she wishes to protect. Similarly, if Alice wanted all parts of her query in which data from different relations is joined to be kept private, an intensional region containing all join nodes could be defined. ■

The above definition of intensional region is also flexible enough to identify broader descriptions of intension considered to be sensitive like "all selection operations that may be executed at the human resources server." Given such intensional regions, we can informally say that a query plan Q maintains a user's privacy if no adversarial principal can learn the intensional regions identified by the user as sensitive. To continue to reason about this notion of user privacy, however, we must first define how query intension is revealed to principals in the system during the evaluation of a user's query.

Definition 6 (Intensional Knowledge). *Given a globally-expanded query plan $Q = \langle N, E \rangle$, we denote by $\kappa_p(Q) \subseteq N \cup E$ the* intensional knowledge *that principal $p \in \mathcal{P}$ has of the query encoded by the plan Q. At a minimum, $\kappa_p(Q)$ contains the set of all locally-expanded query plans for each node $n \in N$ annotated for execution by the principal p, and further all edges leaving or entering such nodes.*

Given a colluding set of principals $P = \{p_1, \ldots, p_k\}$, we can define the combined intensional knowledge of the colluding principals P in the natural way, i.e., $\kappa_P(Q) = \bigcup_{p_i \in P} \kappa_{p_i}(Q)$. For example, the intensional knowledge of the three other principals participating in the evaluation of the query in Fig. 2 can be broken down as follows:

$$\begin{array}{ll}
\kappa_{PollutionWatch}(Q) & \langle \texttt{scan}, \{(\texttt{Polluted\_Waters})\}, \texttt{Pollution Watch} \rangle \\
\kappa_{Facilities}(Q) & \langle \texttt{scan}, \{(\texttt{Plants})\}, \texttt{Facilities} \rangle \\
\kappa_{Inventory}(Q) & \langle \texttt{select}, \{(\texttt{type}, \texttt{=}, \text{``solvent''})\}, \texttt{Inventory} \rangle \\
 & \text{and } \langle \texttt{scan}, \{(\texttt{Supplies})\}, \texttt{Inventory} \rangle
\end{array}$$

The above definitions enable us to precisely define the notion of privacy that we will explore in the remainder of this paper as follows:

Definition 7 ((I, A)-privacy). *Given an intensional region I and a set of colluding adversaries $A \subseteq \mathcal{P}$, a globally-expanded query plan Q is said to be (I, A)-*private *iff $\kappa_A(Q) \not\models I$, where $\models$ denotes an inference procedure for extracting intensional knowledge from a collection of query plans.*

In the above definition, there are many possible candidates for the $\models$ relation. For the purposes of this paper, we will focus our attention on the syntactic condition of the containment relation $\sqsupseteq$ defined below. This relation denotes whether an intensional region explicitly overlaps some collection of adversarial knowledge. We will be exploring a semantic relation for $\models$ as a subject of future work.

Definition 8 ($\sqsupseteq$). *Let Q be a globally-expanded query plan, I be an intensional region, A be a set of colluding adversaries, and $\kappa_A(Q) = (N, E)$ be the intensional knowledge that the adversary set A has about the query plan Q. Then $(N, E) \sqsupseteq I$ iff $N \cap I \neq \emptyset$.*

We feel that using the $\sqsupseteq$ relation as our inference procedure $\models$ is a natural first step for exploring end-user privacy as it is expressive enough to encode private information retrieval (PIR) constraints. This claim will be explored in Sect. 4.2, and a formal proof of it is presented in [7].

In the next section, we will present extensions to SQL that allow users to concisely identify the intensional regions considered sensitive within queries that they may issue. Sect. 5 then shows how privacy requirements over these regions can be combined using partially-ordered preference structures to develop a flexible formal foundation for providing (I, A)-privacy in distributed database systems.

4 Matching Query Plan Nodes

In this section, we will first illustrate how users can use the node matching in order to specify constraints on the evaluation of a query. We then formalize our method for identifying intensional regions through the use of descriptors of nodes in N, and a matching operator, $\pitchfork$.

4.1 Matching Syntax

In order for users to be able to express their notions of privacy to a query planner when issuing a query, we propose a `REQUIRING` clause (`REQUIRING` ... `HOLDS OVER` ...) as an extension to SQL. We are proposing this clause for use in two locations, first as an addition to the SQL `SELECT` statement, and also as an addition to SQL's set operators (`UNION`, `INTERSECT`, and `MINUS`). In the case that it is used in conjunction with a set operator, the requirements expressed by this clause will apply to both the operator itself, and the two `SELECT` statements that it combines. If it is used at the end of a `SELECT` statement, its requirements apply only to that `SELECT` statement. The full syntax for the `REQUIRING` clause is defined in Appendix A.

This clause allows a user to identify one or many intensional regions that match against *node descriptors* in a *match statement*, and further specify *constraints* over how nodes in those intensional regions are treated over the course of executing a query. Node descriptors can be considered similar to regular expressions in that they specify a pattern to be matched against nodes. They are made up of descriptors of the same three attributes present in query plan nodes in $\mathcal{N}$: *op*, *params*, and *p*. They are differentiated from query plan nodes in that the values of these attributes do not have to be grounded, they could instead be expressed as a wildcard (*) or free variable (which we prefix with a "\$"). Constraints can then be expressed as statements over the values of the free variables present in corresponding node descriptors. These constraints can be used to express (I, A)-privacy constraints by defining an intensional region with a node descriptor, and constraining all nodes matching that description such that they can not be executed by principals in A. Constraints can be tests for any of the following: equality ($=$), inequality ($\neq, <, >, \leq, \geq$), or set membership ($\in$).

Example 3. Given the ability to express clauses of the form shown in Appendix A as a part of her SQL queries, Alice could have written her query shown in Fig. 2 to ensure that any operation that she considers private is executed by her trusted server. The adversarial set in this example would hence consist of all principals in the system aside from the querier, denoted as `localhost`.

```
SELECT * FROM Plants, Supplies, Polluted_Waters
WHERE Supplies.type = "solvent",
    AND Supplies.name = Polluted_Waters.pollutant,
    AND Polluted_Waters.location = Plants.location,
    AND Plant.id = Supplies.plant_id
REQUIRING $p = localhost HOLDS OVER <*, {("solvent")}, $p>;
```

Such a query would uphold (I, A)-privacy by keeping an intensional region consisting of all nodes operating on the constant “solvent” from being disclosed to any remote database servers, as any nodes in that intensional region would be constrained to be executed at `localhost`. Note that the query plan shown in Fig. 2 does not adhere to this requirement, and would not be an acceptable query plan for this new query. ■

Examples of the expressive power of the `REQUIRING` clause are given in Appendix B by showing how it can be used to express common policy idioms.

4.2 Matching Operator

While we have presented the practical applications of node descriptors, we now formally define node descriptors:

Definition 9 (Node Descriptor). *A node descriptor $d \in \mathcal{D}$, is a triple $d = \langle op, params, p \rangle$ describing the relational operator (op), a set of parameters to the given operator ($params$), and the principal at which this operator shall be executed (p). A node descriptor is well formed if all of the following hold true:*

- *op is either a valid relational algebra operation, free variable, or $*$*
- *$params$ is either a set of sets (any of which may have a free variable) or $*$*
- *p is either the location of some principal in the system, a free variable or $*$*

We will consider $\mathcal{V}$ to be the set of all free variables that can be declared. To determine whether some node n matches a given descriptor, we define a matching operator ($\pitchfork$) as follows:

Definition 10 (Matching). *Given a node descriptor $d \in \mathcal{D}$, and a node $n \in N$, $d \pitchfork n$ if and only if*

$$(d.op = n.op \vee d.op \in \mathcal{V} \vee d.op = *) \;\wedge\; (d.p = n.p \vee d.p \in \mathcal{V} \vee d.p = *) \;\wedge$$
$$[\forall a \in d.params : (a \in \mathcal{V} \;\vee\; (a \in n.params \vee \exists a' \in n.params : a \subseteq a')) \;\vee d.params = *]$$

Hence, an intensional region I can be defined in terms of a node descriptor d as a subset of $\mathcal{N}$ as: $I = \{n \mid n \in \mathcal{N} \wedge d \pitchfork n\}$.

While matching on *op* or *p* is rather intuitive, matching on *params* requires a bit of explanation. *params* is a set of sets in both nodes and node descriptors that represents the arguments to a relational algebra operator. To allow for easy and concise expression of node descriptors, we state that a node descriptor matches a node based on *params* if every ordered set in the descriptor's *params* is either contained directly in the node's *params* or is a subset of an ordered set in the node's *params*.

Example 4. Our definition of $\pitchfork$ allows nodes with complex *params* attributes, such as ⟨ select, { (`at1`, =, 42), (`AND`), (`at2`, <, 10) }, example_principal ⟩, to be easily matched. Any of the following node descriptors will match this node based on our definition of $\pitchfork$: ⟨ *, { (`at1`) }, * ⟩; ⟨ *, { (`at1`, =, 42) }, * ⟩; ⟨ *, { (`at1`, 42) }, * ⟩; ⟨ *, { (`AND`) }, * ⟩; ⟨ *, { (`at1`, 42), (`AND`) }, * ⟩; ⟨ *, { (`at1`), (`AND`), (`at2`) }, * ⟩ ■

As has been previously stated, privacy is an inherently personal property, and hence, we must ensure that our method of specifying node descriptors and matching nodes against them to define intensional regions is sufficiently general to allow users to define regions based on *any* part of the intension of their query.

Theorem 1. *For any SQL query $q \in \mathcal{S}$, it is possible to specify a node descriptor $d \in \mathcal{D}$ that identifies any clause of q and/or its components (i.e., table or view names, constraints, constraint operators, attributes, or constants). Then, for any query plan $Q = \langle N, E\rangle$ that materializes q, the set of nodes $C = \{n \mid n \in N \wedge d \pitchfork n\}$ contains* exactly *those nodes corresponding the specified clause and/or components of q.*

The proof of this theorem is a case-by-case analysis showing that for any valid SQL operator with any valid set of arguments, there exists a node descriptor that matches the corresponding node in a query plan. This proof is presented in [7]. Furthermore, given that any single node within a query plan can be matched by some node descriptor, we immediately have the following:

Corollary 2. *Any intensional region $I \subseteq \mathcal{N}$ can be specified using a collection $D \subseteq \mathcal{D}$ of node descriptors, and detected within any query plan $Q \in \mathcal{Q}$ using the $\pitchfork$ operator.*

Since it is possible to specify a set of node descriptors $D \subseteq \mathcal{D}$ such that the matching construct $\pitchfork$ identifies the corresponding intensional regions contained within a query q, and constraints can be placed on the location field of any node matching a set of node descriptors D, we trivially have the following:

Corollary 3. *The matching operator $\pitchfork$ is sufficiently expressive to encode any (I, A)-privacy constraint.*

As should now be apparent, (I, A)-privacy is capable of expressing a wide variety of user privacy constraints. One set of such constraints that we would like to highlight (as alluded to in Sect. 3) is that offered by private information retrieval (PIR). The problem addressed by PIR is formulated as follows: given some k servers that store replicated copies of a database that is viewed as a length n binary string $x = x_1 \ldots x_n$, the goal of PIR is to allow a user to learn the value of some desired bit x_i without allowing any server to gain information about the value of i [3]. Initially, k was specified to be $k \geq 2$, however [15] details a technique for achieving PIR from a single computationally-bounded server.

In essence, PIR is a technique for retrieving some information from a database without revealing to the server(s) hosting that database the criteria for selecting items from that database (the indices of the bits that the querier is interested in). By constructing node descriptors that define an intensional region as all nodes which contain some part of the selection criteria of a query, users can constrain this intensional region to ensure that no database servers gain intensional knowledge of the selection criteria of their queries, and hence use (I, A)-privacy to express PIR privacy constraints on their queries. This notion is precisely stated in the following theorem, and a formal proof appears in [7].

Theorem 4. *Let the inference procedure $\models$ be defined using the containment relation $\sqsupseteq$. In this case, (I, A)-privacy can be used to express* any *private information retrieval (PIR) constraint.*

4.3 Constraining Multiple Node Descriptors

We allow free variables to be included in the specification of a node descriptor so that constraints can be placed on the values of those variables. Node descriptors have a dual-purpose in that they not only serve to identify nodes (through their grounded attributes), but also establish which attributes of the nodes in the intensional region that they identify can be constrained. In the case that a constraint is written over the free variables in a single node descriptor, ensuring that such a condition holds over a given query tree is relatively simple. For each node in the query tree that matches the node descriptor, ensure that the condition holds for the values of that node which correspond to the free variables in the node descriptor.

When a condition is specified over multiple node descriptors, however, we allow for two possibilities. Either all node descriptors use the same free variable, or different variables are used in different descriptors. In the case that the same variable is used, it must be ensured that the constraint holds for any node matching any of the descriptors. This is essentially a shortcut for writing multiple identical conditions for different node descriptors. In the case that different variables are used, however, ensuring that a condition holds over a query plan is slightly more complicated, as it must be ensured that for all combinations of nodes that match the independent descriptors, the condition holds. Examples such node descriptor/condition pairs are shown in Appendix B.

5 Preference Algebra

The syntactic and logical constructs defined in Sect. 4 are sufficient for upholding strict (I, A)-privacy requirements that users may define for their queries. However, users may need to consider the enforcement of many potentially-competing privacy constraints, or explicitly balance the desire for private query evaluation with the real-life performance implications of private query evaluation techniques. In this section, we develop a formal preference algebra that allows users to establish complex preference structures over the privacy preservation and performance characteristics of query plans generated from their SQL queries.

5.1 Background

In [13,12], the authors develop a formalism for expressing preferences over the tuples returned by a relational database query. Rather than requiring that an SQL selection specify an *exact match* criteria, the preference SQL described in [13,12] allows the user to specify a partially-ordered preference structure over the tuples returned. This is particularly helpful when exact match criteria cannot be found. Formally, the authors of [12] define a preference as follows:

Definition 11 (Preference $P = (R, <_P)$). *Given a set R of relational attribute names, a* preference P *is a strict partial order $P = (R, <_P)$, where $<_P \subseteq domain(R) \times domain(R)$.*

Given this definition, "$x <_P y$" is interpreted as "I like y better than x." For example, a user querying a database for the cheapest car could express her preference for tuples with the lowest value for the price attribute as: LOWEST(price), where LOWEST is a base preference defined such that $x <_P y$ *iff* x.price $>$ y.price and y.price is as low as possible. Using this base preference constructor, tuple t will be preferred to tuple t' *iff* t represents a lower cost car than tuple t'. We refer the reader to [12] for descriptions of a range of other numeric and non-numeric base preference constructors.

Similarly, [12] defines *complex preferences* through the use of complex preference constructors. For example, two preferences that are equally preferred can be combined through the use of a *Pareto* preference constructor. Given two preferences $P1$ and $P2$ over attributes $A1$ and $A2$, respectively, such a preference is defined for two items x and y containing attributes from both $A1$ and $A2$ as:

$$\begin{aligned} x <_{P1 \otimes P2} y \textit{ iff } & (x_1 <_{P1} y_1 \wedge (x_2 <_{P2} y_2 \vee x_2 = y_2)) \vee \\ & (x_2 <_{P2} y_2 \wedge (x_1 <_{P1} y_1 \vee x_1 = y_1)) \end{aligned}$$

Complex preferences in which one preference is strictly more important than the other can be defined using the *prioritized preference* operator &. The definitions of other complex preference constructors can be found in [12].

5.2 Preferences for Query Plan Execution

We now extend the above preference formalism to enable the expression of preferences over the (I, A)-privacy properties and performance characteristics of a query plan. To establish preferences over query plans—rather than relational tuples—we must redefine the notion of preferences to operate over a set of *query plan evaluation functions* $\mathcal{F}$. Functions within this set might, for instance, check a query plan's compliance with an (I, A)-privacy condition, evaluate the predicted runtime of a query plan, or estimate the amount of data that will need to be transmitted during the evaluation of a query plan. To allow preferences to be specified over (I, A)-privacy conditions, $\mathcal{F}$ contains at least the function $\mathsf{check} : \mathcal{Q} \times 2^{\mathcal{D}} \times \mathcal{C} \to \mathbb{B}$, where $2^{\mathcal{D}}$ is the power set of all node descriptors D. We define check as follows:

Definition 12 (check). *Let $Q = \langle N, E\rangle$ be a query plan, D be a set of node descriptors, c be a constraint over the free variables describing node locations in D, A be the set of principals not permitted to be assigned to a free variable in a node descriptor in D by the constraint c, and $I = \{n \mid n \in N \land \exists d \in D : d \pitchfork n\}$ be the intensional region of Q matched by the set D of node descriptors. Now, $\mathsf{check}(Q, D, c) \leftrightarrow \kappa_A(Q) \not\models I$.*

That is, check examines whether a given query plan upholds (I, A)-privacy as defined by a particular node descriptor, constraint, and inference operator $\models$.

Example 5. Consider the query plan shown in Fig. 2. If we refer to this query plan as Q, then check(Q, $\{\langle$ *, {("solvent")}, $\$p\rangle\}$, $\$p$ = `localhost`) evaluates to `false` (when using $\sqsubseteq$ for $\models$), as the `SELECT` statement is executed at the Inventory Server, not Alice's trusted server. Similarly, check(Q, $\{\langle$ join, *, $\$p\rangle\}$, $\$p$ = `localhost`) evaluates to `true`, since all joins happen on Alice's server. ■

In order to allow users to balance (I, A)-privacy preferences with the estimated performance of query, we can augment $\mathcal{F}$ with additional functions that estimate the performance characteristics of a query plan. For instance, we could include a function $\mathsf{runtime} : \mathcal{Q} \to \mathbb{R}$ that associates a query plan with its estimated runtime (e.g., in seconds). Given such a set $\mathcal{F}$ of query plan evaluation functions, we can formally define a query plan preference as follows:

Definition 13 (Preference $P = (\mathcal{F}, <_P)$). *Let $\mathcal{F}$ be a set of query plan evaluation functions, and let $V = \{range(f) \mid f \in \mathcal{F}\}$. A query plan* preference P *is a strict partial ordering $P = (\mathcal{F}, <_P)$ where $<_P \subseteq V \times V$.*

Preferences over quantitative evaluations of the *performance* of a query plan can be easily defined through the use of the base numerical preference constructors presented in [12] (e.g., LOWEST). However, the expression of preferences over (I, A)-privacy constraints requires a new base preference constructor, which we will call HOLD to mirror the `HOLDS OVER` keywords presented in Sect. 4.1.

Definition 14 ($\mathsf{HOLD}(Q, D, c)$). *Given a query plan Q, a set of node descriptors D, and a constraint c over free variables declared node descriptors in D, it is preferable for check(Q, D, c) to evaluate to `true` as opposed to `false`.*

Complex preferences over both privacy and performance can now be constructed using base numerical preference constructors and the above defined HOLD constructor in addition to the complex preference constructors presented in [12].

Example 6. Alice considers it of paramount importance for her query to run in the least amount of time, but also prefers that all join operations and any operations involving the constant "solvent" be executed by her trusted query processing software, which runs on `localhost`. She considers the latter two preference to be equally desirable, but less desirable than her runtime preference. This complex preference can be represented as follows:

$$\begin{aligned} &\mathit{LOWEST(runtime)}\ \&\ (\mathit{HOLD}(q, \langle *, \{(\textit{“solvent”})\},\ \$p\rangle, \$p = \mathit{localhost}) \\ &\qquad\qquad \otimes\ \mathit{HOLD}(q, \langle \mathit{join}, *, \$p\rangle, \$p = \mathit{localhost})) \end{aligned}$$

■

Since query plan preferences are defined over the range of *every* function in $\mathcal{F}$, complex preference can be expressed over base preference operators with differing types. This allows preferences to be established between multiple (I, A)-privacy constraints, between multiple performance constraints, or between a mix of (I, A)-privacy and performance constraints (as in the above example).

5.3 Preference Syntax

In order to enable users to make use of the query plan preference constructs defined in Sect. 5.2, we now describe another extension to SQL, the `PREFERRING` clause —modeled after the extensions proposed in [13]— that similarly applies to both `SELECT` statements and set operators. We maintain the same notation for preferences over the results of functions with numeric ranges in that such preferences are stated explicitly through the base preference constructor that is to be used and the name of the function that a preference is to be expressed over. We further maintain the use of the keyword `AND` to represent the complex preference constructor ⊗ and `CASCADE` to represent the complex preference constructor &. In the case of expressing privacy preferences over intensional regions, however, this clause will take a similar form to the `REQUIRING` clause presented in Sect. 4.1. The full syntax of the `PREFERRING` clause is presented in Appendix A. To demonstrate the use of this syntax for including query plan preferences within a query, we now consider the following example the combines strict requirements on a query plan with more flexible preferences:

Example 7. Assume that Alice *requires* that any nodes whose operations make use of the constant "solvent" must be annotated for execution at `localhost`. Further, she *prefers* to execute joins on `localhost` as well, given that they do not increase the execution time of her query. Alice could express this combination of preferences using both the `REQUIRING` and `PREFERRING` clauses:

```
SELECT * FROM Plants, Supplies, Polluted_Waters
WHERE Supplies.type = "solvent",
    AND Supplies.name = Polluted_Waters.pollutant,
    AND Polluted_Waters.location = Plants.location,
    AND Plant.id = Supplies.plant_id
REQUIRING $p = localhost HOLDS OVER <*, {("solvent")}, $p>
PREFERRING LOWEST{runtime}
    CASCADE $p = localhost HOLDS OVER <join, *, $p>;
```

■

The notion of query plan preferences can be used to express a wide range of privacy and performance constraints over the execution of user queries. We present examples of common policy idioms encoded using this approach in Appendix B.

5.4 Implementation Considerations

A practical method implementing an (I, A)-privacy aware query optimization would be the direct inclusion of our model in an existing query optimizer. While this work is not focused on the implementation of the model which we describe, here we briefly discuss how the changes that would have to be made to a query optimizer would affect its performance.

Such an implementation could be accomplished with little modification to existing dynamic programming based query optimizers by maintaining separate lists of plan costs for each set of plans that is equally preferred throughout the course of optimization. While this may result in an increase in query optimization time, it should be noted that such lists need only be maintained for query execution restrictions in `PREFERRING` clauses. Restrictions from `REQUIRING` clauses, on the other hand, could be utilized to speed up query optimization by pruning query plans from the search space that do not uphold them. Hence, users making use of only `REQUIRING` clause constraints can only lessen the time that is required to optimize their queries in the average case. Optimization time penalties are only incurred by users wishing to express complex controls over the intension of their queries. Given the expressive power that our model affords users, however, we feel it is quite reasonable to assume that in these cases they will be willing to accept this increased optimization time.

6 Related Work

We now discuss areas of closely related work: private information retrieval and distributed query processing.

Private Information Retrieval. The problem that PIR techniques work to solve was described in Sect. 4.2. PIR was originally proposed in [3] through an approach that required multiple non-colluding servers to host replicas of the database that a user wished to access. This approach has come to be known as information-theoretic PIR. PIR using only a single server was established using computational techniques in [15]. Though the practical feasibility of computational PIR has been called into question [23], such issues can be assuaged through the use of either secure co-processors [26], or general purpose computing on graphics processing units (GPGPU) [18]. Further, a method for performing information-theoretic PIR over widely implemented database access methods (hash indices and B^+ tree indices) was recently demonstrated [20].

Systems implementing (I, A)-privacy support would be able to utilize any of these techniques in the special case that (i) the user specifies privacy constraints on the execution of her query that can be achieved through the use of PIR and $(ii)$the database servers providing the required data support a practical technique for PIR. This requirement in and of itself also highlights an advantage of our work: it allows users to protect their privacy when interacting with database severs only rudimentary query processing capabilities (specifically those outlined in Sect. 2), while still providing the capability for users to take advantage of more advanced techniques such as PIR, when they are available and applicable.

Distributed Query Processing. Distributed query processing is typically performed by either shipping the data required for the query back to the site that issued the query for processing (data shipping), or shipping pieces of the query out to the sites holding the data for parallel processing, returning only the result to the issuing site (query shipping) [14]. These techniques can further be combined as a form of hybrid shipping [10]. Mutant query plans [21] can also provide a form of combined shipping that allows asynchronous query evaluation. In the same manner that our model proposed here is able to glean the advantages of PIR when possible, it can also utilize all of the these query processing techniques to construct query plans that sufficiently balance user privacy preferences with query performance, realizing the proposed hybrid query processor from [8].

7 Conclusions and Future Work

In this paper, we have formally defined a notion of intension-based query privacy called (I, A)-privacy. This type of privacy is designed to allow the user querying a database to express constraints on the portions of her intensional query that should not be leaked to the servers involved in executing her query. We have further shown that private information retrieval is a special case of (I, A)-privacy and can thus be used as a building block for systems seeking to preserve certain types of (I, A)-privacy. We have presented a framework for representing complex user preferences balancing query privacy and performance. Further, we developed a syntax for extending SQL to express such preferences.

Future work will first and foremost include the implementation of our framework within a distributed query optimizer. Future work will also investigate the expression of preferences not only over the dissemination of query plan metadata, but also the flow of extensional query results over the course of query execution. We will explore other possible relations for the $\models$ operator that could take into account both extensional flows of query results and a more powerful adversarial model, and further include semantic notions of end user privacy.

Acknowledgments. This research was supported in part by the National Science Foundation under awards CCF–0916015, CNS–0964295, CNS–1017229, CNS–0914946, CNS–0747247, and CDI OIA–1028162; and by the K. C. Wong Education Foundation.

References

1. Bell, D.E., Lapadula, L.J.: Secure computer system: unified exposition and multics interpretation (March 1976)
2. Botha, R.A., Eloff, J.H.P.: Separation of duties for access control enforcement in workflow environments. IBM Syst. J. 40, 666–682 (2001)
3. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: FOCS (1995)
4. Dierks, T., Rescorla, E.: Rfc 5246: The transport layer security (tls) protocol version 1.2 (August 2008)

5. 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)
6. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison-Wesley, Reading (2007)
7. Farnan, N.L., Lee, A.J., Chrysanthis, P.K., Yu, T.: Dont reveal my intension: Protecting user privacy using declarative preferences during distributed query processing. Technical Report TR-11-179, University of Pittsburgh, Dept. of Computer Science (2011)
8. Farnan, N.L., Lee, A.J., Yu, T.: Investigating privacy-aware distributed query evaluation. In: WPES (2010)
9. Ferraiolo, D., Kuhn, R.: Role-based access control. In: NIST-NCSC (1992)
10. Franklin, M.J., Jónsson, B.T., Kossmann, D.: Performance tradeoffs for client-server query processing. SIGMOD Rec. 25, 149–160 (1996)
11. Information technology - database language sql (1992)
12. Kießling, W.: Foundations of preferences in database systems. In: VLDB (2002)
13. Kießling, W., Köstler, G.: Preference sql: design, implementation, experiences. In: VLDB (2002)
14. Kossmann, D.: The state of the art in distributed query processing. ACM Comput. Surv. 32(4), 422–469 (2000)
15. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS (1997)
16. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE (2007)
17. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM TKDD 1(1), 3 (2007)
18. Melchor, C.A., Crespin, B., Gaborit, P., Jolivet, V., Rousseau, P.: High-speed private information retrieval computation on gpu. In: SECURWARE (2008)
19. National Computer Security Center (NCSC). Glossary of Computer Security Terms (ncsc-tg-04) (October 1988), `http://csrc.nist.gov/publications/secpubs/rainbow/tg004.txt`
20. Olumofin, F.G., Goldberg, I.: Privacy-preserving queries over relational databases. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 75–92. Springer, Heidelberg (2010)
21. Papadimos, V., Maier, D., Tufte, K.: Distributed query processing and catalogs for peer-to-peer systems. In: CIDR (2003)
22. Samarati, P.: Protecting respondents' identities in microdata release. IEEE TKDE 13, 1010–1027 (2001)
23. Sion, R., Carbunar, B.: On the practicality of private information retrieval. In: NDSS (2007)
24. Tran, S., Mohan, M.: Security information management challenges and solutions (July 2006), `http://www.ibm.com/developerworks/data/library/techarticle/dm-0607tran/index.html`
25. Wang, L., Wijesekera, D., Jajodia, S.: A logic-based framework for attribute based access control. In: FMSE (2004)
26. Williams, P., Sion, R.: Usable PIR. In: NDSS (2008)

A SQL Extension Syntax

It should be noted that in the following syntax we leave <literal> ungrounded. We use <literal> as a placeholder for strings of characters which could represent either names of free variables, constants (e.g., “solvent”), relation names, relational attribute names, or principals in the system (e.g., `Facilities`, `localhost`).

```
 <rclause> ::= “REQUIRING” <holds>
   <holds> ::= <hold> [“, AND” <holds>]
    <hold> ::= <cons> “HOLDS OVER” <dlist>

 <pclause> ::= “PREFERRING” <prefs> [<cascade>]
 <cascade> ::= “CASCADE” <prefs> [<cascade>]
   <prefs> ::= <pref> | <hold> [“AND” <prefs>]
    <pref> ::= <num>“(”<f>“)”

    <cons> ::= <operand> <cop> <operand>
 <operand> ::= <fvar> | <literal> | <set>
    <fvar> ::= “$”<literal>
     <cop> ::= “=” | “<>” | “IN” | “NOT IN”
     <set> ::= “{” <items> “}”
   <items> ::= <literal> | <items>

   <dlist> ::= <dnode> [“,” <dlist>]
   <dnode> ::= “⟨” <op> “,” <param> “,” <p> “⟩”
      <op> ::= <fvar> | “scan” | “select” | “project”
             | “join” | “product” | “rename” | “aggregate”
             | “sort” | “deduplicate” | “union” | “intersection”
             | “difference” | “*”
   <param> ::= “{” <pset> “}” | “*”
    <pset> ::= “(” <pitems> “)”
  <pitems> ::= <pitem> | <pitems>
   <pitem> ::= <pop> | <literal> | <set> | <agg>
             | <fvar> | “ASC” | “DESC”
     <agg> ::= “MIN” | “MAX” | “AVG” | “SUM” | “COUNT”
     <pop> ::= “=” | “<>” | “<” | “>” | “<=” | “>=”
             | “IN” | “NOT IN” | “BETWEEN” | “LIKE”
             | “IS NULL” | “IS NOT NULL” | “AND” | “OR”
       <p> ::= <literal> | <set> | <fvar> | “*”
```

Fig. 3. Syntax for the `REQUIRING` and `PREFERRING` clauses

B Expressive Capabilities

The examples that we have presented in the body of this work have served mostly explanatory purposes, illustrating the mechanics of (I, A)-privacy and further motivating the need for the protections that it offers. However, the preferences model described in this work is capable of expressing much more powerful controls over the execution of user queries than have so far been demonstrated. This section will demonstrate a range of common policy idioms that can be expressed within the privacy and execution preference framework developed in this paper.

Discretionary Access Control (DAC). In the access control literature, DAC policies allow users to explicitly list the identities of the other users permitted to access their files [19]. The notion of DAC policies has natural applications to user privacy in distributed query execution, as users might wish to white- or black-list individual servers from learning about their queries. In fact, all of the examples presented in previous sections of the paper have encoded very specific DAC policies restricting access to intensional regions to just the querier. A user could just as easily have required that certain intensional regions be executed by some remote server:

```
REQUIRING $p = Inventory HOLDS OVER <*, {("solvent")}, $p>
```

The above requires that any nodes matching the specified node descriptor be executed by the Inventory server. It is also possible to allow any remote server explicitly identified as belonging to some *set* of trusted servers to handle a particular intensional region:

```
REQUIRING $p IN {P, Q, R} HOLDS OVER <*, {("solvent")}, $p>
```

This `REQUIRING` block would force all matching query nodes be evaluated by some server in the set $\{P, Q, R\}$ of trusted servers.

Mandatory Access Control (MAC). In contrast to DAC systems, MAC systems rely on a centrally-defined security policy that cannot be overridden. For instance, the Bell-LaPadula model [1] is a MAC system that enforces access controls based on centrally-managed security clearances: e.g., users can read any file with a security level lower than their security clearance, but cannot read documents with a higher security level. To enforce MAC constraints, the client software from which queries are issued could automatically apply `REQUIRING` clauses to all outgoing queries. For ease of use in such cases, we allow the user of macros to define collections of principals (denoted here by the prefix "#") which could be parsed and replaced with a static list of principals by the trusted query processor as a first step in parsing.

To illustrate this point, consider an intelligence analyst using a top-secret clearance workstation looking over field agent reports concerning a certain date (say, 01-01-10). To ensure proper compartmentalization of the data from those reports, the query issuing client software on that workstation could ensure that all queries sent out are sent only to servers cleared to handle top-secret data with the following `REQUIRING` clause:

```
REQUIRING $p IN #top-secret-clearance HOLDS OVER <*,{("01-01-10")},$p>
```

Note that the preference framework articulated in this paper can allow MAC and DAC constraints to co-exist, as is often the case in environments using of MAC constraints [1].

Attribute-based Access Control (ABAC). ABAC policies allow access decisions to be made based upon the attributes of principals in the system, rather than their identities [25]. The macro mechanism described to support MAC policies could be leveraged by users—rather than the query issuing client—to enforce ABAC policies. For instance, Alice could require that any operation on the `Salary` table only be visible by servers run by the finance group:

```
REQUIRING $p IN #finance-group-servers HOLDS OVER <*,{("Salary")},$p>
```

In addition to supporting static, user-defined macros to encode server attributes, an interesting avenue of future work would be enabling support for dynamic macros to be built based upon unforgeable digital attribute credentials stored in a server's meta-data catalog. This would allow for more flexible ABAC support in which users can rely on the attestations of trusted certifiers to make attribute-based judgments regarding a server's characteristics.

Separation of Duty. Separation of duty (SoD) policies are used to require that multiple principals cooperate to carry out a particular action [2]. In the context of distributed database systems, one could use SoD to explicitly limit information flow when querying multiple tables replicated across a collection of servers by forcing each table scan to be performed by a different server. This is easily expressed by our model through the use of a single constraint over multiple node descriptors:

```
REQUIRING $p1 != $p2 HOLDS OVER
    <scan,{("Salary")},$p1>, <scan,{("EmploymentHistory")},$p2>
```

The above example would force the scans of the `Salary` and `EmploymentHistory` tables to occur at different sites.

Data Source Preference. In addition to privacy-related constraints, the preference model developed in this paper can also be used to enforce other execution preferences during query evaluation. For instance, users can specify preferences over the sources used to obtain replicated data:

```
PREFERRING $p = P HOLDS OVER <scan,{(A)},$p>
    CASCADE $p != R HOLDS OVER <scan,{(A)},$p>
```

This preference says that a user would prefer to get table `A` from the server `P`. If this fails, the table could be retrieved from any replica other than `R`. Such preferences would clearly benefit users who, even though a table is available from multiple sources, wish it to be acquired from a given source. This type of preference could arise, e.g., due to differing consistency guarantees offered by various sources. For instance, in the above, we could imagine `P` being the primary copy of a relational table, and `R` being an eventually consistent—and thus potentially out of date—replica.

Supporting Concurrency in Private Data Outsourcing

Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Stefano Paraboschi[2], Gerardo Pelosi[3], and Pierangela Samarati[1]

[1] Università degli Studi di Milano, 26013 Crema, Italy
firstname.lastname@unimi.it
[2] Università degli Studi di Bergamo, 24044 Dalmine, Italy
parabosc@unibg.it
[3] Politecnico di Milano, 20133 Milano, Italy
pelosi@elet.polimi.it

Abstract. With outsourcing emerging as a successful paradigm for delegating data and service management to third parties, the problem of guaranteeing proper privacy protection against the external server is becoming more and more important. Recent promising solutions for ensuring privacy in such scenarios rely on the use of encryption and on the dynamic allocation of encrypted data to memory blocks for destroying the otherwise static relationship between data and blocks in which they are stored. However, dynamic data allocation implies the need to re-write blocks at every read access, thus requesting exclusive locks that can affect concurrency.

In this paper we present an approach that provides support for concurrent accesses to dynamically allocated encrypted data. Our solution relies on the use of multiple differential versions of the data index that are periodically reconciled and applied to the main data structure. We show how the use of such differential versions guarantees privacy while effectively supporting concurrent accesses thus considerably increasing the performance of the system.

1 Introduction

The evolution of information and communication technology is leading to information system architectures that rely more and more on the outsourcing to other parties of IT functions that were typically managed within an organization. A major motivation for such trend, is economical: with outsourcing an organization can simplify its structure and benefit from the large scale economies of ad-hoc IT services, with low costs and high availability. However, a significant obstacle to a greater adoption of outsourcing is today represented by possible concerns over improper exposure of confidential or sensitive information. As a matter of fact, while the external service provider can be relied upon for guaranteeing security of data and services managed, it is of utmost importance to protect possible sensitive information from the eyes of the service provider itself.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 648–664, 2011.

The research and development communities have devoted significant attention to the problem of protecting data confidentiality in outsourcing scenarios, producing several solutions addressing different aspects of the problem. All proposals apply encryption to make data not intelligible to the server, providing support for query execution either by associating additional indexes with the encrypted data [1,3,8,14,15,20,21] or extending tree-based indexing structures typically adopted in DBMSs [8,17]. Tree-based approaches, unlike additional indexes, are not vulnerable to privacy breaches exploiting the possible correlation between frequencies of the index values and of the actual data behind them [3]. However, even tree-based data structures remain vulnerable to attacks based on the observation of sequences of accesses and on the analysis of the frequency distribution of access requests (i.e., by observing that certain physical blocks are often accessed). Such vulnerability can be counteracted by adopting approaches that change the location of the encrypted data at every access, so to break the otherwise static relationship between data and their physical location [10,17,22]. Dynamically allocated data structures represent the best defense against frequency attacks by the server. Among them, the shuffle index [10] extends the classical $B+$-tree structure used in databases with encryption, cover searches (to cover the actual target search with additional fake searches to "hide" it in a set and provide uncertainty over the block actually aimed by the access), and shuffling to enforce dynamic allocation. Although the shuffle index enjoys limited overhead with respect to the protection guarantees it offers [10], like other dynamically allocated data structures, it can potentially affect performance in scenarios where accesses need to operate concurrently. In fact, reallocating data at the server side requires write (hence exclusive) locks on the blocks involved in an access even in the execution of read-only operations.

In this paper, we extend the shuffle index to support a scenario where the data owner – who outsources data to the external server – wants to be able to execute several concurrent read-only transactions that need to access the remote data. Our solution to provide concurrent accesses to the shuffle index (Sect. 2) stored at the external server consists in having transactions operating on dynamically created portions of the index, which we call *delta versions* (Sect. 3). Delta versions are maintained in the server main memory, are managed – and shuffled at each access – independently one from the other (Sect. 4), and are periodically reconciled and applied to the main data structure on disk (Sect. 5). The use of periodically reconciled and merged delta versions offers protection against frequency attacks similar or better than the use of a single main index (Sect. 6) while producing an up to fourfold increase in system throughput (Sect. 7), thus offering a convincing argument for its adoption.

2 Preliminary Concepts

Before introducing our approach, we illustrate the shuffle index with which outsourced data are organized [10]. We assume data to be indexed over a candidate key and organized as an abstract *unchained B+-tree*, with actual data stored in

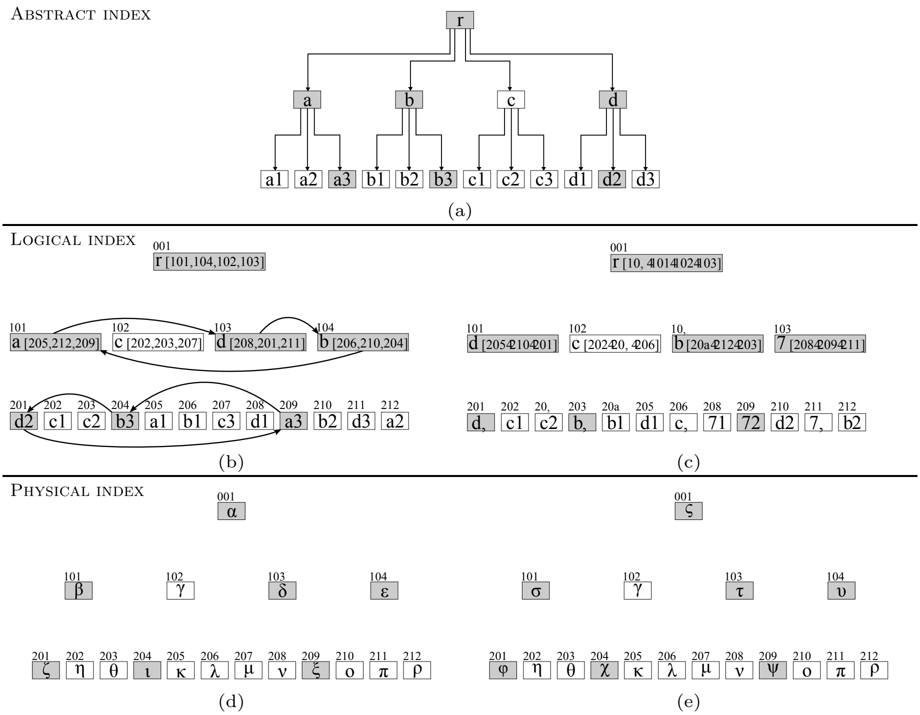


Fig. 1. An example of abstract (a), logical (b)-(c), and physical (d)-(e) index before (b)-(d) and after (c)-(e) the execution of a search operation

the leaves of the tree in association with their index values. The fact that the tree is unchained means that there are no links connecting the leaves. The fan-out F of the tree regulates the number of index values stored in the nodes. Each node stores a list $v[1, \ldots, q]$ of q values, with $\lceil \frac{F}{2} \rceil - 1 \leq q \leq F - 1$ (the lower-bound does not apply to the root) ordered from the smallest to the greatest, and has $q + 1$ children. The i-th children of a node is the root of the subtree containing the values val with $v[i-1] \leq val < v[i]$, $i = 2, \ldots, q$; the first child is the root of the subtree with all values $val < v[1]$, while the last child is the root of the subtree with all values $val \geq v[q]$. Figure 1(a) illustrates a graphical representation of our abstract data structure. For simplicity, in our examples we refer to nodes with a label (not explicitly reporting values in them). At the logical level, nodes are allocated to logical addresses that work as logical *identifiers*.

Pointers between nodes of the abstract data structure correspond, at the logical level, to node identifiers, which can then be easily translated at the physical level into physical addresses at the storing server. In the following, we assume that the physical address corresponds to the logical identifier of the node stored in it. Note that the possible order between identifiers does not necessarily correspond to the order in which nodes appear in the value-ordered abstract representation. Figure 1(b) illustrates a possible representation at the logical level of the abstract data structure in Fig. 1(a). In the figure, nodes appear ordered (left to right) according to their identifiers, which are reported on the top of each node. Pointers to children are represented by reporting in the node the ordered list of the identifiers of its children. For simplicity and easy reference, in our example, the first digit of the node identifier denotes the level of the node in the tree. For external outsourcing, node's content is prefixed with a random salt and then encrypted in CBC mode with a symmetric encryption function producing an encrypted block. Figure 1(d) illustrates the physical representation of the logic data structure in Fig. 1(b) (Greek letters represent the result of encryption). Since the block content is encrypted, the server does not have any information on the content or on the parent-child relationship between nodes stored in blocks. Retrieval of the leaf block containing the tuple corresponding to an index value requires an iterative process. Starting from the root of the tree and ending at a leaf, the read block is decrypted retrieving the address of the child block to be read at the next step. To avoid leaking to the server *i)* the fact that some blocks represent a path in the tree and *ii)* different accesses aim at the same content, the shuffle index extends the search operation by:

- performing, in addition to the target search, other fake *cover searches*, guaranteeing indistinguishability of target and cover searches and operating on disjoint paths of the tree (retrieving at every level of the tree *num_cover*+1 blocks at the same time);
- maintaining a set of blocks in a local *cache*;
- mixing (*shuffling*) the content of all retrieved blocks as well as those maintained in cache and rewriting them accordingly on the server.

Intuitively, cover searches introduce uncertainty over the leaf block actually belonging to the target search and do not allow the server to establish the parent-child relationship between blocks (since multiple blocks are retrieved at every level). The cache is used to make searches repeated within a short time interval not recognizable as being the same search (if the nodes in the target path are already in cache, an additional cover search will be executed instead). Shuffling moves content among blocks, thus breaking the correspondence between nodes (contents) and blocks (addresses). Note that shuffling requires to re-encrypt the involved nodes with a different random salt, so to produce a different encrypted text, and changing the pointers to them in their parents (which will have to point to the new blocks at which nodes have been allocated). Changing the allocation of nodes to blocks provides confidentiality: *i)* subsequent searches looking for the same content would aim at different blocks, and *ii)* subsequent searches hitting the same block would involve a different content.

As an example, consider a search for value $b3$ over the abstract index in Fig. 1(a) that adopts $a3$ as cover, and assume that the local cache contains the path to $d2$ (i.e., (001,103,201)). The nodes involved in the search operation are denoted in gray in the figure. Figure 1(b) illustrates the logical representation of the abstract index before the execution of the search operation and how accessed blocks are shuffled, level by level, to obtain the structure in Fig. 1(c). Note that although the server knows which blocks have been accessed (gray blocks in

Figs. 1(d)-(e)) it cannot detect which of those is the actual search target and how the content of blocks has been shuffled, since blocks are encrypted using a different salt at each encryption.

3 Main Index and Delta Versions

Before introducing the concept of delta version, we need to formalize the different components of the shuffle index data structure and of the shuffling (which were only procedurally managed in the original proposal). Data can be seen at the abstract, logical, and physical levels, which we formally capture as follows.

- *Abstract* ($\mathcal{T}^a$): set $\{n_1^a, \ldots, n_m^a\}$ of abstract nodes forming an unchained B+-tree. Each internal node in $\mathcal{T}^a$ is a pair $n^a = \langle \mathit{values}, \mathit{children} \rangle$ with *values* a list of index values and *children* a list of $q + 1$ child nodes. Leaf nodes have *tuples*, representing the tuples with index value in *values*, instead of *children*.
- *Logical* ($\mathcal{T}$): triple $(\mathcal{T}^a, \mathcal{ID}, \phi)$, where $\mathcal{T}^a$ is an abstract data structure, $\mathcal{ID}$ is a set of logical identifiers, and $\phi : \mathcal{T}^a \rightarrow \mathcal{ID}$ is a bijective function associating each abstract node n^a in $\mathcal{T}^a$ with a logical identifier *id* in $\mathcal{ID}$. Triple $(\mathcal{T}^a, \mathcal{ID}, \phi)$ determines how the abstract nodes in $\mathcal{T}^a$ are allocated to logical identifiers in $\mathcal{ID}$. Each internal node $n^a{=}\langle \mathit{values}, \mathit{children} \rangle \in \mathcal{T}^a$ is then represented by a (logical) node of the form $\langle id, v, p \rangle$, where $id{=}\phi(n^a)$, $v{=}\mathit{values}$, and $p[j]{=}\phi(\mathit{children}[j])$, $j = 1, \ldots, q + 1$. Leaf nodes are represented with logical nodes of the form $\langle id, v, t \rangle$ that include tuples t instead of pointers to children.
- *Physical* ($\mathcal{T}^e$): set of (disk) blocks storing $\mathcal{T}$. Each logical node $\langle id, v, p \rangle \in \mathcal{T}$ (leaf $\langle id, v, t \rangle \in \mathcal{T}$, resp.) is stored in a block that can be seen as a pair of the form $\langle id, b \rangle$, where $b{=}E_k(\mathit{salt}||id||v||p)$ ($b{=}E_k(\mathit{salt}||id||v||t)$, resp.) with E a symmetric encryption function, k the encryption key, and *salt* a value chosen at random during each encryption.

In the following, we use the term node to refer to an abstract content and block to refer to a specific memory slot in the logical/physical structure. When either term can be used, we will use node/block interchangeably.

Shuffling executed at every access randomly exchanges the content among blocks. A shuffling of logical index $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ is equivalent to reallocating nodes to potentially different blocks (the corresponding abstract index remains unaltered), as formally defined in the following.

Definition 1 (Shuffling). *Let* $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ *be a logical index and* $\pi : \mathcal{ID} \rightarrow \mathcal{ID}$ *be a random permutation of* $\mathcal{ID}$*. The* shuffling *of* $\mathcal{T}$ *with respect to* π *is a logical index* $\mathcal{T}' = (\mathcal{T}^a, \mathcal{ID}, \phi')$*, where* $\forall n^a{\in}\mathcal{T}^a$*,* $\phi'(n^a) = \pi(\phi(n^a))$*.*

Note that a change in the allocation of nodes to blocks implies that the pointers to children must be updated to reflect their new allocation, thus preserving the correct parent-child relationship. In the following, for convenience we assume shuffling to operate within the boundary of the tree level (i.e., permutations are always performed among nodes of the same level of the tree).

A delta version is essentially a – potentially shuffled – portion of the main index, as captured by the following definition.

Definition 2 (Delta version). *Let $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ be a logical index. A* delta version *$\Delta_i = (\Delta_i^a, \mathcal{ID}_i, \phi_i)$ of $\mathcal{T}$ is a shuffling of $(\Delta_i^a, \mathcal{ID}_i, \phi)$, where $\Delta_i^a \subseteq \mathcal{T}^a$ such that $\forall n^a \in \Delta_i^a$, the parent of n^a belongs to Δ_i^a; $\mathcal{ID}_i = \bigcup \phi(n^a)$, $n^a \in \Delta_i^a$; and $\phi_i : \mathcal{T}^a \rightarrow \mathcal{ID}$ such that $\phi_i(n^a) = \phi(n^a)$ if $n^a \notin \Delta_i^a$.*

Figure 2(c) illustrates an example of delta version of the logical index in Fig. 2(a). Note that, since a delta version is composed of nodes forming paths that are traversed when executing search operations, the parent of each node in the delta version also belongs to the delta version. As a consequence, every delta version always includes the root of $\mathcal{T}^a$.

Merging a delta version with a main index implies enforcing on the main index the allocation of nodes to blocks prescribed by the delta version, as captured by the following definition.

Definition 3 (Merge). *Let $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ be a logical index and $\Delta_i = (\Delta_i^a, \mathcal{ID}_i, \phi_i)$ be a delta version of $\mathcal{T}$. The* merge *of $\mathcal{T}$ and Δ_i, denoted $\mathcal{T} \oplus \Delta_i$, is logical index $\mathcal{T}' = (\mathcal{T}^a, \mathcal{ID}, \phi_i)$.*

In terms of actual enforcement, $\mathcal{T}'$ can be simply obtained by flushing the blocks of the delta version to the main index (overwriting the corresponding blocks on disk), while leaving the other blocks unaltered. Such an operation – which can be performed without any need to download the involved blocks or performing computation by the client – produces an index that correctly represents the original data structure and includes the shuffling operated in the delta version.

4 Operating on Delta Versions

The basic idea of our approach is that transactions operate on delta versions (dynamically created and maintained in main memory at the server) rather than on the main shuffle index.

Access Execution. Every access operation is executed over a delta version. If the operation needs to read a block that does not belong to the delta version, such a block is taken from the main index and included in the delta version. Access execution works essentially like in the original shuffle index proposal requesting at every level at least *num_cover*+1 blocks. Apart from the need to include new blocks in the delta version, the only notable difference with respect to the original shuffle index proposal is that we depart from the local cache originally maintained for hiding the fact that subsequent searches were aiming at the same node. The reason for departing from the cache is that its maintenance would impose a strong synchronization overhead among the different transactions operating at the client side. To prevent the server from recognizing

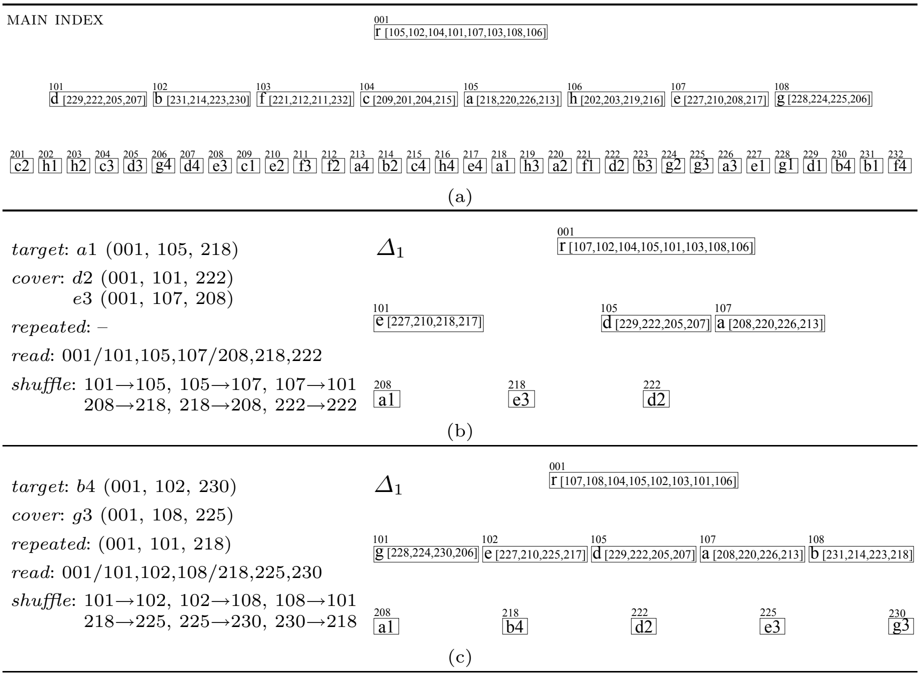

MAIN INDEX
001 r [105,102,104,101,107,103,108,106]
101 d [229,222,205,207]
102 b [231,214,223,230]
103 f [221,212,211,232]
104 c [209,201,204,215]
105 a [218,220,226,213]
106 h [202,203,219,216]
107 e [227,210,208,217]
108 g [228,224,225,206]
201 c2 202 h1 203 h2 204 c3 205 d3 206 g4 207 d4 208 e3 209 c1 210 e2 211 f3 212 f2 213 a4 214 b2 215 c4 216 h4 217 e4 218 a1 219 h3 220 a2 221 f1 222 d2 223 b3 224 g2 225 g3 226 a3 227 e1 228 g1 229 d1 230 b4 231 b1 232 f4
(a)
target: a1 (001, 105, 218)
cover: d2 (001, 101, 222)
e3 (001, 107, 208)
repeated: –
read: 001/101,105,107/208,218,222
shuffle: 101→105, 105→107, 107→101
208→218, 218→208, 222→222
Δ1
001 r [107,102,104,105,101,103,108,106]
101 e [227,210,218,217]
105 d [229,222,205,207]
107 a [208,220,226,213]
208 a1
218 e3
222 d2
(b)
target: b4 (001, 102, 230)
cover: g3 (001, 108, 225)
repeated: (001, 101, 218)
read: 001/101,102,108/218,225,230
shuffle: 101→102, 102→108, 108→101
218→225, 225→230, 230→218
Δ1
001 r [107,108,104,105,102,103,101,106]
101 g [228,224,230,206]
102 e [227,210,225,217]
105 d [229,222,205,207]
107 a [208,220,226,213]
108 b [231,214,223,218]
208 a1
218 b4
222 d2
225 e3
230 g3
(c)

Fig. 2. An example of main index (a) and of execution of two subsequent searches (b)-(c) over it using delta version Δ_1

that two subsequent accesses aim at the same block, we take a dual approach and adopt *repeated searches*. Intuitively, while the cache ensured consequent searches never accessed the same block (if a value just retrieved was needed, a fake value was searched instead, so to ensure no intersection between the two searches and the same number of blocks is accessed at each level), repeated searches always ensure intersection between subsequent searches (regardless of whether the two searches are looking or not for the same value). For enforcing repeated searches, we store, in conjunction with each delta version, a layered structure that keeps track of the identifiers of the blocks accessed during the last search. Execution of an access on a delta version will also request at least one block per level among those appearing in the last search. Each search then accesses *num_cover*+2 blocks at every level of the index, since, besides the blocks of the target and cover searches, an additional block is necessary for the repeated search (the additional blocks are two if the target or cover searches correspond to a repeated search). At the beginning, when the delta version is empty, there is no search to repeat and an additional cover is requested instead. To illustrate, consider the index in Fig. 2(a) and a request for value $a1$ that adopts one cover and operates on empty delta version Δ_1. In this case, two covers (e.g., $d2$ and $e3$) are needed. The blocks on the paths to $a1$, $d2$, and $e3$ are all read from the main index, shuffled, and written back in Δ_1 as illustrated Fig. 2(b). Suppose now to execute another search for value $b4$ over Δ_1, with cover $g3$, and one repeated access (e.g., 001, 101, 218). Since the nodes along the paths to $b4$ and $g3$ (except the root) do not belong to Δ_1 they are read from the main index, and after shuffling their content with all accessed blocks, are copied in the delta version. Figure 2(c) illustrates Δ_1 after the execution of the second search operation.

Delta Version Assignment. To avoid imposing synchronization constraints at the client side, we assume the allocation of delta versions to each transaction to be determined by the server. However, we need to provide a means at the client side to control the proper behavior of the server in the allocation of the versions. It is important to ensure that the server does not discard the shuffling requested, creates a new delta version at each access and having then transactions always operating on the main index (and therefore on a static data structure), or selectively allocates versions to monitor specific activities. Therefore, we assume that the client sets the number of delta versions (i.e., amount of concurrent operations). At the client side, we maintain a table Version($\underline{\Delta id}$, *ts*, *status*), reporting for each delta version Δid the time *ts* of last access and whether its *status* is busy or free. We assume synchronization before execution of each search operation, requesting the transaction at the client side to update the entry for the version allocated to it setting *ts* to the current time and *status* to busy. We instead account for a lazy process for the transactions in setting that the version allocated to them has been released (*status* free). Hence, while a version appearing free in the table is certainly free, a version appearing busy could actually have been released (but the transaction be late in reporting the status change). We request the server to manage delta version allocation according to the MRU policy, that is, an access should be enforced on the most recently used version. The client can then check that the server has performed proper allocation by checking that the delta version allocated to the request has *ts* greater than the greatest *ts* associated with a free version in the table (the greater than condition is to accommodate for possible delays at the client side to set version status free). We also assume the root of every delta version to be timestamped at each access. This allows checking that the root is actually the result of the access executed at the time *ts* recorded in the table for the delta version and, therefore (since the root points to the other blocks in the tree) the freshness of the whole version.

5 Reconciling Delta Versions and Main Index

A delta version grows at every access by including new requested blocks that were not previously contained in the delta version. In the long run, a delta version could potentially grow to include all the blocks of the main index saturating the server main memory. Hence, we periodically synchronize the main index with the delta versions, reporting shuffling operations on the main index and resetting the delta versions. Note that we cannot simply destroy the delta versions without changing the main index. In fact, although all operations are read-only (i.e., the abstract data structure remains unaltered), the principle of the shuffle index is that the allocation of nodes to blocks is dynamic. It is therefore important to apply the shuffle performed on the delta versions to the main index, so to enjoy the protection of shuffling for subsequent accesses.

If there were a single delta version, applying the performed shuffling on the main index would be simple. Indeed, it would be sufficient to simply flush to the main index on disk the blocks included in the delta version. The situation is however complicated by the existence of several delta versions, which can have operated independently on the same nodes/blocks. In this case, a reconciliation is needed to ensure correctness of the index and, in particular, to ensure no content is lost and pointers to child blocks are properly set. We first note that, while it is important that shuffling is enforced in the main index, the specific way in which nodes are shuffled (i.e., which node goes to which block) does not have any impact, provided it represents a random permutation. As long as allocation is dynamic, any rearrangement would do. Hence, a straightforward approach to enforce shuffling on the main index would be to download all the blocks contained in the delta versions at the client side, retrieve (by decrypting) the corresponding nodes, allocate them to blocks, and re-uploading them at the server by rewriting the involved blocks on the main index. Such a naive approach, requiring to download all the blocks and to re-encrypt all the nodes, is clearly too expensive and not needed. Our approach aims at minimizing the blocks to be downloaded and re-uploaded by limiting these blocks to the ones strictly needed to guarantee correctness or to avoid leakage on the node allocation, while flushing as many blocks as possible directly to disk.

To determine which blocks need to be downloaded and re-encrypted, we have to identify the blocks for which the presence of multiple delta versions represents a problem. In principle, it is sufficient for two delta versions to have a block (and hence the corresponding node) in common to require checking all the blocks in them, since the node (which should be reported in only one block to the main index) may have been re-allocated to any of the blocks within each delta version. In practice, however, only the block where the node was originally allocated in the main index and the new block where it has been allocated in each of the delta versions need to be strictly involved in some re-encryption, since the delta versions have conflicting node/block allocation.

We then start by characterizing conflicting node/block allocation among a set of delta versions as follows.

Definition 4 (Conflicting allocations). *Let* $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ *be a logical index and* $\{\Delta_1, \ldots, \Delta_n\}$ *be a set of delta versions of* $\mathcal{T}$*. The* conflicting allocations *of* Δ_i *with respect to* $\{\Delta_1, \ldots, \Delta_n\} \setminus \{\Delta_i\}$ *is a set* $\mathcal{C}_i$ *of pairs* $\langle n_i^a, id_i \rangle$*, where* $n_i^a \in \Delta_i^a$*,* $id_i = \phi_i(n_i^a)$*, and* $\exists n_j^a \in \Delta_j^a$*,* $\Delta_j \in \{\Delta_1, \ldots, \Delta_n\}$ *and* $i \neq j$*, such that either:* 1) $n_i^a = n_j^a$ *(same node); or* 2) $\phi_i(n_i^a) = \phi_j(n_j^a)$ *(same block).*

It is easy to see that, with respect to nodes, the nodes that are in conflict for a given delta version Δ_i are all those nodes that are also present in another version (i.e., belong to $\Delta_i^a \cap \Delta_j^a$, for some j) or are contained in a block which is also present in another version (i.e., are allocated to a block in $\mathcal{ID}_i \cap \mathcal{ID}_j$, for some j). Analogously, with respect to blocks, the blocks that are in conflict for a given delta version Δ_i are all those blocks that are also present in another

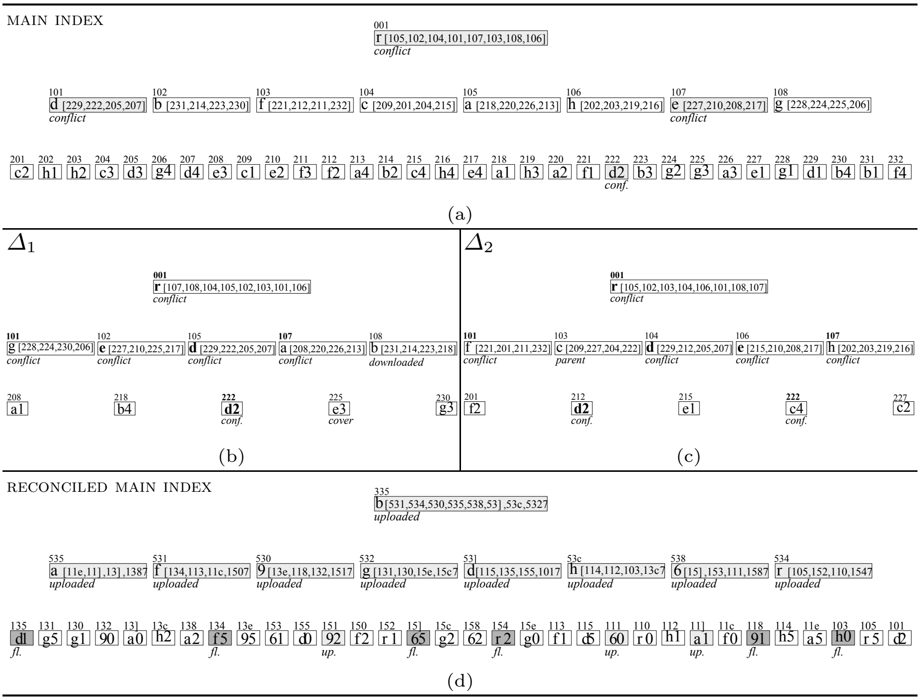

MAIN INDEX
001 r [105,102,104,101,107,103,108,106] conflict
101 d [229,222,205,207] conflict
102 b [231,214,223,230]
103 f [221,212,211,232]
104 c [209,201,204,215]
105 a [218,220,226,213]
106 h [202,203,219,216]
107 e [227,210,208,217] conflict
108 g [228,224,225,206]
201 c2
202 h1
203 h2
204 c3
205 d3
206 g4
207 d4
208 e3
209 c1
210 e2
211 f3
212 f2
213 a4
214 b2
215 c4
216 h4
217 e4
218 a1
219 h3
220 a2
221 f1
222 d2 conf.
223 b3
224 g2
225 g3
226 a3
227 e1
228 g1
229 d1
230 b4
231 b1
232 f4
(a)
Δ_1
001 r [107,108,104,105,102,103,101,106] conflict
101 g [228,224,230,206] conflict
102 e [227,210,225,217] conflict
105 d [229,222,205,207] conflict
107 a [208,220,226,213] conflict
108 b [231,214,223,218] downloaded
208 a1
218 b4
222 d2 conf.
225 e3 cover
230 g3
(b)
Δ_2
001 r [105,102,103,104,106,101,108,107] conflict
101 f [221,201,211,232] conflict
103 c [209,227,204,222] parent
104 d [229,212,205,207] conflict
106 e [215,210,208,217] conflict
107 h [202,203,219,216] conflict
201 f2
212 d2 conf.
215 e1
222 c4 conf.
227 c2
(c)
RECONCILED MAIN INDEX
335 b [531,534,530,535,538,53] ,53c,5327 uploaded
535 a [11e,11],13],1387 uploaded
531 f [134,113,11c,1507 uploaded
530 9 [13e,118,132,1517 uploaded
532 g [131,130,15e,15c7 uploaded
53] d [115,135,155,1017 uploaded
53c h [114,112,103,13c7 uploaded
538 6 [15],153,111,1587 uploaded
534 r [105,152,110,1547 uploaded
135 dl fl.
131 g5
130 g1
132 90
13] a0
13c h2
138 a2
134 f5 fl.
13e 95
153 61
155 d0
151 92 up.
150 f2
152 r 1
15] 65 fl.
15c g2
158 62
154 r 2 fl.
15e g0
113 f1
115 d5
111 60 up.
110 r 0
112 h1
11] a1 up.
11c f0
118 91 fl.
114 h5
11e a5
103 h0 fl.
105 r 5
101 d2
(d)

Fig. 3. An example of main index (a), two delta versions Δ_1 (b) and Δ_2 (c), and the result of their reconciliation (d)

version (i.e., belong to $\mathcal{ID}_i \cap \mathcal{ID}_j$, for some j) or that contain a node that is also present in another version (i.e., belong to $\Delta_i^a \cap \Delta_j^a$, for some j). For completeness, Definition 4 captures both components representing conflicts, in terms of pairs $\langle node, block \rangle$ since the conflict requires to revisit the allocation of the *node* contained in *block*. To illustrate, consider the two delta versions Δ_1 and Δ_2 in Figs. 3(b)-(c). The nodes/blocks representing a conflicting allocation in each version are marked with the word *conflict* below the block.

All blocks involved in a conflict for some delta version are blocks that cannot be simply written to disk as the resulting index would not be correct (some nodes would be lost and others would appear replicated). To ensure consistency of the content, it is important to reconcile the delta versions so that there is agreement – with respect to common nodes or common blocks – on which node is allocated to which block. We capture this by formalizing the definition of reconciled delta version, resulting from a reconciliation of different delta versions, as follows.

Definition 5 (Reconciled delta version). *Let $\mathcal{T} = (\mathcal{T}^a, \mathcal{ID}, \phi)$ be a logical index, $\{\Delta_1, \ldots, \Delta_n\}$ be a set of delta versions of $\mathcal{T}$, and $\mathcal{C}_i$ be the conflicting allocations of Δ_i with respect to $\{\Delta_1, \ldots, \Delta_n\} \setminus \{\Delta_i\}$, $i = 1, \ldots, n$. A* reconciled delta version *of $\{\Delta_1, \ldots, \Delta_n\}$ is a delta version $\Delta_r = (\Delta_r^a, \mathcal{ID}_r, \phi_r)$ where $\Delta_r^a =$*

$\Delta_1^a \cup \ldots \cup \Delta_n^a$, $\mathcal{ID}_r = \mathcal{ID}_1 \cup \ldots \cup \mathcal{ID}_n$, and $\phi_r(n^a) = \phi_i(n^a)$ if $n^a \in \Delta_i^a$ and $\langle n^a, \phi_i(n^a)\rangle \notin \mathcal{C}_i$.

The reconciled delta version can then be enforced on the shuffle index as in the case of a single delta version, by merging $\mathcal{T}$ and Δ_r producing logical index $\mathcal{T}_r = \mathcal{T} \oplus \Delta_r$ that represents the same abstract index represented by $\mathcal{T}$.

For producing the reconciled version, in addition to blocks in conflict also the blocks containing a pointer to a block in conflict (e.g., block 103 in Δ_2 in Fig. 3(c)) need to be re-written, as the pointer should be changed to refer to the new block where the child node (e.g., $c4$) has been allocated.

While the blocks in conflict and their parents are the only ones that should be downloaded by the client and re-uploaded (after shuffling the nodes in conflict) to produce a correct reconciled version (all other blocks in the delta versions could simply be flushed to disk directly by the server), we may need to download (and either include in the shuffling or simply re-write) other blocks. The reason is to ensure that the server cannot infer node/block allocation by observing that only few blocks have been involved in a reconciliation. As an example, for Δ_1 in Fig. 3(b), the only leaf block to download and re-upload would be conflicting block 222, therefore the server can infer that it stores the value accessed (as target or cover) by two searches performed with different delta versions. To avoid leakages like this, and providing the same uncertainty over the block allocation enjoyed by the original shuffle index proposal in the access execution, we require each version, for each level of the index, to: *i)* perform shuffling of either 0 or at least *num_cover*+1 blocks and *ii)* flush directly either 0 or not less that *num_cover*+1 blocks. If for a given level there are less than *num_cover*+1 blocks to flush, additional cover blocks are also downloaded and re-uploaded after re-encrypting them with a new salt (to make them not recognizable). Like parents, these latter nodes are not involved in the shuffling to avoid propagating the need for changes to higher levels of the index. For instance, with reference to Δ_1 in Fig. 3(b): *i)* 225 is added as cover to perform shuffling among at least two nodes at leaf level, and *ii)* 108 is also downloaded since it would have been the only one flushed at level one. Figure 3(d) illustrates the merging of the index in Fig. 3(a) after reconciliation of delta versions Δ_1 and Δ_2 in Figs. 3(b)-(c). The gray blocks are those that have been written on disk because flushed from main memory or re-uploaded by the client.

6 Security Analysis

We analyze the protection offered by our proposal for the new aspects introduced with respect to the serial version operating only with the main index. Like in the original proposal, we focus the analysis on leaves of the index (nodes at a higher level are subject to a greater number of accesses, due to the multiple paths that pass through them, and are then involved in a larger number of shuffling operations, which increase their protection). Since our search operations execute essentially like in the original proposal (with repeated searches instead of cache), our solution enjoys the protection guarantees given by covers like in [10]. The only potential exposure in our solution is when two different delta versions require access to a block in the main index for the first time. Since the main index changes only upon reconciliation, the server can infer that the two requests actually refer to the same node. However, since every access execution entails reading at least $num\_cover{+}1$ blocks (in addition to the repeated search) at every level, and covers are chosen guaranteeing indistinguishability (with respect to access profiles) between target and covers, the server cannot determine whether the transactions operating on the two different delta versions are actually aiming at the same target, or either or both of them are accessing the block as a cover. The probability that the two transactions aimed at the same target is then $\frac{1}{(num\_cover+1)^2}$; when m delta versions request access to the same block from the main index, the probability that all the transactions aimed at the same target is $\frac{1}{(num\_cover+1)^m}$.

The crucial property we are interested in evaluating is the protection against the inferences the server may make on the data content by exploiting information on the frequency of accesses to the blocks. Applying classical concepts of information theory, we can model the information available to the server on the association between a node n_i^a and block id_j storing it as probability $\mathcal{P}(n_i^a, id_j)$. A value equal to 1 for this probability means that the server will be able to correctly identify a node, whereas a value equal to $\frac{1}{|\mathcal{T}^a|}$ will correspond to the absence of any knowledge. If the block is replicated in delta versions, each instance will be associated with the analogous probability. Let ID' be the set of blocks involved in an access in a version (excluding the repeated search). For all $n_i^a \in \mathcal{T}^a$, and all $id_j{\in}ID'$, $\mathcal{P}(n_i^a, id_j)$ after the shuffling becomes $\sum_{id_j \in ID'} \frac{\mathcal{P}(n_i^a, id_j)}{num\_cover+1}$, because the shuffling can associate each node with any of the blocks involved in the access with equal probability, thus flattening the probability distribution. After the reconciliation, all the blocks that have been accessed by a single version will be transferred to the main index, where they will be associated with the probabilities computed in the version. Blocks accessed by multiple versions will be shuffled together, with a further averaging of probabilities among the blocks. As a consequence, $\mathcal{P}(n_i^a, id_j)$ for each node n_i^a after each access and each reconciliation will progressively move toward value $\frac{1}{|\mathcal{T}^a|}$.

It is natural to study the evolution of these probabilities using the concept of entropy, which allows us to identify at an aggregate level the knowledge of the server and its degradation due to shuffling and merging. In particular, we are interested in the impact of delta versions over the entropy, which we evaluated – as common in the study of codes and channels when analytical models become unmanageable – experimentally. We then designed a set of experiments with an initial configuration corresponding to a worst case assumption where the server has a precise knowledge about the node-block correspondence, and then the entropy is equal to zero, and evaluated how the entropy increases with access execution (for the serial index) and with access execution and merging after reconciliation (for our proposal). The experiments have considered a variety of configurations, with different numbers of nodes, number of versions, *num_cover*, and access profiles. Access profiles have been simulated by synthetically generating a sequence of

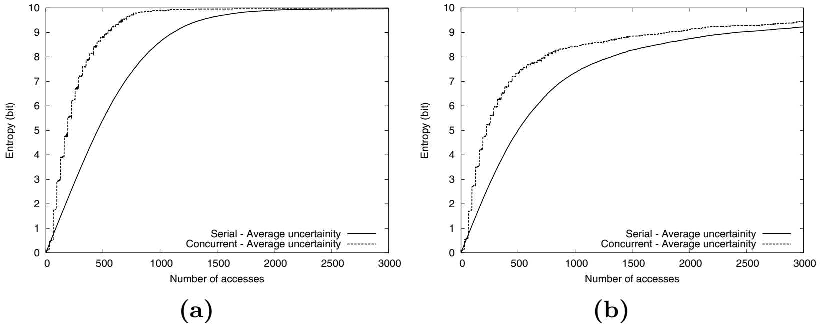


Fig. 4. Evolution of the entropy for values of γ equal to 0.5 (a) and 0.25 (b)

accesses that follow a self-similar probability distribution with skewness γ in the range $[0.25, 0.5]$ (given a domain of cardinality d, a self-similar distribution with skewness γ provides a probability equal to $1 - \gamma$ of choosing one of the first γd domain values). We then applied the same sequence of accesses to the serial and concurrent shuffle index and evaluated the growth of the entropy. Figure 4 illustrates the experimental results using 4 covers, 4 versions, 1000 nodes, skewness γ equal to 0.5 and 0.25, and varying the number of accesses. Experiments with different configurations presented a similar behavior.

As visible from the figure, before the first reconciliation, the entropy is slightly lower in the concurrent scenario with respect to the serial index. The reason is that each delta version serves a smaller number of accesses than the index in the serial version (assuming uniform distribution of load among versions, each transaction has one fourth of the accesses operating on the main index). However, already at the first reconciliation, the entropy for the concurrent scenario becomes higher than that of the serial scenario, and keeps maintaining higher. While an even higher entropy might sound not intuitive and an unexpected advantage (more protection with better performances), the explanation for such a behavior is simply that reconciliation and merging enjoy shuffling over a larger number of nodes all at one time. In fact, reconciliation makes the concurrent shuffle index stronger because this phase applies a shuffle over all the nodes in the conflict set. The size of this set depends on the number of conflicts and our model forces it to be for each delta version at least as large as the number of covers used for every shuffle. The size of the conflict set will often be greater than the number of covers, and the growth of entropy produced by a shuffle increases more than linearly with the number of blocks involved in the shuffle (i.e., the execution of two shuffles over two sets of m distinct elements produces lower entropy than a single shuffle over the set of $2m$ elements). The cost of such better protection can be reconducted to the cost of the reconciliation, which is below 10% of the access cost in the configuration that maximizes the server throughput (Sect. 7).

7 Performance Analysis

We implemented the search and reconciliation algorithms with Java programs. To assess the system performance, we used a data set of 1TB stored in the leaves of a shuffle index with 4 levels, built on a numerical candidate key of fixed-length, with fan-out 512, and representing 2^{32} (over 4 billion) different index values. The hardware used in the experiments included a server machine with 2 Intel Xeon Quad 2.0GHz L3-4MB, 12GB RAM, four 1TB disks, 7200RPM, 32MB cache, and Linux Ubuntu 9.04 x86_64 with the ext4 file system, and a client machine with an Intel Core 2 Duo CPU T5500 at 1.66GHz, 2GB DRAM, and Linux Ubuntu 9.04 x86. The client and the server operate in a local area network (100Mbps Ethernet, with average RTT of 0.48ms). The results reported in this section have been obtained as the average over 50 runs and, for each run, the number of accesses is 5000 and the number of covers adopted at each access is 4. The inverse of the average disk time needed to perform a single search is 52tps and represents the upper bound for the maximum throughput of the system.

To emulate the workload of an outsourcing service, we designed a generator scheme, modeling the number of access requests per second as a random variable following a Poisson distribution with mean arrival rate λ (the time when an access request arrives is independent from the time of arrival of previous requests). In our experiments, we considered λ=16tps and λ=32tps, which correspond to 30% and to 60%, respectively, of the physical maximum throughput (52tps). These are sensible workloads for a service hosted on a single machine and a robust test for the deployment of the proposed solution in a real world scenario. In fact, a workload of 60% of the maximum disk service rate is known to be optimal with respect to the upper bound of the physical maximum throughput [16].

To evaluate the performance gain obtained with the support of concurrent searches and the overhead due to reconciliation, we compare the server throughput in three different scenarios: *i)* serial shuffle index [10]; *ii)* concurrent shuffle index where delta versions are never reconciled; and *iii)* concurrent shuffle index where delta versions are periodically reconciled. In the experiments, delta versions are reconciled every 128 and every 256 access requests, for the configuration with λ=16tps and λ=32tps, respectively. A higher reconciliation frequency increases overhead because it more often requires write locks on the disk blocks to be re-written. On the other hand, a lower frequency requires less often such locks but over a considerably larger number of blocks (conflicts among versions grow more than linearly with respect to the number of searches). Experiments (which we do not present here for space reasons) show that the chosen threshold values balance the two aspects offering the maximum server throughput for the employed operating setup. Figure 5 reports the server throughput, varying the maximum number of delta versions between 1 and 128 with access request arrival rate equal to λ=16tps and λ=32tps, respectively. Although the performance overhead of concurrent applications highly depends on the random disk access patterns required to execute read and write accesses to blocks, Fig. 5 demonstrates how the adoption of our concurrency support offers a threefold (fourfold, respectively) increase of the server throughput compared to the serial shuffle index when

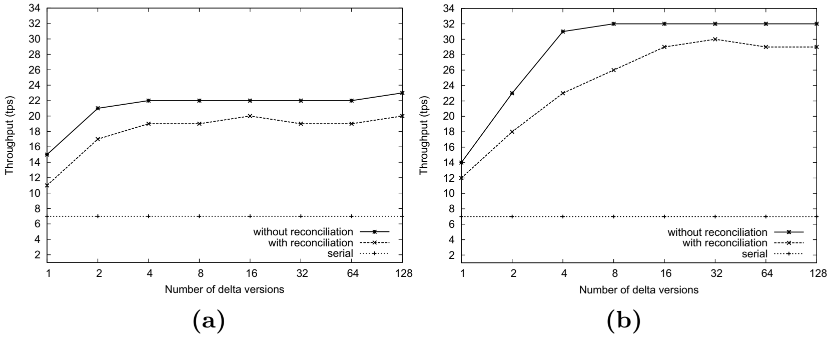


Fig. 5. Server throughput varying the number of delta versions between 1 and 128, with access request arrival rate equal to λ=16tps (a) and λ=32tps (b)

λ=16tps (λ=32tps, respectively). Note that the server throughput is higher than or equal to the mean arrival rate λ of client requests, meaning that the time necessary to the server to process an access request is lower than the time between two consecutive accesses. Figure 5 also highlights the limited cost due to reconciliation, which has a maximum of 25% and is 6% in the configuration that maximizes the server throughput.

8 Related Work

Previous work is related to the definition of indexing structures for the execution of queries on encrypted outsourced data (e.g., [1,8,14,15,20,21]). The proposals in [8,21] specifically adopt the B+-tree and the B-tree data structures to define an index able to efficiently support search operations on the key attribute. Although these solutions efficiently support accesses to the outsourced data, they are static and do not offer protection against the attacks based on the frequency of the accesses. Another line of work related to our is represented by Private Information Retrieval (PIR) [4,18]. These proposals typically protect the confidentiality of users' queries while data confidentiality is not considered an issue.

The proposals in [10,17,22] aim to protect data confidentiality and the accesses realized by the client over the data. The solution in [17] is based on the definition of a B-tree index and of a technique for accessing the content of a node in the tree that prevents the server from inferring which node has been accessed. However, the server can observe repeated accesses to the same physical block, which correspond to repeated searches for the same values, and apply a frequency attack to infer information about the values stored by each node in the B-tree. The proposal in [22] adopts the pyramid-shaped database layout of Oblivious RAM [13] and an enhanced reordering technique between adjacent levels of the data structure to protect both data confidentiality and the secrecy of users' queries. The performance of a search operation is however highly affected by the reordering of lower levels of the database, since this reordering can take hours and needs to be periodically performed. This appears a strong obstacle to the real deployment of such a solution. The architecture proposed in [22] also requires a secure coprocessor trusted by the client on the server. The first proposal combining shuffling, cover searches, and cache to offer an extensive protection of confidentiality with a limited overhead in response times is illustrated in [10], where data are organized according to a novel data structure whose management does not rely on a trusted component at the server side. However, such proposal as well as the proposals in [17,22] do not support concurrency, with consequent performance limits in many real life scenarios.

9 Conclusions

Dynamically allocated data structures have recently emerged as a promising solution to provide privacy protection of data whose storage and management are delegated to external servers. However, even solutions guaranteeing limited performance overheads could be affected in scenarios where several accesses need to operate concurrently, therefore impacting their application. In this paper, we have addressed this problem and presented a proposal for accommodating concurrent executions over a shuffle index whose working (based on multiple searches and dynamic data allocation) would otherwise require several exclusive locks which, while causing only a limited overhead in serial environments, could considerably affect concurrent accesses. Our proposal, based on operating on multiple differential versions of the index, enjoys a privacy protection against frequency attacks comparable to or better than the serial solution while offering up to fourfold throughput, thus providing a convincing argument for its adoption.

Acknowledgments. This work was supported in part by the EC within the 7FP, under grant agreements 216483 (PrimeLife) and 257129 (PoSecCo), by the Italian Ministry of Research within the PRIN 2008 project “PEPPER” (2008SY2PH4), and by the Università degli Studi di Milano within the project “PREVIOUS”.

References

1. Agrawal, R., Kierman, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of ACM SIGMOD 2004, Paris, France (June 2004)
2. Atallah, M., Frikken, K.: Securely outsourcing linear algebra computations. In: Proc. of ASIACCS 2010, Beijing, China (April 2010)
3. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC 8(1), 119–152 (2005)
4. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. JACM 45(6), 965–981 (1998)
5. Cimato, S., Gamassi, M., Piuri, V., Sassi, R., Scotti, F.: Privacy-aware biometrics: Design and implementation of a multimodal verification system. In: Proc. of ACSAC 2008, Anaheim, CA, USA (December 2008)

6. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation design for efficient query execution over sensitive distributed databases. In: Proc. of ICDCS 2009, Montreal, Canada (June 2009)
7. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM TISSEC 13(3), 22:1–22:33 (2010)
8. Damiani, E., De Capitani Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of CCS 2003, Washington, DC, USA (October 2003)
9. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM TODS 35(2), 12:1–12:46 (2010)
10. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of ICDCS 2011, Minneapolis, MN, USA (June 2011)
11. Gamassi, M., Lazzaroni, M., Misino, M., Piuri, V., Sana, D., Scotti, F.: Accuracy and performance of biometric systems. In: Proc. of IMTC 2004, Como, Italy (May 2004)
12. Gamassi, M., Piuri, V., Sana, D., Scotti, F.: Robust fingerprint detection for access control. In: Proc. of RoboCare Workshop 2005, Rome, Italy (May 2005)
13. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. JACM 43(3), 431–473 (1996)
14. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE 2002, San Jose, CA, USA (February 2002)
15. Hacigümüs, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD 2002, Madison, WI, USA (June 2002)
16. Lazowska, E., Zahorjan, J., Graham, G., Sevcik, K.: Quantitative system performance: Computer system analysis using queueing network models. Prentice-Hall, Inc., Upper Saddle River (1984)
17. Lin, P., Candan, K.: Hiding traversal of tree structured data from untrusted data stores. In: Proc. of WOSIS 2004, Porto, Portugal (April 2004)
18. Olumofin, F., Goldberg, I.: Privacy-preserving queries over relational databases. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 75–92. Springer, Heidelberg (2010)
19. Sadeghi, A., Schneider, T., Winandy, M.: Token-based cloud computing. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 417–429. Springer, Heidelberg (2010)
20. Shmueli, E., Waisenberg, R., Elovici, Y., Gudes, E.: Designing secure indexes for encrypted databases. In: Proc. of IFIP DBSec 2005, Storrs, CT, USA (August 2005)
21. Wang, H., Lakshmanan, L.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB 2006, Seoul, Korea (September 2006)
22. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In: Proc of CCS 2008, Alexandria, VA, USA (October 2008)

Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-Based Protection Methods

Hannes Federrath[1], Karl-Peter Fuchs[1], Dominik Herrmann[1],
and Christopher Piosecny[2]

[1] Computer Science Department, University of Hamburg, Germany
[2] Dept. of Management Information Systems, University of Regensburg, Germany

Abstract. We propose a dedicated DNS Anonymity Service which protects users' privacy. The design consists of two building blocks: a broadcast scheme for the distribution of a "top list" of DNS hostnames, and low-latency Mixes for requesting the remaining hostnames unobservably. We show that broadcasting the 10,000 most frequently queried hostnames allows zero-latency lookups for over 80 % of DNS queries at reasonable cost. We demonstrate that the performance of the previously proposed Range Queries approach severely suffers from high lookup latencies in a real-world scenario.

1 Introduction

The Domain Name System (DNS), a globally distributed directory service, is mainly used to translate domain names (hostnames) to IP addresses. The bulk of the translation work is offloaded to DNS resolvers, which query the directory service on behalf of users. Unfortunately, the DNS protocol does not account for privacy. In fact, each DNS resolver has easy access to the IP addresses of its users and the domain names they are interested in. The upcoming DNSSEC protocol does not address in any way the confidentiality of DNS traffic, either. In fact, this was a "deliberate design choice" [3].

During the last years a "third-party ecosystem" for DNS services has evolved. Besides the ISPs there are many more providers offering DNS resolvers. The most popular providers are Google Public DNS and OpenDNS.[1] The DNS providers advertize higher availability, protection from phishing and drive-by-downloads, content filtering and higher performance. These services are also used to circumvent DNS-based censorship. The dissemination of alternative DNS servers has increased significantly during the last years according to figures published by OpenDNS: while they received 3 billion requests per day in September 2007[2], this number has increased to 30 billion by 2010[3].

[1] Homepages at http://code.google.com/speed/public-dns/ and http://opendns.com/

[2] http://www.opendns.com/about/announcements/49/

[3] http://blog.opendns.com/2011/01/24/2010-the-numbers-we-saw

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 665–683, 2011.

The benefits of public DNS servers come at a price: users must give up some privacy. DNS providers have access to all the DNS queries of their users, which may disclose their interests, relations and habits. Recent research results on user session re-identification [21, 29] also suggest that long-term profiling of users may be feasible solely based on the accessed hosts, enabling a malicious DNS resolver to monitor users over long periods of time and at different locations.

Previous research on privacy-enhancing DNS has not resulted in readily available systems so far. In this paper we aim for a practical and usable solution that allows users to access DNS resolvers privately, i. e., issue DNS queries without disclosing the desired hostnames to the DNS provider. As shown in [20] and [15] usability and especially low latency are crucial factors for the acceptance of Privacy Enhancing Technologies. Our solution addresses this challenge by trading in a little amount of additional traffic for significantly lower latencies.

Contributions. Firstly, we propose a DNS Anonymity Service that can improve privacy and performance at the same time through a combination of broadcast and Mixes. Using real-world DNS traffic dumps we demonstrate the practicability of our solution, which offers zero-latency and totally unobservable lookups for up to 80 % of DNS requests. Secondly, we provide an extensive analysis on the performance of the previously proposed Range Queries approach for real-world web traffic, showing that lookup latencies dominate overall performance.

The rest of this paper is structured as follows. In Section 2 we review related work, and we provide an overview of DNS in Section 3. We outline the architecture of our DNS Anonymity Service in Section 4. In Section 5 we present our broadcast scheme for frequently accessed domain names, before we discuss Mixes and Range Queries in Section 6. In Section 7 we present results from our trace-driven simulations before we conclude the paper in Section 8.

2 Related Work

Previous research efforts regarding privacy-preserving access to DNS servers have mainly focused on the concept of "Range Queries", which achieves privacy by hiding the queries of a client within a set of dummy queries. Zhao et al. [30] propose a random-set Range Query approach using a single DNS resolver. We will provide a detailed description in Section 6.2. Zhao et al. also propose an improved Range Query scheme [31] inspired by Private Information Retrieval [12]. Their improved scheme reduces the required bandwidth, but requires two non-collaborating DNS resolvers running non-standard DNS software. Although the authors suggest their schemes especially for web surfing applications, they fail to demonstrate their practicability using empirical results. In contrast, our study includes a performance evaluation using actual web traffic of a large user group and a concrete implementation of Range Queries. This allows us to assess the real-world performance of Zhao's Range Query proposal.

Castillo-Perez et al. [8, 9] study privacy issues of DNS in a different context, namely the ENUM protocol and the Object Naming Service (ONS). They propose a variation of the original Range Query scheme published by Zhao et al. in

[30] using multiple DNS resolvers in parallel. They implemented their proposal in order to evaluate its performance. Their results are of limited relevance for our scenario, though, as their evaluation setup does not resemble the effective DNS topology on the Internet.

Lu and Tsudik propose PPDNS [22], a privacy-preserving DNS system, which is also based on Range Queries, but uses a next-generation DNS infrastructure based on distributed hashtables and peer-to-peer technologies. While PPDNS is a promising approach, we do not expect that it will be widely adopted in the near future due to the need for a completely different DNS infrastructure and its high computational complexity, which requires special hardware.

We conclude that there is no readily available, practical solution for web users to protect their DNS queries, and the performance of the proposed Range Query schemes in real-world settings is unknown.

3 Overview of DNS and the Dataset

The Domain Name System is a distributed database which essentially maps domain names to IP adresses. On each client machine there is a *stub resolver*, a software component of the operating system, which relays DNS queries to the local nameserver. Local nameservers, which are also called *caching resolvers*, fetch the requested information *(Resource Records)* from the *authoritative nameservers* or – whenever possible – from their cache. Each Resource Record has a time-to-live (TTL) value, which indicates how long it can be cached by clients and caching resolvers. The original DNS protocol does not contain any security measures safeguarding integrity and privacy of messages. While integrity protection will become available with the adoption of DNSSEC [3], privacy of message contents and protection of the identity of clients are open problems.

3.1 Characteristics of DNS Traffic

To outline the most relevant characteristics of DNS traffic we summarize the main findings of two well-known studies from 2004 [6] and 2001 [19] and verify and complement them with more recent statistics derived from our 2010 dataset.

An important attribute of DNS traffic is its low traffic volume. Brandhorst et al. showed in their study that DNS packets are responsible for only 0.05 % of overall traffic [6]. In our logs the daily DNS traffic per user added up to about 120 KB with 33 KB for requests and 87 KB for replies. The average sizes of request and reply packets were 36 and 102 bytes respectively. However the low bandwidth requirement is not only a consequence of small request and reply sizes. It is also due to the fact that Resource Records can be cached according to their TTL by resolvers and clients. Jung et al. found that 60 % to 80 % of all requests could be answered using client-side caches [19].

We also found numerous *request bursts* in our logs, i. e., clients query for several hostnames with little or no delay between requests. Further analysis revealed two causes: (1) some websites embed media files from multiple domains, and (2) some web browsers pre-resolve all hostnames found in links on a site for

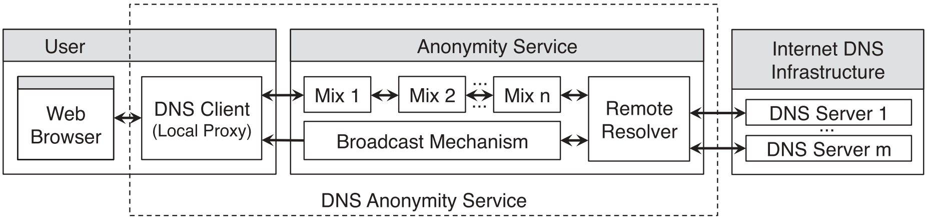


Fig. 1. Overview of the DNS Anonymity Service

performance reasons. Thus, visiting *www.wikipedia.org* may result in up to 150 DNS queries, as each country has its own domain (e. g., *de.wikipedia.org*) linked from the home page.

Another important characteristic of DNS traffic, which is beneficial for the efficiency of caching, is that the popularity of hostnames follows a power-law distribution [19]. A small set of popular hostnames is responsible for the vast majority of all queries, while the "long tail" of remaining hostnames is queried rarely. In the study of Jung et al. 68 % of requests affected the 10 % most popular hosts. We found in our traces that the 10 % most popular hostnames account for 97.7 % of the requests, and the 10,000 most popular hostnames account for 80.2 % of all requests.

3.2 Overview of the Dataset

We cooperated with the computer center of our university to retain a log of all DNS queries from the student housing subnet in pseudonymized form. The log file covers 159 days (from February to July 2010). During that time we observed 9,946,138 distinct hostnames from 4159 users in total. On average there were 2126 users active per day The original log file contains only DNS requests, but lacks information on the replies. Therefore, we issued recursive DNS queries for all hostnames to Google's DNS resolver. We recorded the size of the query and the reply packets as well as the *lookup latency.* We also obtained the TTL values for all hostnames by querying the respective authoritative nameservers. For CNAME Resource Records we followed the trail until an A record was returned and used the minimum of all observed TTL values to obtain the effective TTL. NXDOMAIN replies were handled according to [2].

4 DNS Anonymity Service

The DNS Anonymity Service (cf. Fig. 1) consists of four components, namely a DNS Client, a Mix Cascade, a Broadcast Mechanism and a DNS Resolver ("Remote Resolver"). The DNS Client is installed on the user's computer and acts like a regular DNS resolver towards the users' operating systems. The Remote Resolver is shared by the clients and looks up DNS entries with the help of the existing DNS infrastructure. Both, DNS Client and Remote Resolver employ caching of replies according to the TTL value to avoid redundant queries. Communication between DNS Client and Remote Resolver is protected by a Mix Cascade (cf. Section 6).

The reason for this design is twofold: On the one hand, we want the whole process of resolving to be transparent for the user. This enables users to keep their usual web browser and additional software. On the other hand, we want to avoid solutions that would require changes to the DNS infrastructure of the Internet.

Attacker model. As the network infrastructure of the Internet does not offer reliable broadcast, we assume that the broadcast messages are distributed consistently to all clients, e. g., by employing Byzantine-fault-tolerant protocols such as [10, 23].

The attacker model for our mix system resembles the attacker models of Tor [14] and JonDonym (formerly AN.ON [5]), two deployed mix-based anonymization systems for low-latency traffic. Specifically, we designed our system to protect against three types of *local attackers*, namely adversaries that control a single (entry or middle) mix or a single communication line (A_1) and adversaries that control an exit mix (A_2) or the DNS resolver (A_3).

We explicitly do not consider a global passive adversary (GPA) with access to all communication lines, as – at least in our web traffic scenario – a GPA can deanonymize users by eavesdropping on HTTP traffic anyway. Initially we set out to also include protection against adversaries controlling both, entry and exit mixes (A_4). The implementation presented in this paper does not protect against such distributed adversaries, though (for reasons explained in Section 6.1). We also do not consider attacks on the integrity of DNS replies by the exit mix or the DNS resolver. Such attacks can be detected by the client once DNSSEC is widely deployed. Finally, we assume that the attacker is computationally bounded and cannot break the cryptographic primitives used.

5 Broadcasting Popular DNS Records

The power-law characteristics of DNS traffic mentioned in Section 3.1 suggest that broadcasting the Resource Records of a small fraction of all hostnames might cover the vast majority of all user traffic. As the replies for the affected queries would be available to users immediately, this solution promises lower latencies than existing non-anonymity providing DNS resolvers. From a security point of view broadcasting is favorable as well since the affected queries can be answered locally with the Resource Records cached in the DNS Client. As a result, resolving of the affected queries becomes unobservable.

Despite the low traffic volume of the DNS protocol, it would be very inefficient and impractical to broadcast all records of the distributed DNS database to all clients due to the large number of registered domains and the long-tailed distribution of query names.[4] Therefore, we suggest a hybrid strategy. Combining broadcast for popular hosts with Mixes for the remaining hosts allows us to find a suitable trade-off between latency and bandwidth usage.

For this purpose, we define a complete ordered list H of all hostnames h_i, sorted by the total number of accesses in descending order. The list is split after θ elements, resulting in two sublists, $\mathrm{TopList}_\theta$ and $\mathrm{LongTail}_\theta$, i. e., $H = \mathrm{TopList}_\theta \cup \mathrm{LongTail}_\theta$ and $H = \mathrm{TopList}_\theta \cap \mathrm{LongTail}_\theta = \emptyset$. $h \in \mathrm{TopList}_\theta$ if $\mathrm{rank}(h) \leq \theta$, otherwise $h \in \mathrm{LongTail}_\theta$.

θ allows us to control the trade-off between latency and bandwidth usage, as it determines the number of hosts to be broadcast. To choose an adequate θ, the interdependent factors "cache hit ratio" (i. e., the percentage of requests affected by broadcast) and "bandwidth requirement" (i. e., the cumulative size of all DNS entries to be broadcast over time) must be considered. While the cache hit ratio is determined by the power-law distribution, bandwidth requirement is limited by the anonymity service's and clients' capacity. To improve the cache hit ratio, further DNS entries must be broadcast, what in turn results in increased bandwidth requirements. A more precise analysis of the interdependencies between both factors is given in Section 7.1.

5.1 Obtaining the Most Popular Hosts

As all the queries for hostnames from the TopList are answered from a local cache in the DNS Client, they are unobservable for the Anonymity Service. Consequently, it is challenging for the Anonymity Service to obtain and maintain the TopList. An obvious approach is to use global web statistics publicly available from companies like Alexa, Com-Score or NetRatings (Strategy 1). They do not represent the usage behavior of varying regionally dependent user groups due to their global focus, though.

A more promising approach is to use the statistics of another DNS resolver located in the same region as the anonymity service (Strategy 2). Opening the DNS Anonymity Service's Remote Resolver for public access (i. e., for users not interested in anonymization) might provide appropriate statistics as well. Alternatively, DNS cache probing [25, 1] can be employed to assemble the TopList.

To fit the TopList as closely as possible to the Anonymity Service's users, rescinding the unobservability property of the broadcast mechanism (i. e., which hosts where queried) is another option (Strategy 3). This should be achieved without revealing which individual user queried which hostnames, of course. With [26], [27] and [7] several well known protocols based on secure multiparty computation techniques exist to solve this challenge, but they suffer from high communicational and computational overhead.

A more pragmatic solution is client-side logging. Users could record the number of requests they were able to save for individual hosts due to broadcast of the TopList. If these user statistics were provided to the DNS Anonymity Service

[4] Given an estimate of 205.3 million domain names in Q4/2010 [28] and a size of 50 bytes for each Resource Record, a snapshot of the whole distributed DNS database would amount to more than 9.5 GB.

in regular intervals (e. g., once a week) via an anonymous channel, the TopList could be kept up to date. While the anonymous channel (e. g., provided by a Mix Cascade) could hide which statistics belong to which users, communication contents (i. e., the statistics themselves) would not be protected, rendering this approach less secure than the protocols mentioned above. As a result, linking user statistics could be possible by means of probabilistical profiling techniques such as [29, 21]. Their impact would be limited though, since linking user statistics with statistics derived from the Mix Cascade used to anonymize hostnames from the long tail is hardly possible, since TopList $\cap$ LongTail $= \emptyset$.

5.2 Realization of the Broadcast Mechanism

The broadcast mechanism consists of two parts: first of all, the DNS Anonymity Service must **refresh all entries in the TopList**, since DNS records retrieved from authoritative nameservers expire after a certain time. To this end we use a database containing an entry for each hostname that supplies the corresponding DNS record and a timestamp of its next expiration. A worker thread refreshes the records just before expiration using the Remote Resolver.

Secondly, the **TopList must be distributed to the clients**. Immediately after a client has established a connection to the DNS Anonymity Service, it receives a complete copy of the TopList. As long as the client is connected to the service, it receives a steady stream of incremental updates of the TopList. An update for a record is broadcast only, if the respective record has actually changed since the last update. Since according to [18] DNS records change rarely in comparison to their TTL values, the data volume of incremental updates is supposed to cause only little overhead. Additional reduction in bandwidth can be achieved with compression as pointed out in [18] as well. The broadcast mechanism in our prototype was implemented using TCP/IP unicast, i. e., the DNS Anonymity Service delivers the TopList within a dedicated TCP stream to each connected client. Efficiency could be increased using IP multicast [4].

6 Anonymizing the Long Tail

As outlined in Section 5 we broadcast only a small number of very popular domains in the TopList. Thus, clients need a means to resolve hostnames from the long tail without disclosing them to the resolver. In this paper we study the effectiveness and performance of Mixes and Range Queries for this purpose.

6.1 Mixes

A Mix is a cryptographic technique to enable untraceable communication introduced by David Chaum [11]. The basic idea is to route messages over several independent communication proxies (called Mixes), which hide the communication relationship between senders and receivers. Chaum introduced Mixes for asynchronous applications like electronic voting and e-mail, and he proposed to employ a hybrid cryptosystem using asymmetric and symmetric keys on a per-message basis. Pfitzmann et al. [24] and Goldschlag et al. [17] adapted this concept for real-time protocols that can handle a continuous stream of data with low latency. A client establishes a "channel" which can be used to send multiple consecutive messages through the Mixes. To this end the client establishes shared keys with every Mix using its asymmetric public key. The actual messages are encrypted using fast symmetric ciphers. As all messages transferred within the same channel are linkable, channels have to be switched regularly.

Chaum suggested to repeatedly collect messages until a certain threshold m is reached and only then deliver (flush) all m messages at once in different order to hide the true sender among all present senders. As DNS messages are quite small and most of them are quite similar in size, the application of such an output strategy seems feasible and also promising as it would allow for the construction of a mix system resisting end-to-end attacks. Accordingly, we chose to implement an unbiased, generic mix system instead of building on Tor or JonDonym, both of which are highly optimized for TCP and HTTP traffic and tailored to their respective network topologies. A new development in quite early stage is *ttdnsd*[5], the Tor TCP DNS daemon, which relays DNS queries via TCP to DNS resolvers. Including the official version, without further tuning, in our evaluation would not have allowed for a comparison on fair grounds, though. The then current version 0.7 caused high traffic overhead (queries and replies took up a full 512 byte cell each) and offered poor performance whenever the TCP connection had to be re-established after periods of inactivity.

Security Analysis. The attackers considered in Section 4 may undermine the protection of Mixes in various ways. A_1 may record message sizes of query and reply packets to infer the queried hostnames, exploiting characteristic patterns caused by individual websites. We can thwart this attack by padding packets to a common length. Learning all queried hostnames and being able to link consecutive queries within a channel, A_2 may carry out a user re-identification attack and link consecutive channels. We can decrease the probability of its success by using short-lived channels. While A_3 has access to queries, too, he cannot link queries originating from the same channel. A_2 and A_3 may detect the presence of a certain user based on unique, immediately identifying queries, which is out of the scope of our solution, though. A_4 may correlate timings of incoming and outgoing packets in order to totally deanonymize users. Foiling this attack requires dummy traffic and synchronous batching [24].

Implementation. Our Mix implementation is written in Java using non-blocking I/O operations and the Bouncy Castle crypto provider. In extensive experiments (not reported due to space limitations) we implemented and evaluated several output strategies, e. g., timed and threshold batches with and without dummy traffic, but – even for very small batch sizes – we could not achieve satisfying results in terms of latencies and overhead for any configuration. Thus, we resorted to forward incoming messages immediately, i. e., our mix system does not offer protection against A_4. The implemented channel setup and replay

[5] Code repository at https://gitweb.torproject.org/ioerror/ttdnsd.git

detection resemble JonDonym's mechanisms. Channels are established with an asymmetric cryptosystem (RSA 2048 bit keys) and switched every 60 seconds to limit the information available to A_2.

Requests and reply messages have fixed sizes to address A_1 and are structured as follows: (MAC [16 bytes], length [2], fragmentID [1], payload including padding [s]). For each mix a layer of encryption is applied using a symmetric cipher (AES, 128 bit keys, OFB mode). To find an acceptable trade-off between "message overhead" and the "number of fragmented packets", we analyzed the distribution (weighted by access frequency) of query and reply sizes. We determined $s_{\text{query}} = 57$ bytes and $s_{\text{reply}} = 89$ bytes fitting best. Once DNSSEC is widely deployed, we can change s accordingly.

Our implementation includes several straightforward optimizations, e. g., new channels are established in the background, Mixes use multiple threads to decrypt messages in parallel, and connections between Mixes are multiplexed.

6.2 Range Queries

Various Range Query schemes have been proposed for preserving the privacy of DNS queries (cf. Section 2). Their benefits include a security model that does not rely on the participation of other users and a simple topology, which does not depend on relaying packets over multiple hops. In the following we describe the Range Query scheme evaluated in this paper. It closely resembles the original scheme introduced by Zhao et al. and improved upon by Castillo-Perez et al. In contrast to the PPDNS scheme, which only operates on a DHT-based DNS infrastructure, it is suitable for the DNS infrastructure deployed today.

Each time a client queries a domain name d, it constructs a query set $Q(d)$, of size n, comprised of d and $n-1$ dummy domain names. The client queries the DNS resolver for each of the n names and receives n replies from the server, discarding all but the desired one. Previous work [30, 31, 8] suggests that the client should draw the dummies randomly and independently from a large database of domain names. Assuming that the resolver cannot distinguish the dummies from the desired queries, its chances to correctly guess the desired query are $p = 1/n$. In order to counter intersection attacks mentioned in [9], which can be carried out by an active adversary to uncover the desired hostname, the client uses the same set of dummies for retransmissions of failed queries.

Castillo-Perez et al. and Lu and Tsudik have evaluated the performance of prototypical implementations of their Range Query schemes. Their results are not applicable to our scenario, though, because they assume that all queries can be answered by the DNS resolver immediately, neglecting delays introduced by recursive lookups. In contrast to previous work, we do study the influence of lookup latencies, which may have a significant impact on the overall latencies of Range Queries.

Security Analysis. The security of range queries depends on the resolver being unable to tell apart dummies and actual queries. This assumption is challenged by two traffic analysis attacks that exploit the characteristics of the DNS traffic generated by web browsers, and which have not been studied previously. They allow a malicious resolver to reduce the effective size of the range query whenever consecutive queries are not independent from each other, e. g., for *query bursts*. Firstly, the resolver could mount a *semantic intersection attack* by searching for hostnames known to belong to the same site in consecutive ranges. Instead of randomly and independently sampling dummy hostnames, the ranges must be constructed using plausible sets of hostnames to foil such attacks. Effective protection against intersection attacks is a complex issue, which is outside of the scope of this paper, though, and left open for future work. Instead, we focus on the second traffic analysis attack: the resolver might be able to mount a *timing attack* to identify the dummy replies. If a client issues a range query as an immediate consequence of having processed the desired reply of a previous range query (e. g., when downloading embedded images served from various web servers), the secondary query may reach the resolver before all the replies belonging to the primary query have been received from the upstream DNS servers or sent to the client. Thus, the resolver may deduce that all the pending replies of the previous query are likely dummies. An active adversary could also maliciously send out the replies in a trickle to increase the effectiveness of the attack.

Implementation. For the purpose of evaluation we built a DNS Range Query client in Java. The client bundles up the Range Query into a single package, which is compressed using the zlib library, and sends it to the server component over a TCP socket. The server component resolves all queries in parallel using the Remote Resolver and returns them to the client. We have implemented two alternative strategies that aim to foil the *timing attack* mentioned above. With the *StallDesiredReply* strategy the client waits until all replies of a range query have been received and only then returns the desired answer to the caller. The client can also employ the *DelayConsecutiveQuery* strategy, i. e., return the desired reply to the caller immediately once it is available, but hold back consecutive range queries issued before the still-pending query has been fully processed. We will evaluate the two strategies in Section 7.3.

7 Evaluation

7.1 Broadcasting the TopList

In this section we evaluate the broadcast mechanism regarding cache hit ratio and required bandwidth, taking into consideration the interdependencies between both factors as outlined in Section 5. Our main goal is to quantify the trade-off between latency and bandwidth usage in order to choose an adequate θ. For our simulations we selected a 24h-sample from our dataset. As we focus on DNS queries issued by web browsers in this paper, we selected only type A queries (2,591,240 requests, i. e., 95.7 % of the sample).

Cache Hit Ratio. In a first experiment we examine the suitability of different sources the TopList can be obtained from. We use three different TopLists matching the scenarios described in Section 5. For evaluating Strategy 1 (the use of global web statistics) we derived domain names from the Alexa top one million hostlist. To mitigate the problems in terms of precision with this list, we retrieved the contained web sites using an automated Firefox script. The occuring DNS requests were recorded and combined to a new list with a cut-off after θ elements *(Global TopList)*. To analyze the second strategy we use the most popular hosts obtained from a proxy server used by 50 German schools *(Same Region TopList)*. For the third strategy we have determined the most popular hosts from our DNS dataset to simulate a top list matching user behavior perfectly *(Optimal TopList)*.

In the following we discuss the results obtained through our simulations with $\theta = 10000$. As expected the highest hit rate (83.94 %) was achieved with the *Optimal TopList*. Quite surprisingly the *Same Region TopList* provides comparable results (68.72 %). With only 41.32 % the *Global TopList* performs worst, as expected. Surprisingly, hit rates can be further improved using a client-side cache, which saves 15.72 % of requests. Apparently the caching strategies of user-side stub resolvers fail to exploit the full potential of caching.

In a second experiment we analyze the influence of θ on the cache hit ratio. We used the *Optimal TopList* with varying values for θ between 100 and 100,000. Hit rates from client-side caches were included in the simulation. With a TopList of 100 hosts a hit ratio of 40.02 % was achieved. For $\theta = 1000$ and $\theta = 10000$ the TopList can satisfy 63.94 % and 83.94 % of requests, respectively. Raising θ above 10,000 leads to minor improvements only. At $\theta = 100000$ a hit ratio of 94.54 % was achieved.

Required Bandwidth. We implemented the broadcast mechanism to measure its bandwidth requirements for varying values of θ using our *Optimal TopList*. Therefore, we set up a local instance of the BIND nameserver. We configured BIND to resemble the behavior of typical third-party resolvers by enabling the *minimal-responses* configuration directive. For experimental purposes we disabled BIND's internal cache and configured it to forward all queries immediately to the authoritative nameservers.

We analyze the traffic requirements separately from the perspective of the DNS Anonymity Service, which must continuously refresh its database, and from the perspective of a client of the service.

Traffic for refreshing the TopList. The traffic volume caused by refreshing the TopList database amounts to the traffic caused by DNS requests and replies issued by the DNS Anonymity Service whenever an entry expires. The traffic volume is independent of the number of clients. The daily traffic and the average number of queries per second are shown in Fig. 2 for varying values of θ. The figure indicates that the cost per additional hostname is constant up to $\theta = 2000$ and decreases slightly from there onwards. The daily traffic volume required for refreshing a TopList with $\theta = 10000$ is approximately 352.37 MB. On average the DNS Anonymity Service will have to issue 38.89 queries per second to keep all hostnames up to date. The majority of queries (and therefore traffic) pertains to a

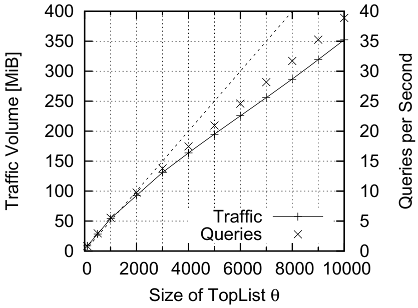


Fig. 2. Refreshing the TopList

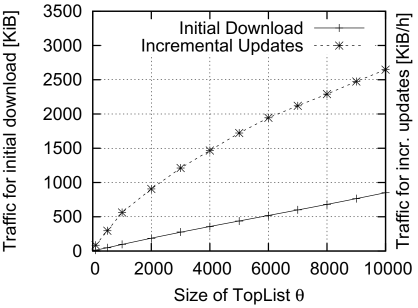


Fig. 3. Distribution to Clients

small fraction of hosts with TTL=60 (1,733 of 10,000 hostnames). In future work we will study optimizations such as enforcing a minimum TTL>60 seconds for all broadcast hostnames and advanced caching methods for round robin DNS replies that are used by many popular web sites for load balancing. Advanced schemes, which make authoritative servers push revocations or update notifications to resolvers, are also promising.

Traffic for distributing the TopList to the clients. The distribution of the TopList database to clients consists of two parts. Whenever a client connects to the DNS Anonymity Service, it receives a full copy of the TopList. After that it receives a steady stream of incremental updates. Using our measurement setup we determined the traffic volume for the initial download of the TopList and for receiving the incremental updates. The results are shown in Fig. 3. The initial download of the TopList amounts to 850 KB for $\theta = 10000$. On average the incremental updates cost 2.58 MB per hour and client (62.02 MB per day and client), which can be streamed with a bandwidth of less than 0.8 KB/s to each client. For 2000 connected clients, broadcasting the TopList consumes a bandwidth of 1.44 MB/s.

Further measurements indicate that the amount of traffic can be reduced considerably by compression. Using the zlib library, we reduced the initial download size on average by almost two thirds (290 KB for $\theta = 10000$), while the volume of the incremental updates was cut by roughly 40 % (to 1.5 MB/h). This finding matches the results in [18].

7.2 Trace-Driven Simulations

We evaluate our implementations of Mixes and Range Queries using trace-driven simulations. This approach allows us to study the effectiveness and performance under different loads induced by real users in a controlled environment. In each experiment we replay actual traffic from the log files in real-time to obtain statistics regarding bandwidth and latency.

In a pretest we found that experimental results stabilize already after a very short time. Thus, we randomly selected 10 chunks from the log file, each containing the traffic of a continuous two-hour period. For ease of exposition we will only provide results for one sample. We repeated the experiments with the remaining samples and validated the results presented in this section. The selected sample contains the DNS queries of 2082 users issued on April 20th, 2010 between 7.00 pm and 9.00 pm. Again, we selected queries of type A only. The resulting log file contains 465,435 requests for 193,133 distinct hostnames.

To allow for a fine-grained analysis of the latencies introduced by our system, we decided to start out neglecting network latencies and congestion, as both are known to dominate overall latencies in practical mix systems. Therefore, the first set of experiments was carried out in a local 1000 Mbit network (cf. Section 7.3). We dedicated a second set of experiments to the analysis of network latencies and congestion to study the expected real-world performance (cf. Section 7.4).

For both sets of experiments, we have implemented a DNS traffic simulator, which instrumented a number of DNS client processes (one per simulated user) according to the recorded traffic from the log file. The traffic simulator and the DNS client processes were running on a single machine. The Remote Resolver artificially delayed queries according to the *lookup latency* τ_l (see below) recorded in our dataset. For the evaluation of the Mix system we set up three Mix nodes, a common configuration also used by JonDonym and Tor, on three dedicated machines with a single DNS resolver on the last machine. Range Queries were evaluated using a DNS resolver with a thread pool of 1,500 workers running on a single machine. All machines were equipped with an Intel Core Duo 2.8 GHz CPU and 4 GB of RAM.

7.3 Performance Comparison of Mixes and Range Queries

In the following we provide the results of the trace-driven simulations for various configurations (first set of experiments). Client-side caches were enabled.

Reported Latencies. We model the *user-perceived latency* as $\tau = \tau_c + \tau_p + \tau_l$, i. e., it consists of the *client network latency* τ_c between the user's machine and the Anonymity Service, the *processing latency* τ_p within the Anonymity Service and the *lookup latency* τ_l for resolving the query at the Remote Resolver. In our experiments we determine the user-perceived latency by measuring the *difference between the time when the client sends the query and the time it receives the corresponding reply.* The reported latency values refer only to the queries that are relayed to the server component, i. e., local cache hits and requests for hostnames contained in the TopList are not included for clarity reasons. Including them would bring down the reported figures to 0 for most experiments. In fact user-perceived latency is 0 seconds for the majority of queries, if the TopList is enabled, of course (cf. Section 7.1).

Mixes. The results for four configurations using Mixes are shown in Fig. 4. Each boxplot shows the minimum latency, the percentile for 25 % the median and

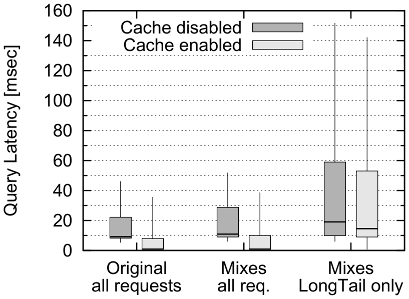


Fig. 4. Latency for Mixes

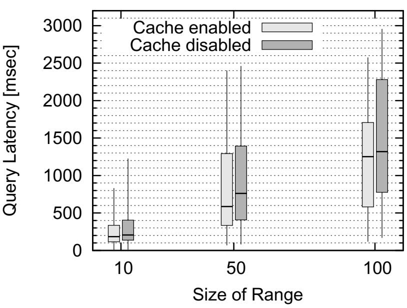


Fig. 5. Latency for Range Queries

the percentiles for 75 % and 90 %. The baseline configuration ("Original, cache disabled") shows the user-perceived latency, i. e., the distribution of τ_l, without our techniques. The median is 9.2 ms, and the 90 % percentile is 46.2 ms.

The configuration "Mixes all requests" consists of Mixes only (no TopList broadcasting, no caching on the Remote Resolver). The median increases slightly to 10.9 ms, and 90 % of the queries were answered within 52 ms.

Enabling the shared cache on the Remote Resolver ("Cache enabled") brings down latencies significantly: 75 % of the queries are answered within 10 ms. About 60 % of the requests scored cache hits in the Remote Resolver, which matches the findings in [19]. We found the majority of cache hits to be scored by hosts contained in the TopList. Therefore, cache effectiveness is expected to decrease once the TopList is enabled.

The configuration "Mixes LongTail only" shows the latencies observed for the hostnames contained in the LongTail set, i. e., for the queries remaining if the client has access to a TopList ($\theta = 10000$). Latencies are higher in this configuration as the average latency for hostnames from the TopList is considerably smaller (35.25 ms) than for all hostnames on overall (79.74 ms). As expected effectiveness of the cache on the Remote Resolver is limited in this scenario.

The results show that the overhead introduced by the cryptographic operations carried out by Mixes is small. User-perceived latencies will mainly depend on network latencies between clients and Mixes as well as on congestion effects. We study their influence in Section 7.4.

Range Queries. We also measured latency of Range Queries with range sizes $n = 10,\ 50,\ 100$ in our environment. Again, we neglect network delays for now.

The DNS Client creates ranges by randomly drawing dummies without replacement from the set of 193,133 hostnames contained in our dataset. This limitation is artificially introduced by the nature of our trace-driven simulation: we need to know τ_l for every hostname – and also for all possible dummy hostnames. In reality the dummies should be drawn from a much larger set, ideally from the set of all currently active hostnames on the Internet.

The overhead introduced by our Range Query implementation is negligible for isolated queries: we observed that τ_l of the desired query remained virtually unaffected for range sizes between 10 and 1000. In the following, we focus our analysis on the performance impact of the two strategies to counter the timing attack described in Section 6.2. Fig. 5 shows the perceived latency with the optimal TopList ($\theta = 10000$) and the *StallDesiredReply* strategy enabled. We observe that latencies are much higher than τ_l of the desired replies. Even for $n = 10$ 50 % of the requests take longer than 206 ms. For $n = 50$ and $n = 100$ the median is well above 500 ms and 1200 ms, respectively. Not a single Range Query could be fully answered from the cache in our experiments. The performance impact of this strategy is due to holding back the desired reply until the slowest (dummy) reply has been received by the client.

Chances of at least one slow query to be included are very high when they are drawn randomly from the whole population. Given a population of N hostnames containing αN "slow hostnames" with $\tau_l > T$, the probability that a randomly assembled range query of size n does contain at least one slow hostname can be obtained using the hypergeometric distribution:

$$P_T^n = P(X > 0) = 1 - P(X = 0) = 1 - \frac{\binom{\alpha N}{0}\binom{(1-\alpha)N}{n}}{\binom{N}{n}} = 1 - \frac{\binom{(1-\alpha)N}{n}}{\binom{N}{n}}$$

For $T = 200$ms we obtained $\alpha = 7\,\%$, i. e., $P_{200\text{ms}}^{10} \approx 0.516$ and $P_{200\text{ms}}^{100} \approx 0.999$, which explains the poor performance of the *StallDesiredReply* strategy. We expected the *DelayConsecutiveQuery* strategy, which achieves low latencies for singular queries at the cost of only delaying closely following queries, to achieve lower latencies on overall. The results indicate otherwise, though: even for small ranges ($n = 10$, cache on Remote Resolver disabled) the median is already 407 ms (90 % percentile: 10.45 s). Further analysis suggests that about 50 % of queries have to wait for their predecessors. In conclusion we find that, while a basic range query scheme may be fast, obscuring timing patterns of web browsers comes at a considerable cost.

7.4 Real World Latencies

In this section we present the results of the second set of experiments that aim at assessing the real world performance of our system by taking into account network latencies. We extend the experimental setup of the previous section by deploying WANem[6] delay boxes between network nodes, which are capable of simulating network latencies and congestion. To parameterize the delay boxes realistically we have cooperated with the JonDonym project, which kindly provided us with common network parameters derived from mixes actually deployed in their cascades across Europe (RTT between mixes: 20 ms; bandwidth of mixes: 100 Mbit). For client delay boxes we used a latency of 40 ms, which reflects the widespread RTT of about 80 ms for common ADSL connections.

[6] http://wanem.sourceforge.net/

Table 1. Effects of network delays and congestion on performance of Mix Cascade. The results for our trace show user-perceived latency ($\tau_c + \tau_p + \tau_l$), while the results for various synthetic loads with the given constant query rate include $\tau_c + \tau_p$ only.

percentile	our trace	synthetic	100	500	1000	2000	3000	4000	5000
50 %	171		139	139	141	245	342	527	1389
90 %	274		140	144	168	341	580	1544	7783

The resulting user-perceived latencies (τ) are shown in Table 1 for our trace-driven simulations with 2082 users (107 queries/s on average): in comparison to the measurements without delay boxes latencies for the 50 % (171 ms) and 75 % (274 ms) percentiles increase by 160 ms and 222 ms, respectively. Given that the sum of simulated network latencies is 120 ms, congestion effects are barely visible for this load. Total latencies are still low enough for practical usage.

To study the effects of congestion we induced synthetic traffic, i. e., we sent a constant number of queries per second (qps) to the cascade and measured latencies for different query rates (cf. Table 1). Until 1000 qps, little to no congestion is visible. Between 1000 qps and 3000 qps latencies start to increase noticeably with the load, although some users may find them still acceptable for practical usage. Above 3000 qps, effects of congestion are apparently dominating the performance of the cascade: latencies become unacceptably high.

A straightforward and scalable solution to prevent congestion in practice is to deploy multiple redundant mix cascades. As this would lead to splitted anonymity groups and may have a negative impact on privacy, the adoption of *Free Routes* or intermediate solutions like expander graphs [13] or stratified networks [16] would be worth consideration. As our current implementation is capable of handling a rather high number of participants already, we leave the study of further topologies to future work.

7.5 Traffic Overhead

We measure the traffic overhead by comparing the size of the original query and reply packets in our DNS dataset ("Original") with the traffic volume for Mixes and Range Queries. The resulting overhead is shown in Table 2. Due to message padding and multiple layers of encryption our mix system increases traffic by 99 %. This is considerably less than the overhead for Range Queries.

Interestingly, the overhead for Range Queries is not as high as expected. Without compression, traffic increases by only 583 % instead of the expected 900 % for $n = 10$ (not shown in table). This can be explained by the fact that dummies are drawn uniformly from the set of all hostnames. While the (by access frequency) weighted average size of the DNS replies issued by our users is 102 bytes, the average unweighted reply size is only 72 bytes. Even with compression the overhead is still 314 % for $n = 10$, though.

The table also indicates the traffic savings gained by the TopList for various values of θ. The columns labelled with "A" depict the overhead when traffic needed for refreshing the TopList is neglected, while the "B"-columns

Table 2. Traffic overhead relative to the original traffic volume for the two hour trace (compression enabled for Range Queries and refreshing the TopList)

	Original		Mix		RQ 10		RQ 50		RQ 100	
	A	B	A	B	A	B	A	B	A	B
No TopList	1	–	1.99	–	4.14	–	14.05	–	23.53	–
$\theta = 10000$	0.15	105.5	0.32	105.7	0.80	106.1	2.81	108.1	5.08	110.4
$\theta = 1000$	0.36	23.34	0.55	23.53	1.81	24.79	6.67	29.66	12.42	35.40
$\theta = 100$	0.63	4.10	0.83	4.30	3.01	6.48	11.10	14.57	20.66	24.13

incorporate this traffic. It is apparent from the numbers, that the overhead caused by refreshing the TopList significantly outweighs the remainder. We want to stress that the absolute traffic volume is still manageable, though: on average each user transferred 3245 KB (including 3096 KB for refreshing the TopList with $\theta = 10000$) in the Mix configuration within the two hours of simulation.

8 Conclusion

We proposed a DNS Anonymity Service, which combines broadcast with Mixes in order to trade in traffic volume for low latencies, which are critical for DNS. Our proposal exploits the power-law characteristics of DNS traffic to offer improvements on both, privacy and performance, at the same time. We found that broadcasting a small fraction of all hostnames enables unobservability for a large share of user traffic. Moreover, the broadcasted DNS responses are available to users immediately, i. e., faster than with common non-anonymous third-party DNS resolvers. This property of the DNS Anonymity Service may serve as an effective incentive to foster its adoption. Our broadcast component can also be used by conventional DNS providers, who want to offer superior performance.

Moreover, we have evaluated the applicability of Range Queries and Mixes for anonymizing the remaining hostnames with real traffic traces. We found that Range Queries offer poor performance, if dummy hosts are randomly drawn from a large set of hostnames and protection against timing attacks is desired, while Mixes do not introduce considerable delays apart from network latencies. Regarding privacy, a definitive comparison of the two systems is difficult to obtain, due to their different topology and techniques. The security of Mixes and their limitations is well-understood, enabling us to build a practical low-latency system. In light of attacks that exploit semantic interdependencies of queries, the security of Range Queries for web traffic seems much more fragile. It depends on a good source for dummy hostnames as well as a secure range construction scheme, both of which being fertile areas for future work.

Acknowledgements. We thank the anonymous reviewers, Jaideep Vaidya and Benedikt Westermann for their critical feedback. This work has been partially sponsored and supported by the European Regional Development Fund (ERDF).

References

[1] Akcan, H., Suel, T., Brönnimann, H.: Geographic Web Usage Estimation By Monitoring DNS Caches. In: Proceedings of the First International Workshop on Location and the Web, LOCWEB 2008, vol. 300, pp. 85–92. ACM, New York (2008)

[2] Andrews, M.: Negative Caching of DNS Queries (DNS NCACHE). RFC 2308 (1998)

[3] Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS Security Introduction and Requirements. RFC 4033 (2005)

[4] Armstrong, S., Freier, A., Marzullo, K.: Multicast Transport Protocol. RFC 1301 (1992)

[5] Berthold, O., Federrath, H., Köpsell, S.: Web MIXes: A System for Anonymous and Unobservable Internet Access. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 115–129. Springer, Heidelberg (2001)

[6] Brandhorst, C., Pras, A.: DNS: A Statistical Analysis of Name Server Traffic at Local Network-to-Internet Connections. In: EUNICE 2005: Networks and Applications Towards a Ubiquitously Connected World, pp. 255–270 (2006)

[7] Burkhart, M., Dimitropoulos, X.: Fast Privacy–Preserving Top–k Queries using Secret Sharing. In: Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN), pp. 1–7. IEEE, Los Alamitos (2010)

[8] Castillo-Perez, S., García-Alfaro, J.: Anonymous Resolution of DNS Queries. In: Chung, S. (ed.) OTM 2008, Part II. LNCS, vol. 5332, pp. 987–1000. Springer, Heidelberg (2008)

[9] Castillo-Perez, S., García-Alfaro, J.: Evaluation of Two Privacy–Preserving Protocols for the DNS. In: Proceedings of the Sixth International Conference on Information Technology: New Generations, pp. 411–416. IEEE Computer Society Press, Washington, DC, USA (2009)

[10] Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI 1999, pp. 173–186. USENIX Association, Berkeley (1999)

[11] Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)

[12] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, pp. 41–50. IEEE Computer Society, Los Alamitos (1995)

[13] Danezis, G.: Mix-Networks with Restricted Routes. In: Dingledine, R. (ed.) PET 2003. LNCS, vol. 2760, pp. 1–17. Springer, Heidelberg (2003)

[14] Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The Second–Generation Onion Router. In: Proceedings of the 13th USENIX Security Symposium, pp. 303–320. USENIX, Berkeley (2004)

[15] Dingledine, R., Serjantov, A., Syverson, P.F.: Blending Different Latency Traffic with Alpha-mixing. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 245–257. Springer, Heidelberg (2006)

[16] Dingledine, R., Shmatikov, V., Syverson, P.: Synchronous batching: From cascades to free routes. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 186–206. Springer, Heidelberg (2005)

[17] Goldschlag, D., Reed, M., Syverson, P.: Onion routing. Communications of the ACM 42(2), 39–41 (1999)

[18] Handley, M., Greenhalgh, A.: The case for pushing DNS. In: ACM Workshop on Hot Topics in Networking (Hotnets) (2005)

[19] Jung, J., Sit, E., Balakrishnan, H., Morris, R.: DNS Performance and the Effectiveness of Caching. IEEE/ACM Transactions on Networking (TON) 10(5), 589–603 (2002)

[20] Köpsell, S.: Low Latency Anonymous Communication – How Long Are Users Willing to Wait? In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 221–237. Springer, Heidelberg (2006)

[21] Kumpošt, M., Matyáš, V.: User Profiling and Re-identification: Case of University-Wide Network Analysis. In: Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G. (eds.) TrustBus 2009. LNCS, vol. 5695, pp. 1–10. Springer, Heidelberg (2009)

[22] Lu, Y., Tsudik, G.: Towards Plugging Privacy Leaks in the Domain Name System. In: Proceedings of the Tenth International Conference on Peer–to–Peer Computing (P2P), pp. 1–10. IEEE, Los Alamitos (2010)

[23] Pease, M., Shostak, R., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM 27, 228–234 (1980)

[24] Pfitzmann, A., Pfitzmann, B., Waidner, M.: ISDN-MIXes: Untraceable Communication with Very Small Bandwidth Overhead. In: Proc. GI/ITG-Conference Kommunikation in Verteilten Systemen (Communication in Distributed Systems), pp. 451–463 (1991)

[25] Rajab, M.A., Monrose, F., Provos, N.: Peeking Through the Cloud: Client Density Estimation via DNS Cache Probing. ACM Trans. Internet Technol. 10, 9:1–9:21 (2010)

[26] Vaidya, J., Clifton, C.: Privacy–Preserving Top–k Queries. In: Proceedings of the 21st International Conference on Data Engineering (ICDE), pp. 545–546. IEEE Computer Society, Los Alamitos (2005)

[27] Vaidya, J., Clifton, C.: Privacy–Preserving Kth Element Score over Vertically Partitioned Data. IEEE Trans. Knowl. Data Eng. 21(2), 253–258 (2009)

[28] Verisign Inc.: The Domain Name Industry Brief (February 2011), http://verisigninc.com/assets/domain-name-report-feb-2011.pdf

[29] Yang, Y.C.: Web user behavioral profiling for user identification. Decision Support Systems 49, 261–271 (2010)

[30] Zhao, F., Hori, Y., Sakurai, K.: Analysis of Privacy Disclosure in DNS Query. In: Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), pp. 952–957. IEEE Computer Society, Los Alamitos (2007)

[31] Zhao, F., Hori, Y., Sakurai, K.: Two–Servers PIR Based DNS Query Scheme with Privacy–Preserving. In: Proceedings of the The 2007 International Conference on Intelligent Pervasive Computing, pp. 299–302. IEEE Computer Society, Los Alamitos (2007)

Author Index