Dimitris Gritzalis
Bart Preneel
Marianthi Theoharidou (Eds.)

LNCS 6345

Computer Security – ESORICS 2010

15th European Symposium on Research in Computer Security
Athens, Greece, September 2010
Proceedings

Springer

Lecture Notes in Computer Science 6345

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

Editorial Board

Dimitris Gritzalis Bart Preneel
Marianthi Theoharidou (Eds.)

Computer Security – ESORICS 2010

15th European Symposium on Research in Computer Security
Athens, Greece, September 20-22, 2010
Proceedings

Springer

Volume Editors

Dimitris Gritzalis
Marianthi Theoharidou
Athens University of Economics and Business
Information Security and Critical Infrastructure Protection Research Group
Department of Informatics
76 Patission Ave., Athens, 10434, Greece
E-mail:{dgrit, mtheohar}@aueb.gr

Bart Preneel
Katholieke Universiteit Leuven
Department of Electrical Engineering-ESAT/COSIC
Kasteelpark Arenberg 10, Bus 2446, 3001 Leuven, Belgium
E-mail: bart.preneel@esat.kuleuven.be

Library of Congress Control Number: 2010933238

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

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-642-15496-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15496-6 Springer Berlin Heidelberg New York

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

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

ESORICS 2010 was the 15th edition of the event. It was held in Athens, Greece, September 20-22, 2010. The conference received 201 submissions. The papers went through a careful review process. In a first round, each paper received three independent reviews. For the majority of the papers an electronic discussion was also organized to arrive at the final decision. As a result of the review process, 42 papers were selected for the final program, resulting in an acceptance rate of as low as 21%. 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 Udo Helmbrecht, Executive Director of ENISA (European Network and Information Security Agency).

ESORICS 2010 was organized under the aegis of three Ministries of the Government of Greece, namely: (a) the Ministry of Infrastructure, Transport, and Networks, (b) the General Secretariat for Information Systems of the Ministry of Economy and Finance, and (c) the General Secretariat for e-Governance of the Ministry of Interior, Decentralization, and e-Government.

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 Sokratis Katsikas—our General Chair—for his kind encouragement, as well as to Nikos Kyrloglou—our Organizing Committee Cochair—for his continuous support. Our appreciation goes to Triaena Tours & Congress S.A., our local organizer and official travel agent, for our fruitful cooperation.

Last, but not least, we are sincerely grateful to our sponsor, Vodafone S.A., as well as to our supporters (in alphabetical order) Adacom S.A., Encode S.A., Ernst & Young S.A., Quality & Reliability S.A., and Unisystems S.A. for their kind and generous support.

Finally, we would like to thank the submitters, authors, presenters, and participants who, all together, made ESORICS 2010 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 towards an on-line world with adequate security.

September 2010

Dimitris Gritzalis
Bart Preneel
Marianthi Theoharidou

Organization

General Chair

Sokratis Katsikas	University of Piraeus (Greece)

Program Committee Chairs

Dimitris Gritzalis	Athens University of Economics and Business (Greece)
Bart Preneel	K. U. Leuven (Belgium)

Organizing Committee Chairs

Nikolaos Kyrloglou	Athens Chamber of Commerce and Industry (Greece)
Marianthi Theoharidou	Athens University of Economics and Business (Greece)

Publicity Chair

Sara Foresti	Università degli Studi di Milano (Italy)

Program Committee

Vijay Atluri	Rutgers University (USA)
Michael Backes	Saarland University and MPI-SWS (Germany)
Feng Bao	Institute for Infocomm Research (Singapore)
Joachim Biskup	University of Dortmund (Germany)
Carlo Blundo	Università di Salerno (Italy)
Xavier Boyen	Stanford University (USA)
Jan Camenisch	IBM Research Zurich (Switzerland)
Srdjan Capkun	ETH Zurich (Switzerland)
Richard Clayton	Cambridge University (UK)
Véronique Cortie	LORIA-CNRS (France)
Frédéric Cuppens	TELECOM Bretagne (France)
George Danezis	Microsoft Research (UK)
Sabrina de Capitani di Vimercati	Università degli Studi di Milano (Italy)
Claudia Diaz	K.U. Leuven (Belgium)
Simon Foley	University College Cork (Ireland)
Cédric Fournet	Microsoft Research (UK)

Deborah Frincke	Pacific Northwest National Laboratory (USA)
Dieter Gollmann	Hamburg University of Technology (Germany)
Thorsten Holz	Vienna University of Technology (Austria)
Bart Jacobs	University of Nijmengen (The Netherlands)
Sushil Jajodia	George Mason University (USA)
Tom Karygiannis	NIST (USA)
Stefan Katzenbeisser	T.U. Darmstadt (Germany)
Dogan Kesdogan	University of Siegen (Germany)
Aggelos Kiayias	University of Athens (Greece)
Michiharu Kudo	IBM Tokyo Research Laboratory (Japan)
Klaus Kursawe	Philips Research (The Netherlands)
Costas Lambrinoudakis	University of Piraeus (Greece)
Wenke Lee	Georgia Institute of Technology (USA)
Javier Lopez	University of Malaga (Spain)
Ioannis Mavridis	University of Macedonia (Greece)
Chris Mitchell	University of London (UK)
John Mitchell	Stanford University (USA)
Radia Perlman	Intel Corporation (USA)
Andreas Pfitzmann	T.U. Dresden (Germany)
Benny Pinkas	University of Haifa (Israel)
Michael Reiter	University of North Carolina (USA)
Peter Ryan	University of Luxembourg (Luxembourg)
Rei Safavi-Naini	University of Calgary (Canada)
Pierangela Samarati	Universitá degli studi Milano (Italy)
Einar Snekkenes	Gjovik University College (Norway)
George Spanoudakis	City University London (UK)
Ioannis Stamatiou	University of Ioannina (Greece)
Paul Syverson	Naval Research Laboratory (USA)
Bill Tsoumas	Athens University of Economics and Business (Greece)
Michael Waidner	IBM T.J. Watson Research Center (USA)
Dirk Westhoff	HAW Hamburg (Germany)

Additional Reviewers

Agudo, Isaac	Brinkman, Richard	Clauß, Sebastian
Ahmadi, Hadi	Buttyan, Levente	Cuppens-Boulahia, Nora
Alcaraz, Cristina	Chada, Rohit	D' Arco, Paolo
Anderson, Jonathan	Chadha, Rohit	De Caro, Angelo
Autrel, Fabien	Chan, Haowen	Deursen, van, Ton
Barati, Masoud	Chase, Melissa	Dobias, Jaromir
Batina, Lejla	Chen, Liqun	Doets, Peter Jan
Ben Ghorbel, Meriam	Chenette, Nathan	Dritsas, Stelios
Bonneau, Joseph	Clarkson, Michael	Drogkaris, Prokopis

Fan, Junfeng
Fernandez-Gago, Carmen
Fitzgerald, William
Gagne, Marin
Galindo, David
Garcia, Flavio
Garcia-Alfaro, Joaquin
Geneiatakis, Dimitris
Gierlichs, Benedikt
Gouglidis, Antonios
Gregoire, Benjamin
Hartog, den, Jerry
Hermans, Jens
Hoepman, Jaap-Henk
Hoffman, Johannes
Iovino, Vincenzo
Jarrous, Ayman
Jonker, Hugo
Köpsell, Stefan
Kellermann, Benjamin
Kirchner, Matthias
Konstantinou, Elisavet
Kontogiannis, Spyros
Laud, Peeter
Leh, Hoi
Li, Jiangtao
Li, Peng
Lochner, Jan Hendrik
Maes, Roel
Maffei, Matteo
Meier, Michael
Moran, Tal
Mostowski, Wojciech
Murdoch, Steven
Najera, Pablo
Narayan, Shivaramakrishnan
Nastou, Panayiotis
Nieto, Ana
Onieva, Jose A.
Oostendorp, Thom
Papagiannakopoulos, Panagiotis
Paskin-Cherniavsky, Anat
Poll, Erik
Reinman, Tzachy
Rekleitis, Evangelos
Rial, Alfredo
Rizomiliotis, Panagiotis
Roman, Rodrigo
Safa, Nashad Ahmad
Scafuro, Alessandra
Schiffner, Stefan
Shahandashti, Siamak
Song, Boyeon
Steel, Graham
Traore, Jacques
Troncoso, Carmela
Tuhin, Ashraful
Valeontis, Eytyhios
Vavitsas, Giorgos
Vercauteren, Frederik
Vergnaud, Damien
Visconti, Ivan
Vivas, Jose L.
Vrakas, Nikos
Wang, Pengwei
Yoshihama, Sachiko

Table of Contents

RFID and Privacy

Software Security

Cryptographic Protocols

Traffic Analysis

End-User Security

Formal Analysis

E-voting and Broadcast

Authentication, Access Control, Authorization and Attestation

Anonymity and Unlinkability

Network Security and Economics

Secure Update, DOS and Intrustion Detection

A New Framework for RFID Privacy*

Robert H. Deng[1], Yingjiu Li[1], Moti Yung[2], and Yunlei Zhao[3,**]

[1] Singapore Management University
[2] Google Inc. and Columbia University
[3] Software School, Fudan University
ylzhao@fudan.edu.cn

Abstract. Formal RFID security and privacy frameworks are fundamental to the design and analysis of robust RFID systems. In this paper, we develop a new definitional framework for RFID privacy in a rigorous and precise manner. Our framework is based on a zero-knowledge (ZK) formulation [8,6] and incorporates the notions of adaptive completeness and mutual authentication. We provide meticulous justification of the new framework and contrast it with existing ones in the literature. In particular, we prove that our framework is strictly stronger than the ind-privacy model of [18], which answers an open question posed in [18] for developing stronger RFID privacy models. We also clarify certain confusions and rectify several defects in the existing frameworks. Finally, based on the protocol of [20], we propose an efficient RFID mutual authentication protocol and analyze its security and privacy. The methodology used in our analysis can also be applied to analyze other RFID protocols within the new framework.

1 Introduction

Radio Frequency IDentification (RFID) tags are low-cost electronic devices, from which the stored information can be collected by an RFID reader efficiently (from tens to hundreds of tags per second) at a distance (from several centimeters to several meters) without the line of sight [25]. RFID technology has been widely used in numerous applications, ranging from manufacturing, logistics, transportation, warehouse inventory control, supermarket checkout counters, to many emerging applications [1]. As a key component of future ubiquitous computing environment, however, RFID technology has triggered significant concerns on its security and privacy as a tag's information can be read or traced by malicious readers from a distance without its owner's awareness [18,13,15,19,5,14].

It is critical to investigate formal RFID security and privacy frameworks that are fundamental to the design and analysis of robust RFID systems [18,3,26,23,10,21,20,22].

* The first author and the second author's work is partly supported by A*Star SERC Grant No. 082 101 0022 in Singapore. The first author's work is also partly supported by the Office of Research at Singapore Management University. The fourth author's work is partly supported by a grant from the Major State Basic Research Development (973) Program of China (No. 2007CB807901) and a grant from the National Natural Science Foundation of China NSFC (No. 60703091) and the QiMingXing Program of Shanghai.

** Contact author.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 1–18, 2010.

However, due to high system complexity, it turns out to be full of subtleties in developing rigorous and precise RFID system models. By examining the existing RFID system models, in this paper we develop a new definitional framework for RFID security and privacy in a rigorous and precise manner. Our framework is based on a zero-knowledge formulation [8,6], and incorporates the notions of adaptive completeness and mutual authentication. Compared to existing frameworks, our framework is more practical than those of [10,20], and is stronger in terms of privacy than those of [18,3]. Along the way, we also clarify certain confusions and rectify several defects in the existing frameworks.

To show how this new framework can be applied, we design an efficient RFID mutual authentication protocol based on the RFID protocol of [20] and analyze its security and privacy. The methodology used in our analysis is of independent interest and can be applied to analyze other RFID protocols within the new framework.

2 Preliminaries

If $A(\cdot,\cdot,...)$ is a randomized algorithm, then $y \leftarrow A(x_1, x_2, ...; \rho)$ means that y is assigned with the unique output of the algorithm A on inputs $x_1, x_2, ...$ and coins ρ, while $y \leftarrow A(x_1, x_2, ...)$ is a shorthand for first picking ρ at random and then setting $y \leftarrow A(x_1, x_2, ...; \rho)$. Let $y \leftarrow A^{O_1,...,O_n}(x_1, x_2, ...)$ denote that y is assigned with the output of the algorithm A which takes $x_1, x_2, ...$ as inputs and has oracle accesses to $O_1, ..., O_n$. If S is a set, then $s \in_R S$ indicates that s is chosen uniformly at random from S. If $x_1, x_2, ...$ are strings, then $x_1||x_2||\cdots$ denotes the concatenation of them. If x is a string, then $|x|$ denotes its bit length in binary code. If S is a set, then $|S|$ denotes its cardinality (i.e. the number of elements of S). Let Pr[E] denote the probability that an event E occurs, $\mathcal{N}$ denote the set of all integers, $\mathcal{R}$ denote the set of all real numbers.

A function $f : \mathcal{N} \rightarrow \mathcal{R}$ is said to be *negligible* if for every $c > 0$ there exits a number $m \in \mathcal{N}$ such that $f(n) < \frac{1}{n^c}$ holds for all $n > m$.

Given a security parameter κ, let $m(\cdot)$ and $l(\cdot)$ be two positive polynomials in κ. We say that $\{F_k : \{0,1\}^{m(\kappa)} \longrightarrow \{0,1\}^{l(\kappa)}\}_{k \in_R \{0,1\}^\kappa}$ is a pseudorandom function (PRF) ensemble according to the definition given in [7].

3 Model of RFID Systems

In this section, we first give a formal description of RFID system setting and adversary. We then define RFID systems to be "complete" in term of *adaptive completeness*, and "sound" in terms of *mutual authentication*.

3.1 RFID System Setting

Consider an RFID system comprising of a single legitimate reader R and a set of ℓ tags $\mathcal{T} = \{\mathcal{T}_1, ..., \mathcal{T}_\ell\}$, where ℓ is a polynomial in a security parameter κ. The reader and the tags are probabilistic polynomial time (PPT) interactive Turing machines. The RFID system $(R, \mathcal{T})$ is setup by a procedure, denoted Setup(κ, ℓ). Specifically, on (κ, ℓ), this setup procedure generates the public system parameter σ_R, the reader secret-key k_R

and initial internal state s_R^1 (if needed) for R. It may also setup an initial database DB^1 for R to store necessary information for identifying and authenticating tags. For each i, $1 \leq i \leq \ell$, this procedure generates the public parameter $\xi_{\mathcal{T}_i}$ and the initial secret-key $k_{\mathcal{T}_i}^1$ for a tag $\mathcal{T}_i$ and sets the tag's initial internal state $s_{\mathcal{T}_i}^1$ (typically, $s_{\mathcal{T}_i}^1$ includes the public parameters $\sigma_R, \xi_{\mathcal{T}_i}$). It may also associate the tag $\mathcal{T}_i$ with its unique ID, as well as other necessary information such as tag key and/or tag state information, as a record in the initial database DB^1 of R. Note that $\xi_{\mathcal{T}_i}$ or/and $s_{\mathcal{T}_i}^1$ can be empty strings.

We use $para = (\sigma_R, \xi_1, \cdots, \xi_\ell)$ to denote the public system parameters. We assume that in the RFID system, the reader is secure; in other words, the legitimate reader is a "black-box" to an adversary.

A tag $\mathcal{T}_i$, $1 \leq i \leq \ell$, exchanges messages with the reader R through a protocol $\pi(R, \mathcal{T}_i)$. Without loss of generality, we assume the protocol run of π is always initiated by R and π consists of $2\gamma + 1$ rounds for some $\gamma \geq 1$. Each protocol run of π is called a session. We assume each tag interacts with the reader sequentially, but multiple tags can interact with the reader "concurrently" (with some anti-collision protocols [27]). To allow and distinguish concurrent sessions (at the side of the reader R), we associate each session of protocol π with a unique session identifier sid. In practice, sid is typically generated by the reader when it is invoked to send the first-round message. We assume each message from a tag to the reader always bears the corresponding session-identifier.

Each tag $\mathcal{T}_i$, as well as the reader R, uses fresh and independent random coins (generated on the fly) in each session, *in case it is an randomized algorithm*. We assume that the random coins used in each session are erased once the session is completed (whether successfully finished or aborted). Also, in each session run, the tag may update its internal state and secret-key, and the reader may update its internal state and database. We assume that the update process of new internal state and secret-key by an uncorrupted tag automatically overwrites (i.e., erases) its old internal state and secret-key.

Given a security parameter κ, we assume that each tag $\mathcal{T}_i$ takes part in at most s (sequential) sessions in its life time with R, and thus R involves at most $s\ell$ sessions, where s is some polynomial in κ. In practice, the value s can be a fixed constant (e.g., $s = 2^{28}$ [1]).

More precisely, for the j-th session (ordered by the session initiation time) where $1 \leq j \leq s\ell$, the reader R takes the input from the system parameters $para$, its secret-key k_R, current internal state s_R^j, database DB^j, random coins ρ_R^j, and a partial transcript T, where T is either an empty string (which indicates the starting of a new session) or a sequence of messages $(sid, c_1, \alpha_1, c_2, \alpha_2, \cdots, c_u, \alpha_u)$, $1 \leq u \leq \gamma$ (which indicates the on-going of session sid). The reader R outputs the next message c_{u+1}. In the case of $T = (sid, c_1, \alpha_1, c_2, \alpha_2, \cdots, c_\gamma, \alpha_\gamma)$, besides sending back the last-round message $c_{\gamma+1}$, the reader R also updates its internal state to s_R^{j+1}, its database to DB^{j+1}, and stops the session by additionally outputting a bit, denoted by o_R^{sid}. This output bit indicates either acceptance ($o_R^{sid} = 1$) or rejection ($o_R^{sid} = 0$) of the current session.

Without loss of generality, we assume that the j-th session run by the reader R corresponds to the v-th session (of session-identifier sid) run by tag $\mathcal{T}_i$, where $1 \leq v \leq s$ and $1 \leq i \leq \ell$. In this session, $\mathcal{T}_i$ takes the input from the system parameters $para$, its current secret-key $k_{\mathcal{T}_i}^v$, current internal state $s_{\mathcal{T}_i}^v$, random coins $\rho_{\mathcal{T}_i}^v$, and a partial transcript $T = (sid, c_1, \alpha_1, \cdots, \alpha_{u-1}, c_u)$, where $1 \leq u \leq \gamma$. The tag $\mathcal{T}_i$ outputs the

next message (sid, α_u). In the case of $T = (sid, c_1, \alpha_1, \cdots, c_\gamma, \alpha_\gamma, c_{\gamma+1})$ (i.e., $\mathcal{T}_i$ has received the last-round message of the session sid), $\mathcal{T}_i$ updates its internal state to $s_{\mathcal{T}_i}^{v+1}$, its secret-key to $k_{\mathcal{T}_i}^{v+1}$, and stops the session by additionally outputting a bit, denoted by $o_{\mathcal{T}_i}^{sid}$. This output bit indicates either acceptance ($o_{\mathcal{T}_i}^{sid} = 1$) or rejection ($o_{\mathcal{T}_i}^{sid} = 0$) of the current session run by $\mathcal{T}_i$.

Note that in the above description, it is assumed that the reader and tags update their internal states, database, or keys *at the end of each protocol run*. In reality, this can be performed at any point of each protocol run. Also, for RFID protocol π with unidirectional authentication from tag to reader, the tag may not have a session output. In this case, the session output $o_{\mathcal{T}_i}^{sid}$ is set to "0" always.

3.2 Adversary

After an RFID system $(R, \mathcal{T})$ is setup by invoking Setup(κ, ℓ), we model a probabilistic polynomial-time concurrent man-in-the-middle (CMIM) adversary $\mathcal{A}$ against $(R, \mathcal{T})$, with adaptive tag corruption. We use $\hat{m}$ to denote a message sent by adversary $\mathcal{A}$, and m to denote the actual message sent by reader R or an uncorrupted tag. The adversary is given access to the following oracles:

InitReader(): $\mathcal{A}$ invokes R to start a session of protocol π and generate the first-round message c_1 which is also used as the session identifier sid. Supposing that the new session is the j-th session run by R, the reader R stores c_1 into its internal state s_R^j, and returns c_1 to the adversary.

SendT($\mathcal{T}_i, \hat{m}$): Adversary $\mathcal{A}$ sends $\hat{m}$ to $\mathcal{T}_i$. (Here, for simplicity, we abuse the notation $\mathcal{T}_i$ to denote any virtual identity of a tag in $\mathcal{T}$ (not the tag's real identity) labeled by $\mathcal{A}$ when $\mathcal{A}$ selects the tag from $\mathcal{T}$.) After receiving $\hat{m}$, $\mathcal{T}_i$ works as follows: (1) If $\mathcal{T}_i$ currently does not run any existing session, $\mathcal{T}_i$ initiates a new session with the session-identifier sid set to $\hat{m}$, treats $\hat{m}$ as the first-round message of the new session, and returns the second-round message (sid, α_1). (2) If $\mathcal{T}_i$ is currently running an incomplete session with session-identifier $sid = \hat{c}$, and is waiting for the u-th message from R, where $u \geq 2$, $\mathcal{T}_i$ works as follows: If $2 \leq u \leq \gamma$, it treats $\hat{m}$ as the u-th message from the reader and returns the next round message (sid, α_u). If $u = \gamma + 1$ (i.e., $\mathcal{T}_i$ is waiting for the last-round message of the session sid), $\mathcal{T}_i$ returns its output $o_{\mathcal{T}_i}^{sid}$ to the adversary, and (internally) updates its internal state to $s_{\mathcal{T}_i}^{v+1}$, assuming that the session sid is the v-th session run by $\mathcal{T}_i$, where $1 \leq v \leq s$.

SendR($\widehat{sid}, \hat{\alpha}$): Adversary $\mathcal{A}$ sends $(\widehat{sid}, \hat{\alpha})$ to R. After receiving $(\widehat{sid}, \hat{\alpha})$, R checks from its internal state whether it is running a session of session identifier $sid = \widehat{sid}$, and works as follows: (1) If R is currently running an incomplete session with $sid = \widehat{sid}$ and is waiting for the u-th message from a tag, where $1 \leq u \leq \gamma$, R acts as follows: If $u < \gamma$, it treats $\hat{\alpha}$ as the u-th message from the tag, and returns the next round message c_{u+1} to $\mathcal{A}$. If $u = \gamma$, it returns the last-round message $c_{\gamma+1}$ and the output o_R^{sid} to $\mathcal{A}$, and internally updates its internal state to s_R^{j+1} and the database to DB^{j+1}, assuming that the session sid corresponds to the j-th session run by R. (2) In all other cases, R returns a special symbol $\perp$ (indicating invalid query).

Corrupt($\mathcal{T}_i$): Adversary $\mathcal{A}$ obtains the secret-key and internal state information (as well as the random coins) currently held by $\mathcal{T}_i$. Once a tag $\mathcal{T}_i$ is corrupted, all its actions are controlled and performed by the adversary $\mathcal{A}$.

Let O_1, O_2, O_3 and O_4 denote the above oracles, respectively. These oracles fully capture the capability of any PPT CMIM adversary with adaptive tag corruption. (Here, for simpler definitional complexity, we assume all tags are always within the attack scope of adversary. In practice, some tags may be in or out from the attack scope of adversary at different time [26].) For presentation simplicity, we denote by $\mathcal{O}$ the set of the four oracles $\{O_1, O_2, O_3, O_4\}$ specified above. *An adversary is a* (t, n_1, n_2, n_3, n_4)*-adversary, if it works in time* t *and makes oracle queries to* O_μ *without exceeding* n_μ *times, where* $1 \leq \mu \leq 4$. We treat each oracle call as a unit operation, and thus for a t-time adversary it holds that $\Sigma_{\mu=1}^{4} n_\mu \leq t$. We denote by $A^{\mathcal{O}}(R, \mathcal{T}, para)$ a PPT algorithm A that, on input of some system public parameter $para$, concurrently interacts with R and the tags in $\mathcal{T}$ via the four oracles in $\mathcal{O}$, where $(R, \mathcal{T})$ is setup by Setup(κ, ℓ).

Note that in our formulation, the output bits of protocol participants (which indicate authentication success or failure) are *publicly* accessible to the adversary. The reason is that, in reality, such outputs can be publicly observed from the behaviors of protocol participants during/after the protocol run or can be learnt by some other side channels.

3.3 Adaptive Completeness and Mutual Authentication

Roughly speaking, adaptive completeness says that, after any attacks (*particularly the desynchronizing attacks*) made by the adversary $\mathcal{A}$, the protocol execution between the reader R and any honest uncorrupted tag is still complete (e.g., being able to recover from desynchronization). In other words, after undergoing arbitrary attacks, the uncorrupted parties of the RFID system still can recover *whenever the attacks stop*.

Definition 3.1 (adaptive completeness). *For an RFID system* $(R, \mathcal{T})$ *setup by* Setup (κ, ℓ)*, denote by*

$$(sid, c_1^{sid}, \alpha_1^{sid}, \cdots, \alpha_\gamma^{sid}, c_{\gamma+1}^{sid}, o_R^{sid}, o_{\mathcal{T}_i}^{sid}) \leftarrow \pi(R, \mathcal{T}_i)$$

the running of a session with identifier sid *of the protocol* π *between* R *and an uncorrupted tag* $\mathcal{T}_i \in \mathcal{T}$*. Suppose that the session* sid *corresponds to the* v*-th session at the side of* $\mathcal{T}_i$ *and the* j*-th session at the side of* R*, where* $1 \leq v \leq s$ *and* $1 \leq j \leq s\ell$*. Consider the case that the two sessions are of the same round messages, and that all the exchanged messages in these two (matching) sessions are honestly generated by* R *and* $\mathcal{T}_i$ *respectively. Denote by* E *the event that* $o_R^{sid} = 0$ *holds (or* $o_{\mathcal{T}_i}^{sid} = 0$ *holds if the protocol* π *is for mutual authentication) or* R *identifies a different tag* $\mathcal{T}_{i'} \neq \mathcal{T}_i$ *in its* j*-th session.*

A PPT CMIM adversary $\mathcal{A}$ $(t, \epsilon, n_1, n_2, n_3, n_4)$*-breaks the adaptive completeness of the RFID system against the uncorrupted* $\mathcal{T}_i$*, if the probability that event* E *occurs is at least* ϵ *and* $\mathcal{A}$ *is a* (t, n_1, n_2, n_3, n_4)*-adversary. The probability is taken over the coins used by* Setup(κ, ℓ)*, the coins of* $\mathcal{A}$*, the coins used by* R *(*up to finishing the j-th session*), and the coins used by* $\mathcal{T}_i$ *(*up to finishing the v-th session*). An RFID system* $(R, \mathcal{T})$ *satisfies adaptive completeness, if for all sufficiently large* κ *and for any uncorrupted tag* $\mathcal{T}_i$*, there exists no adversary* $\mathcal{A}$ *that can* $(t, \epsilon, n_1, n_2, n_3, n_4)$*-break the adaptive completeness against* $\mathcal{T}_i$*, for any* (t, ϵ)*, where* t *is polynomial in* κ *and* ϵ *is non-negligible in* κ*.*

Next, we define mutual authentication of RFID protocols. Roughly speaking, for a protocol π of the RFID system $(R, \mathcal{T})$, authentication from reader to tag (resp., from tag to reader) means that a CMIM adversary $\mathcal{A}$ cannot impersonate the reader R (resp., an uncorrupted tag $\mathcal{T}_i \in \mathcal{T}$) to an uncorrupted tag $\mathcal{T}_i \in \mathcal{T}$ (resp., reader R), unless $\mathcal{A}$ honestly relays messages actually generated and sent by R and the uncorrupted tag $\mathcal{T}_i$. Before we define mutual authentication for RFID protocols, we first clarify the notion of matching sessions.

Definition 3.2 (matching sessions). *Denote by* $(sid, c_1^{sid}, \alpha_1^{sid}, \cdots, \alpha_\gamma^{sid}, c_{\gamma+1}^{sid})$ *the transcript of exchanged round messages (except the session outputs) of a* successfully completed *session* sid *of the protocol* π *run by a tag* $\mathcal{T}_i$*, where* $1 \leq i \leq \ell$*. This session has a matching session at the side of the reader* R*, if* R *ever successfully completed a session of the identical session transcript.*

Denote by $(sid', c_1^{sid'}, \alpha_1^{sid'}, \cdots, \alpha_\gamma^{sid'}, c_{\gamma+1}^{sid'})$ *the transcript of exchanged round messages (except the session outputs) of a* successfully completed *session* sid' *run by* R*. This session has a matching session at the side of some tag* $\mathcal{T}_i$*, where* $1 \leq i \leq \ell$*, if either of the following conditions holds:*

- $\mathcal{T}_i$ *ever* completed*, whether successfully finished or aborted, a session of the identical transcript* prefix $(sid', c_1^{sid'}, \alpha_1^{sid'}, \cdots, \alpha_\gamma^{sid'})$*;*
- *Or,* $\mathcal{T}_i$ *is now running a session with partial transcript* $(sid', c_1^{sid'}, \alpha_1^{sid'}, \cdots, \alpha_\gamma^{sid'})$ *and is waiting for the last-round message of the session* sid'*.*

The matching-session definition, for a successfully completed session run by the reader R, takes into account the following "cutting-last-message" attack: a CMIM adversary $\mathcal{A}$ relays the messages being exchanged by R and an uncorrupted tag $\mathcal{T}_i$ for a protocol run of π until receiving the last-round message $c_{\gamma+1}^{sid'}$ from R; after this, $\mathcal{A}$ sends an arbitrary message $\hat{c}_{\gamma+1}^{sid'} (\neq c_{\gamma+1}^{sid'})$ to $\mathcal{T}_i$ (which typically causes $\mathcal{T}_i$ to abort the session), or, just drops the session at the side of $\mathcal{T}_i$ without sending $\mathcal{T}_i$ the last-round message. Such "cutting-last-message" attacks are unpreventable.

Figure 1 shows the authentication experiment $\mathbf{Exp}_{\mathcal{A}}^{auth}[\kappa, \ell]$. A CMIM adversary $\mathcal{A}$ interacts with R and tags in $\mathcal{T}$ via the four oracles in $\mathcal{O}$; At the end of the experiment, $\mathcal{A}$ outputs the transcript, $trans$, of a session. Denote by E_1 the event that $trans$ corresponds to the transcript of a successfully completed session run by R in which R successfully identifies an *uncorrupted* tag $\mathcal{T}_i$, but this session has no matching session at the side of the uncorrupted tag $\mathcal{T}_i$. Denote by E_2 the event that $trans$ corresponds to the transcript of a successfully completed session run by some *uncorrupted* tag $\mathcal{T}_i \in \mathcal{T}$, and this session has no matching session at the side of R.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{auth}[\kappa, \ell]$
1. run $\mathsf{Setup}(\kappa, \ell)$ to setup the reader R and a set of tags $\mathcal{T}$; denote by $para$ the public system parameters;
2. $trans \leftarrow \mathcal{A}^{\mathcal{O}}(R, \mathcal{T}, para)$.

Fig. 1. Authentication Experiment

Definition 3.3 (authentication). *On a security parameter κ, an adversary $\mathcal{A}$ $(\epsilon, t, n_1, n_2, n_3, n_4)$-breaks the authentication of an RFID system $(R, \mathcal{T})$ against the reader R (resp., an uncorrupted tag $\mathcal{T}_i \in \mathcal{T}$) if the probability that event E_1 (resp., E_2) occurs is at least ϵ and $\mathcal{A}$ is a (t, n_1, n_2, n_3, n_4)-adversary.*

The RFID system $(R, \mathcal{T})$ satisfies tag-to-reader authentication (resp., reader-to-tag authentication), if for all sufficiently large κ there exists no adversary $\mathcal{A}$ that can $(\epsilon, t, n_1, n_2, n_3, n_4)$-break the authentication of $(R, \mathcal{T})$ against the reader R (resp., any uncorrupted tag $\mathcal{T}_i \in \mathcal{T}$), for any (t, ϵ), where t is polynomial in κ and ϵ is non-negligible in κ. An RFID system is of mutual authentication, if it satisfies both tag-to-reader authentication and reader-to-tag authentication.

4 Zero-Knowledge Based RFID Privacy

In this section, we present a zero-knowledge based definitional framework for RFID privacy. To make our definition formal, we need to clarify the notion of blind access to tags and the notion of clean tags.

Let $A^{\mathcal{O}}(R, \widehat{\mathcal{T}}, \mathcal{I}(\mathcal{T}_g), aux)$ be a PPT algorithm A that, on input $aux \in \{0,1\}^*$ (typically, aux includes the system parameters or some historical state information of A), concurrently interacts with R and a set of tags $\widehat{\mathcal{T}}$ via the four oracles $\mathcal{O} = \{O_1, O_2, O_3, O_4\}$. We say that A has *blind access* to a *challenge* tag $\mathcal{T}_g \notin \widehat{\mathcal{T}}$ if A interacts with $\mathcal{T}_g$ via a special interface $\mathcal{I}$. Specifically, $\mathcal{I}$ is a PPT algorithm that runs $\mathcal{T}_g$ internally, and interacts with A externally. To send a message $\hat{c}$ to $\mathcal{T}_g$, A sends to $\mathcal{I}$ a special O_2 oracle query of the form $\mathsf{SendT}(challenge, \hat{c})$; after receiving this special O_2 query, $\mathcal{I}$ invokes $\mathcal{T}_g$ with $\mathsf{SendT}(\mathcal{T}_g, \hat{c})$, and returns to A the output by $\mathcal{T}_g$. From the viewpoint of A, it does not know which tag it is interacting with. It is also required that A interacts with $\mathcal{T}_g$ via O_2 queries only.

Next, we define the notion of clean tags. A tag $\mathcal{T}_i$ is called *clean*, if it is not corrupted (i.e., the adversary has not made any O_4 query to $\mathcal{T}_i$), and is not currently running an incomplete session with the reader (i.e., the last session of the tag has been either finished or aborted). In other words, a clean tag is an uncorrupted tag that is currently at the status of waiting for the first-round message from the reader to start a new session.

Now, we are ready to give a formal definition of zero-knowledge based RFID privacy (zk-privacy, for short). Figure 2 (page 8) illustrates the real world of the zk-privacy experiment, $\mathbf{Exp}_{\mathcal{A}}^{zkp}[\kappa,\ell]$ ($\mathbf{Exp}_{\mathcal{A}}^{zkp}$, for simplicity), in which a PPT CMIM adversary $\mathcal{A}$ is comprised of a pair of algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ and runs in two stages. In the *first stage*, algorithm $\mathcal{A}_1$ is concurrently interacting with R and all the tags in $\mathcal{T}$ via the four oracles in $\mathcal{O}$, and is required to output a set $\mathcal{C}$ of *clean* tags at the end of the first stage, where $\mathcal{C} \subseteq \mathcal{T}$ consists of δ *clean* tags, denoted as $\{\mathcal{T}_{i_1}, \cdots, \mathcal{T}_{i_\delta}\}$. The algorithm $\mathcal{A}_1$ also outputs a state information st, which will be transmitted to algorithm $\mathcal{A}_2$. Between the first stage and the second stage, a challenge tag, denoted as $\mathcal{T}_g$, is taken uniformly at random from $\mathcal{C}$. Note that if $\delta = 0$, then no challenge tag is selected, and $\mathcal{A}$ is reduced to $\mathcal{A}_1$ in this experiment. In the *second stage*, on input st, $\mathcal{A}_2$ concurrently interacts with the reader R and the tags in $\widehat{\mathcal{T}} = \mathcal{T} - \mathcal{C}$ via the four oracles in $\mathcal{O}$, and additionally has blind access to $\mathcal{T}_g$. Note that $\mathcal{A}$ cannot corrupt any tag (particularly $\mathcal{T}_g$) in $\mathcal{C}$, and $\mathcal{A}$ does not have access to tags in $\mathcal{C} - \{\mathcal{T}_g\}$ in the second stage. Finally, $\mathcal{A}_2$ outputs its

view, denoted by $view_{\mathcal{A}}$, at the end of the second stage. Specifically, $view_{\mathcal{A}}$ is defined to include the system public parameters $para$, the random coins used by $\mathcal{A}$, $\rho_{\mathcal{A}}$, and the (ordered) list of all oracle answers to the queries made by $\mathcal{A}$ in the experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$. Note that $view_{\mathcal{A}}$ does not explicitly include the oracle queries made by $\mathcal{A}$ and $\mathcal{A}$'s output at the first stage, as all these values are implicitly determined by the system public parameter $para$, $\mathcal{A}$'s coins and all oracle answers to $\mathcal{A}$'s queries. The output of experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$ is defined to be $(g, view_{\mathcal{A}})$. Denote by $(g, view_{\mathcal{A}}(\kappa, \ell))$ the random variable describing the output of experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}[\kappa, \ell]$.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}[\kappa, \ell]$
1. run $\mathsf{Setup}(\kappa, \ell)$ to setup the reader R and a set of tags $\mathcal{T}$; denote by $para$ the public system parameter;
2. $\{\mathcal{C}, st\} \leftarrow \mathcal{A}_1^{\mathcal{O}}(R, \mathcal{T}, para)$, where $\mathcal{C} = \{\mathcal{T}_{i_1}, \mathcal{T}_{i_2}, \cdots, \mathcal{T}_{i_\delta}\} \subseteq \mathcal{T}$ is a set of *clean* tags, $0 \leq \delta \leq \ell$;
3. $g \in_R \{1, \cdots, \delta\}$, set $\mathcal{T}_g = \mathcal{T}_{i_g}$ and $\widehat{\mathcal{T}} = \mathcal{T} - \mathcal{C}$;
4. $view_{\mathcal{A}} \leftarrow \mathcal{A}_2^{\mathcal{O}}(R, \widehat{\mathcal{T}}, \mathcal{I}(\mathcal{T}_g), st)$;
5. output $(g, view_{\mathcal{A}})$.

Fig. 2. zk-privacy experiment: real world

Experiment $\mathbf{Exp}_{\mathcal{S}}^{zkp}[\kappa, \ell]$
1. run $\mathsf{Setup}(\kappa, \ell)$ to setup the reader R and a set of tags $\mathcal{T}$; denote by $para$ the public system parameter;
2. $\{\mathcal{C}, st\} \leftarrow \mathcal{S}_1^{\mathcal{O}}(R, \mathcal{T}, para)$, where $\mathcal{C} = \{T_{i_1}, T_{i_2}, \cdots, T_{i_\delta}\} \subseteq \mathcal{T}$ is a set of *clean* tags, $0 \leq \delta \leq \ell$;
3. $g \in_R \{1, \cdots, \delta\}$, and set $\widehat{\mathcal{T}} = \mathcal{T} - \mathcal{C}$;
4. $sview \leftarrow \mathcal{S}_2^{\mathcal{O}}(R, \widehat{\mathcal{T}}, st)$, where $sview$ particularly includes all oracle answers to queries made by $\mathcal{S}$;
5. output $(g, sview)$.

Fig. 3. zk-privacy experiment: simulated world

Figure 3 illustrates the simulated world of zk-privacy experiment, $\mathbf{Exp}_{\mathcal{S}}^{zkp}[\kappa,\ell]$ ($\mathbf{Exp}_{\mathcal{S}}^{zkp}$, for simplicity), in which a PPT simulator $\mathcal{S}$ is comprised of a pair of algorithms $(\mathcal{S}_1, \mathcal{S}_2)$ and runs in two stages. In the *first stage*, algorithm $\mathcal{S}_1$ concurrently interacts with R and all the tags in $\mathcal{T}$ via the four oracles in $\mathcal{O}$, and outputs a set, denoted $\mathcal{C}$, of *clean* tags, where $|\mathcal{C}| = \delta$ and $0 \leq \delta \leq \ell$. It also outputs a state information st, which will be transmitted to algorithm $\mathcal{S}_2$. Between the two stages, a value g is taken uniformly at random from $\{1, \cdots, |\mathcal{C}|\}$ (which is unknown to $\mathcal{S}$). In the *second stage* of $\mathcal{S}$, on input st, $\mathcal{S}_2$ concurrently interacts with the reader R and the tags in $\widehat{\mathcal{T}} = \mathcal{T} - \mathcal{C}$, and outputs a simulated view, denoted $sview$, at the end of the second stage. We require that all oracle answers to the queries made by $\mathcal{S}$ (in both the first stage and the second

stage) in the experiment $\mathbf{Exp}_{\mathcal{S}}^{zkp}$ are included in $sview$. The output of the experiment $\mathbf{Exp}_{\mathcal{S}}^{zkp}$ is defined to be $(g, sview)$. Denote by $(g, sview(\kappa, \ell))$ the random variable describing the output of the experiment $\mathbf{Exp}_{\mathcal{S}}^{zkp}[\kappa,\ell]$.

Informally, an RFID protocol π satisfies zk-privacy, if what can be derived by interacting with the challenge tag $\mathcal{T}_g$ in the second-stage of $\mathcal{A}$ can actually be derived by $\mathcal{A}$ itself *without interacting with* $\mathcal{T}_g$. In this sense, the interaction between $\mathcal{A}_2$ and $\mathcal{T}_g$ leaks "zero knowledge" to $\mathcal{A}$. For this reason, our RFID privacy notion is named zk-privacy.

Definition 4.1 (zk-privacy). *An RFID protocol π satisfies computational (resp., statistical) zk-privacy, if for any PPT CMIM adversary $\mathcal{A}$ there exists a polynomial-time simulator $\mathcal{S}$ such that for all sufficiently large κ and any ℓ which is polynomials in κ (i.e., $\ell = poly(\kappa)$, where $poly(\cdot)$ is some positive polynomial), the following ensembles are computationally (resp., statistically) indistinguishable:*

- $\{g, view_{\mathcal{A}}(\kappa, \ell)\}_{\kappa\in N, \ell\in poly(\kappa)}$
- $\{g, sview(\kappa, \ell)\}_{\kappa\in N, \ell\in poly(\kappa)}$

That is, for any polynomial-time (resp., any computational power unlimited) algorithm D, it holds that $|Pr[D(\kappa, \ell, g, view_{\mathcal{A}}(\kappa, \ell)) = 1] - Pr[D(\kappa, \ell, g, sview(\kappa, \ell)) = 1]| = \varepsilon$, where ε is negligible in k. The probability is taken over the random coins used by $\mathsf{Setup}(\kappa, \ell)$, the random coins used by $\mathcal{A}$, $\mathcal{S}$, the reader R and all (uncorrupted) tags, the choice of g, and the coins used by the distinguisher algorithm D.

We now extend our definition to forward and backward zk-privacy. Denote by $(k_{\mathcal{T}_g}^f, s_{\mathcal{T}_g}^f)$ (resp., $(k_{\mathcal{T}_g}^1, s_{\mathcal{T}_g}^1)$) the final (resp., initial) secret-key and internal state of $\mathcal{T}_g$ at the end of (resp., beginning) of the experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$. An RFID protocol π is of *forward* (resp., *backward*) *zk-privacy*, if for any PPT CMIM adversary $\mathcal{A}$ there exists a polynomial-time simulator $\mathcal{S}$ such that for all sufficiently large κ and any $\ell = poly(\kappa)$, the following distributions are indistinguishable: $\{k_{\mathcal{T}_g}^f, s_{\mathcal{T}_g}^f(resp., k_{\mathcal{T}_g}^1, s_{\mathcal{T}_g}^1), g, view_{\mathcal{A}}(\kappa, \ell)\}$ and $\{k_{\mathcal{T}_g}^f, s_{\mathcal{T}_g}^f(resp., k_{\mathcal{T}_g}^1, s_{\mathcal{T}_g}^1), g, sview(\kappa, \ell)\}$. For forward/backward zk-privacy, it is required that the challenge tag $\mathcal{T}_g$ should remain *clean* at the end of experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$. Note that the adversary is allowed to corrupt the challenge tag after the end of $\mathbf{Exp}_{\mathcal{A}}^{zkp}$.

4.1 Discussions

Why allow $\mathcal{A}_1$ to output an arbitrary set *$\mathcal{C}$ of tags, and limit $\mathcal{A}_2$ to blind access to a challenge tag chosen randomly from $\mathcal{C}$?* The definition of zk-privacy implies that the adversary $\mathcal{A}$ cannot distinguish any challenge tag $\mathcal{T}_g$ from any set $\mathcal{C}$ of tags; otherwise, $\mathcal{A}$ can figure out the identity of $\mathcal{T}_g$ in $\mathcal{C}$ from its view $view_{\mathcal{A}}$, while this tag's identity cannot be derived from any simulator's view $sview$ (a formal proof of this in case of $|\mathcal{C}| = 2$ is provided in Section 5.1). If $\mathcal{C}$ is removed from the definition of zk-privacy, it is possible for the adversary to distinguish any two tags under its attack, even if each of the tags can be perfectly simulated by a simulator. A special case is that each tag has an upper-bound of sessions in its life time so that an adversary can distinguish any two tags by setting one tag to be run out of sessions in the learning stage [18]. In addition, we do not restrict $\mathcal{C}$ to two tags so as to take into account the case that any number of tags may be correlated.

Why limit $\mathcal{A}_1$ to output of clean *tags?* If $\mathcal{A}_1$ is allowed to output "unclean tags", $\mathcal{A}_2$ can trivially violate the zk-privacy. Consider that $\mathcal{A}_1$ selects two tags that are waiting for different round message (e.g., one tag is clean and the other is not), then $\mathcal{A}_2$ can trivially distinguish them by forwarding to $\mathcal{T}_g$ different round messages.

Why allow $\mathcal{S}$ to have access to oracles in $\mathcal{O}$? Suppose that $\mathcal{S}$ simulates a tag from scratch and $\mathcal{A}$ (run by $\mathcal{S}$ as a subroutine) requests to corrupt the tag in the middle of the simulation. Without oracle access, it is difficult or even impossible for $\mathcal{S}$ to continue its simulation and keep it consistent with its previous simulation for the same tag.

Why limit $sview$ to include all oracle answers to queries made by $\mathcal{S}$? This is to restrict $\mathcal{S}$ not to access the oracles in $\mathcal{O}$ more than $\mathcal{A}$ does. The indistinguishability between the simulated view $sview$ and the real view $view_{\mathcal{A}}$ of adversary $\mathcal{A}$ in zk-privacy implies that for any (t, n_1, n_2, n_3, n_4)-adversary $\mathcal{A}$, with overwhelming probability, $\mathcal{S}$ cannot query O_1, O_2, O_3, O_4 more than n_1, n_2, n_3, n_4 times, respectively.

Why require $\mathcal{T}_g$ to remain clean at the end of $\mathbf{Exp}_{\mathcal{A}}^{zkp}$ for forward/backward privacy? In general, forward/backward privacy cannot be achieved if the adversary is allowed to corrupt the challenge tag before the end of its sessions in $\mathbf{Exp}_{\mathcal{A}}^{zkp}$ (i.e., the tag is not clean at the moment of corruption); otherwise, the adversary is able to derive certain protocol messages from the tag's internal state, secret-key, random coins, and the partial session transcript.

More on backward privacy. In general, backward privacy means that even if $\mathcal{A}$ learns the internal state and secret-key of a tag for the v-th session, it still cannot distinguish the run of $(v+1)$-th session run by this tag from a simulated session run. Without loss of generality, we assume that the internal state and secret-key known to $\mathcal{A}$ are the initial ones (i.e., $k^1_{\mathcal{T}_g}$ and $s^1_{\mathcal{T}_g}$). For most RFID protocols in practice, the internal state and the secret-key of any tag at any time t can be determined by the tag's initial state, initial secret-key, and the session transcript related to the tag up to time t. In such a case, the indistinguishability between the simulated view $sview$ of $\mathcal{S}$ and the real view $view_{\mathcal{A}}$ of $\mathcal{A}$ relies upon the random coins used by $\mathcal{T}_g$ in experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$. These random coins are not disclosed to $\mathcal{A}$ since the random coins used by an uncorrupted tag in any session are erased once the session is completed, and the challenge tag $\mathcal{T}_g$ is required to be clean at the end of $\mathbf{Exp}_{\mathcal{A}}^{zkp}$.

On some special cases in zk-privacy experiments. One special case is that in the experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$, $\mathcal{A}_1$ outputs $\mathcal{C} = \mathcal{T}$. In this case, the simulator $\mathcal{S}_2$ does not have oracle access to any tag. The zk-privacy is analogue to auxiliary-input zero-knowledge [6], where the view of $\mathcal{A}_1/\mathcal{S}_1$ corresponds to the auxiliary input. Another special case is that $\mathcal{A}_1$ outputs only a single tag in $\mathcal{C}$, and all other tags can be corrupted by $\mathcal{A}_1$ and $\mathcal{A}_2$. In this case, the forward/backward zk-privacy implies that both adversary $\mathcal{A}$ and simulator $\mathcal{S}$ have access to certain secret information of all tags.

5 Comparison with Existing Frameworks

In this section, we compare our RFID security and privacy framework with typical existing frameworks. We argue that our framework is more reasonable in practice than some frameworks, and it is stronger in terms of privacy than at least one of the existing frameworks. We also clarify some subtleties and confusions in the existing frameworks.

The detailed comparisons, along with subtlety clarifications, also further justify the zk-privacy formulation.

5.1 Comparison with Model in [18]

The RFID privacy model proposed in [18] describes the indistinguishability between any two tags by an adversary. We refer to this privacy notion as "ind-privacy". It was mentioned in [18] that an important area for future research is to study stronger RFID privacy notions. We shall prove that zk-privacy is strictly stronger than a revised version of ind-privacy after some subtleties are clarified.

Roughly speaking, consider any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$: $\mathcal{A}_1$ outputs a pair of uncorrupted tags $(\mathcal{T}_{i_0}, \mathcal{T}_{i_1})$ after arbitrary attacks, then a bit g is chosen randomly and independently (which is unknown to $\mathcal{A}$), and then $\mathcal{A}_2$ is given blind access to $\mathcal{T}_{i_g}$ and finally outputs a guessed bit b'. We say a PPT adversary $\mathcal{A}$ $(\epsilon, t, n_1, n_2, n_3, n_4)$-breaks the ind-privacy of an RFID system if $\mathcal{A}$ is a (t, n_1, n_2, n_3, n_4)-adversary and $\Pr[b' = g] = \frac{1}{2} + \epsilon$, where ϵ is non-negligible and t is polynomial in κ.

On some subtleties in ind-privacy. In the original definition of ind-privacy, it is not explicitly specified that the two tags output by $\mathcal{A}_1$ must be clean tags. In the definition of forward ind-privacy [18], it is not precisely specified the time point of tag corruption and the actions of adversary after tag corruption.

zk-privacy vs. ind-privacy for single-tag systems. We note that any RFID protocol, *even if it just reveals the tag's secret-key*, trivially satisfies ind-privacy for special RFID systems consisting only one tag (e.g., for a unique item of high value). The reason is that in this special scenario, the view of $\mathcal{A}$ is independent of the random bit g (as the challenge tag $\mathcal{T}_{i_g}$ is always the unique tag regardless of the choice of g), and thus $\Pr[b' = g]$ is just $\frac{1}{2}$ for any adversary. In comparison, in this special scenario the zk-privacy is essentially degenerated to the traditional zero-knowledge definition, which still provides very reasonable privacy guarantee.

Theorem 1. *zk-privacy is stronger than ind-privacy.*

Proof. First, we show that zk-privacy implies ind-privacy, which holds unconditionally. In other words, if an RFID system $(R, \mathcal{T})$ does not satisfy ind-privacy, then it also does not satisfy zk-privacy. To prove this, we show that if there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which can $(\epsilon, t, n_1, n_2, n_3, n_4)$-break the ind-privacy of the RFID system $(R, \mathcal{T})$, then we can construct another PPT adversary $\mathcal{A}'$ such that no PPT simulator exists for $\mathcal{A}'$.

In the experiment $\mathsf{Exp}^{zkp}_{\mathcal{A}'}$, let $\mathcal{A}'$ run $\mathcal{A}$ and do whatever $\mathcal{A}$ does. In particular, $\mathcal{A}'$ and $\mathcal{A}$ are of the same parameters (t, n_1, n_2, n_3, n_4). Since $\mathcal{A}$ run by $\mathcal{A}'$ always outputs a *pair* of clean tags at the end of its first stage, $\mathsf{Exp}^{zkp}_{\mathcal{A}'}$ outputs $(g, view_{\mathcal{A}'})$, where $g \in \{0, 1\}$ is a random bit, and $view_{\mathcal{A}'}$ implicitly determines the output of $\mathcal{A}$ (i.e., the guessed bit b'). That is, the guessed bit b' can be computed out from $view_{\mathcal{A}'}$ in polynomial-time. As we assume $\mathcal{A}$ $(\epsilon, t, n_1, n_2, n_3, n_4)$-breaks ind-privacy, it holds that $\Pr[b' = g]$ is at least $\frac{1}{2} + \epsilon$ for the output of $\mathsf{Exp}^{zkp}_{\mathcal{A}'}$. However, the simulated view $sview$ in the output of the experiment $\mathsf{Exp}^{zkp}_{\mathcal{S}}$ is independent of g (recall that the random value g is unknown to the simulator $\mathcal{S}$). Therefore, for the guessed bit b' implied by

$sview$ (which can be computed out from $sview$ in polynomial-time), it always holds that $Pr[b' = g] = \frac{1}{2}$. This shows that for the above $\mathcal{A}'$ and for any polynomial-time simulator, there exists a polynomial-time distinguisher that can distinguish the output of $\mathsf{Exp}_{\mathcal{A}}^{zkp}$ and that of $\mathsf{Exp}_{\mathcal{S}}^{zkp}$ with non-negligible probability at least ϵ.

Next, we present several protocol examples (based on one-time secure signatures or CPA-secure public-key encryption) that satisfy ind-privacy but dissatisfy zk-privacy.

Consider a special RFID system that consists of only one tag $\mathcal{T}_1$ (and a reader R). The secret-key of $\mathcal{T}_1$ is the signature of $\mathcal{T}_1$'s ID, denoted s_{ID}, signed by R under the public-key of R. Consider an RFID protocol π in which $\mathcal{T}_1$ just reveals its secret-key s_{ID} to R. As discussed above, any RFID protocol trivially satisfies ind-privacy for RFID systems consisting of only one tag, and thus the protocol π is of ind-privacy. But, π clearly does not satisfy zk-privacy. Specifically, considering an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ simply outputs $\mathcal{C} = \{\mathcal{T}_1\}$ and then $\mathcal{A}_2$ invokes $\mathcal{T}_g = \mathcal{T}_1$ to get the signature s_{ID}, no PPT simulator can output s_{ID} by the security of the underlying signature scheme. Note that one-time secure signature is sufficient to show this protocol example not satisfying zk-privacy, and one-time secure signatures can be based on any one-way function [24].

Given any ind-private two-round RFID protocol $\pi = (c, a)$ for an RFID system $(R, \mathcal{T})$, where $\mathcal{T}$ consists of polynomially many tags, c is the first-round message from the reader and a is the response from a tag, we transform π into a new protocol π' as follows: In the protocol π', besides their respective secret-keys all tags in $\mathcal{T}$ also share a unique pair of public-key PK and secret-key SK for a CPA-secure public-key encryption scheme. For a protocol run of π' between the reader R and a tag $\mathcal{T}_i$, R sends $c' = E_{PK}(c)$ in the first-round, and $\mathcal{T}_i$ decrypts c' to get c and then sends back $a' = c||a$. The protocol π' could appear in the scenario of tag group authentication, where the ability of sending back c can demonstrate the membership of the group identified by the public-key PK. Furthermore, in the scenario of anonymizer-enabled RFID systems [9], the decryption operation can be performed by the anonymizer. As in the new protocol π' all tags share the same public-key PK, the ind-privacy of π' is inherited from that of π. Specifically, the session transcripts of π' can be computed in polynomial-time from the session transcripts of π and the public-key PK. However, π' does not satisfy zk-privacy. Specifically, consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ simply outputs the set of clean tags $\mathcal{C} = \mathcal{T}$ (in particular, $\mathcal{A}$ never corrupts tags) and then $\mathcal{A}_2$ blindly interacts with the challenge tag $\mathcal{T}_g$ for only one session. By the CPA-security of the underlying public-key encryption scheme, no PPT simulator can handle the $\mathsf{SendT}(challenge, \hat{c})$ queries made by $\mathcal{A}_2$, as such ability implies the ability of ciphertext decryption. Note that CPA security is sufficient here, as the adversary $\mathcal{A}$ involves only one session with the challenge tag $\mathcal{T}_g$. □

We remark that though the above two protocol examples may not be very realistic, they do separate the zk-privacy notion and the ind-privacy notion. We leave it an interesting question to find more protocol examples that are ind-private but not zk-private.

5.2 Comparison with Model in [26,23]

In [26,23], the simulator is not required to handle tag corruption queries by the adversary. In other words, the simulator works only for those adversaries which do not make tag corruption queries. It is not clear how such a simulator acts upon tag corruption

queries made by an adversary. Suppose that $\mathcal{S}$ simulates a tag from scratch and $\mathcal{A}$ (typically run by $\mathcal{S}$ as a subroutine) requests to corrupt the tag in the middle of simulation (possibly in the middle of a session run). Without access to tag corruption queries, it is difficult or even impossible for $\mathcal{S}$ to continue its simulation for the tag and keep it consistent with its previous simulation for the same tag.

The adversary considered in our framework essentially corresponds to strong adversary in [26,23], with the difference in that the adversary cannot corrupt any tag in set C before the end of zk-privacy experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$. In comparison, the model in [26,23] poses no restriction on tag corruption (though it is not clear how the simulator handles such adversaries), which implies that an adversary can corrupt any tag at any time (possibly in the middle of session). However, in such a case, forward/backward privacy may not be achievable if the challenge tag is corrupted in the middle of a session; this is the reason why we require that the challenge tag $\mathcal{T}_g$ must remain *clean* at the moment of corruption. Indeed, there are some confusions in [26,23].

The matching session concept defined in [26,23] is restricted to identical session transcript, without clarifying some subtleties such as the "last-round-message attacks" for defining authentication from tag to reader.

The notion of adaptive completeness is not defined in [26,23]. The completeness notion in [26,23] is defined for honest protocol execution only, with no adversarial desynchronizing attacks being taken into account.

The privacy notions proposed in [26,23] and that proposed in [18] are essentially incomparable, while the privacy notion proposed in this work is strictly stronger than that of [18].

5.3 Comparison with Models in [10,20]

The RFID privacy notion given in [10,20] is formulated based on the unpredictability of protocol output. We refer to this privacy notion as "unp-privacy." The unp-privacy is formulated with respect to RFID protocols with a 3-round canonical form, denoted as $\pi = (c, r, f)$, where c, r, f stand for the first, second, and third round message, respectively. Note that our framework, as well as models in [18,26,23]), are not confined to this protocol structure.

The unp-privacy notion formulated in [10,20] essentially says that the second-round message sent from a tag must be pseudorandom (i.e., indistinguishable from a truly random string). We observe that this requirement has certain limitations. First, given any unp-private RFID protocol $\pi = (c, r, f)$ between a reader and a tag, we can modify the protocol to $\pi' = (c, r||1, f)$, where "||" denotes the string concatenation operation. That is, the modified protocol π' is identical to π except that in the second-round the tag additionally concatenates a bit '1' to r. This modified RFID-protocol π' is not of unp-privacy, as the second-round message $r||1$ is clearly not pseudorandom. However, intuitively, the tags' privacy should be preserved since the same bit '1' is appended to all second-round messages for all tags. Notice that when RFID-protocols are implemented in practice, the messages being exchanged between reader and tags normally bear some non-random information such as version number of RFID standard. Another limitation is that the unp-privacy may exclude the use of public-key encryption in RFID-protocols, as public-key generated ciphertexts are typically *not* pseudorandom.

Another point is that the adversaries considered in the definition of unp-privacy [10,20] is not allowed to access protocol outputs. Therefore, such adversaries are *narrow* ones as defined in [26,23]. Informally, the unp-privacy experiment works as follows. Given a first-round message c (which could be generated by the adversary $\mathcal{A}$), the experiment selects a value r which could be either the actual second-round message generated by an uncorrupted tag in response to c or just a random value in a certain domain; then the experiment presents the value r to $\mathcal{A}$. The unp-privacy means that $\mathcal{A}$ cannot determine in which case the value r is. Note that if $\mathcal{A}$ has access to protocol outputs, it can simply distinguish between the two cases of r. What $\mathcal{A}$ needs to do is to forward r to the reader R as the second round message. If r is generated by an uncorrupted tag (and the value c was generated by the reader in a matching session), R will always output "accept." On the other hand, if r is just a random value, with overwhelming probability R will reject the message due to authentication soundness from tag to reader.

In summary, we argue that zk-privacy is more reasonable than unp-privacy in practice. It allows for more general protocol structure, more powerful adversary, and non-pseudorandom protocol messages.

6 An RFID Protocol within Our Framework

Let F_k: $\{0,1\}^{2\kappa} \rightarrow \{0,1\}^{2\kappa}$ be a pre-specified keyed PRF and F_k^0 (resp., F_k^1) the κ-bit prefix (resp., suffix) of the output of F_k, where κ is the system security parameter. In practice, the PRF can be implemented based on some lightweight stream or block ciphers [12,2,11]. When a tag $\mathcal{T}_i$ with identity ID registers to the reader R, it is assigned a secret-key $k \in_R \{0,1\}^\kappa$, a counter ctr of length l_{ctr} with initial value 1. R precomputes an initial index $I = F_k^0(1||pad_1)$ for the tag, where $pad_1 \in \{0,1\}^{2\kappa - l_{ctr}}$ is a fixed padding, and stores the tuple (I, k, ctr, ID) into its database.

At the start of a new protocol session, R sends a challenge string $c \in_R \{0,1\}^\kappa$ to $\mathcal{T}_i$, which also serves as the session identifier. To simplify the presentation, the session identifier and the corresponding verification of the identifier by protocol players are implicitly implied and will not be explicitly mentioned in the following.

Upon receiving c from R, $\mathcal{T}_i$ computes $I = F_k^0(ctr||pad_1)$, $(r_0, r_1) = F_k(c||I)$ (where $r_0 = F_k^0(c||I)$ and $r_1 = F_k^0(c||I)$), and $r_{\mathcal{T}} = r_0 \oplus (ctr||pad_2)$. $\mathcal{T}_i$ sends $(I, r_{\mathcal{T}})$ to R and then updates its counter $ctr = ctr + 1$, where $pad_2 \in \{0,1\}^{\kappa - l_{ctr}}$ is another predetermined padding string.

After receiving $(I, r_{\mathcal{T}})$, R searches its database to find a tuple indexed by I:

- If R finds such a tuple, say (I, k, ctr', ID), it computes $(r_0, r_1) = F_k(c||I)$, and checks whether $ctr'||pad_2 = r_0 \oplus r_{\mathcal{T}}$: If yes, R accepts $\mathcal{T}_i$ by outputting "1", sends $r_R = r_1$ to the tag, updates the tuple (I, k, ctr', ID) with $ctr' = ctr' + 1$ and $I = F_k^0(ctr'||pad_1)$; If not, R searches for the next tuple including I (to avoid potential collision of index I, i.e., two different tuples are of the same index I).
- If no tuple is found to have an index I (which indicates counter desynchronization between R and $\mathcal{T}_i$), for each tuple (I', k, ctr', ID) in its database, R computes $(r_0, r_1) = F_k(c||I)$ and $ctr||pad_2 = r_0 \oplus r_{\mathcal{T}}$, and checks whether $I = F_k^0(ctr||pad_1)$: If yes (which indicates ctr is the correct counter value at $\mathcal{T}_i$), R

accepts $\mathcal{T}_i$, outputs "1", sends back $r_R = r_1$ as the third message, and updates the tuple (I', k, ctr', ID) with $ctr' = ctr + 1$ and $I' = F_k^0(ctr'||pad_1)$. In the case that R fails with all the tuples in its database, it rejects the tag and outputs "0".

Upon receiving r_R, $\mathcal{T}_i$ checks whether $r_R = r_1$: If yes, $\mathcal{T}_i$ accepts the reader and outputs "1"; otherwise it rejects the reader and outputs "0".

In comparison with the protocol proposed in [20], the above protocol adds mutual authentication (and is logically more precise), and we can formally prove that it is of adaptive completeness, mutual authentication, and zk-privacy within the new framework. Analysis of completeness and authentication was not conducted in [20], and as we shall see, the zk-privacy analysis of the new protocol is much more complicated than the unp-privacy analysis in [20]. We suggest that the methodology used in our analysis is of independent interest, which can be applied to analyze other RFID protocols (particularly those based on PRFs) within our new framework.

Theorem 2. *Assuming F_k is a pseudorandom function, the protocol specified above satisfies adaptive completeness, mutual authentication and zk-privacy.*

The reader is referred to the full paper [4] for the complete proof of this theorem. Below we provide a high level analysis of the zk-privacy property.

The core of the simulation by the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, who runs the underlying adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as a subroutine, lies in the actions of $\mathcal{S}_2$ in dealing with the following queries made by $\mathcal{A}_2$ to the reader R and the challenge tag $\mathcal{T}_g$. $\mathcal{S}_1$ just mimics $\mathcal{A}_1$ by using the PRF F_k.

1. On oracle query $\mathsf{InitReader}()$, $\mathcal{S}_2$ makes the same oracle query to R, and gets back a random string $c \in \{0,1\}^\kappa$ from R. Then, $\mathcal{S}_2$ relays back c to $\mathcal{A}_2$.
2. On oracle query $\mathsf{SendT}(challenge, \hat{c})$, where the challenge tag $\mathcal{T}_g$ (simulated by $\mathcal{S}_2$) currently does not run any session, $\mathcal{S}_2$ opens a session for $\mathcal{T}_g$ with $\hat{c}$ as the first-round message (that also serves as the session-identifier of this new session); Then, $\mathcal{S}_2$ randomly selects $I, r_{\mathcal{T}} \in_R \{0,1\}^\kappa$, and sends back $I||r_{\mathcal{T}}$ to $\mathcal{A}_2$ as the second-round message.
3. On oracle query $\mathsf{SendR}(\hat{c}, \hat{I}||\hat{r}_{\mathcal{T}})$, $\mathcal{S}_2$ works as follows:
   Case-3.1. If $\hat{I}||\hat{r}_{\mathcal{T}}$ was sent by $\mathcal{T}_g$ (simulated by $\mathcal{S}_2$) in a session of session-identifier $\hat{c}$, $\mathcal{S}_2$ simulates the responses of the reader R as follows:
   Case-3.1.1 If R is running an incomplete session of session-identifier $\hat{c}$ (i.e., $\hat{c}$ was sent by R upon an $\mathsf{InitReader}$ query and R is waiting for the second-round message), $\mathcal{S}_2$ just returns a random string $r_R \in_R \{0,1\}^\kappa$ to $\mathcal{A}_2$, and outputs "1" indicating "accept".
   Case-3.1.2. Otherwise, $\mathcal{S}_2$ simply returns a special symbol "$\perp$" indicating invalid query.
   Case-3.2. In all other cases, $\mathcal{S}_2$ makes the same oracle query $\mathsf{SendR}(\hat{c}, \hat{I}||\hat{r}_{\mathcal{T}})$ to the reader R, and relays back the answer from R to $\mathcal{A}_2$.
4. On oracle query $\mathsf{SendT}(challenge, \hat{r}_R)$, where the challenge tag $\mathcal{T}_g$ (simulated by $\mathcal{S}_2$) currently runs a session of partial session-transcript $(\hat{c}, I||r_{\mathcal{T}})$ and is waiting for the third-round message, $\mathcal{S}_2$ works as follows:
   Case-4.1. If there exists a matching session of the same session transcript $(\hat{c}, I||r_{\mathcal{T}}, \hat{r}_R)$ at the side of R (where $\hat{r}_R$ may be simulated by $\mathcal{S}_2$ as in the above Case-3.1), $\mathcal{S}_2$ outputs "1" indicating "accept".

Case-4.2. Otherwise, $\mathcal{S}_2$ simply outputs "0" indicating "reject".

5. Output of $\mathcal{S}_2$: Finally, whenever $\mathcal{A}_2$ stops, $\mathcal{S}_2$ also stops and outputs the simulated view $sview$ as specified in the zk-privacy definition, which particularly consists of all oracle answers (including ones provided by the real oracles in $\mathcal{O}$ and *ones simulated by* $\mathcal{S}_2$) to queries made by $\mathcal{A}$.

It is easy to see that $\mathcal{S}$ works in polynomial-time. We investigate the differences between the simulated view $sview$ output by $\mathcal{S}$ and the real view $view_{\mathcal{A}}$ of $\mathcal{A}$:

Difference-1: In Case-4.1 (resp., Case-4.2) $\mathcal{S}_2$ always outputs "accept" (resp., "reject"), while the actual challenge tag $\mathcal{T}_g$ may output "reject" in Case-4.1 (resp., "accept" in Case-4.2) in the experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$.

Difference-2: On oracle query $\mathsf{SendT}(challenge, \hat{c})$ or in Case-3.1 upon the oracle query $\mathsf{SendR}(\hat{c}, \hat{I}||\hat{r}_{\mathcal{T}})$, $\mathcal{S}_2$ always returns truly random strings, while the actual players (i.e., $\mathcal{T}_g$ and R) provide pseudorandom strings in the experiment $\mathbf{Exp}_{\mathcal{A}}^{zkp}$ by invoking the PRF F_k where k is the secret-key of $\mathcal{T}_g$.

Intuitively, Difference-1 can occur only with negligible probability, by the properties of adaptive completeness and mutual authentication. The subsequent analysis argues that the properties of adaptive completeness and mutual authentication indeed hold under the simulation of $\mathcal{S}$ in $\mathbf{Exp}_{\mathcal{S}}^{zkp}$.

Intuitively, Difference-2 should not constitute distinguishable gap between $sview$ and $view_{\mathcal{A}}$, due to the pseudorandomness of F_k. However, the technical difficulty and subtlety here is that: the difference between pseudorandomness and real randomness only occurs in the second stages of both $\mathbf{Exp}_{\mathcal{A}}^{zkp}$ and $\mathbf{Exp}_{\mathcal{S}}^{zkp}$ (i.e., $\mathcal{A}_2$ and $\mathcal{S}_2$), while both $\mathcal{S}_1$ and $\mathcal{A}_1$ are w.r.t. the PRF F_k. In other words, to distinguish the PRF F_k from a truly random one in the second stage, the distinguisher has already accessed F_k for polynomially many times in the first stage. In general, the definition of PRF says nothing on the pseudorandomness in the second stage. To overcome this technical difficulty, we build a list of hybrid experiments.

In the first hybrid experiment, a polynomial-time algorithm $\hat{S}$ runs $\mathcal{A}$ as a subroutine and has oracle access to the PRF F_k or a truly random function H. $\hat{S}$ first randomly guesses the challenge tag $\mathcal{T}_g$ (by taking g uniformly at random from $\{1, \cdots, \ell\}$), and then setups the RFID system $(R, \mathcal{T})$ except for the challenge-tag $\mathcal{T}_g$. Note that $\hat{S}$ can perfectly handle all oracle queries made by $\mathcal{A}$ to the reader R and all tags in $\mathcal{T} - \{\mathcal{T}_g\}$. For oracle queries directed to $\mathcal{T}_g$, $\hat{S}$ mimics $\mathcal{T}_g$ with the aid of its oracle, i.e, the PRF F_k or a truly random function H. Denote by the view of $\mathcal{A}$ under the run of $\hat{S}$ with oracle access to F_k (resp., H) as $view_{\mathcal{A}}^{\hat{S}^{F_k}}$ (resp., $view_{\mathcal{A}}^{\hat{S}^{H}}$). By the pseudorandomness of F_k, we have that $view_{\mathcal{A}}^{\hat{S}^{F_k}}$ and $view_{\mathcal{A}}^{\hat{S}^{H}}$ are indistinguishable. Next, suppose $\hat{\mathcal{S}}$ successfully guesses the challenge tag $\mathcal{T}_g$ (that occurs with probability $\frac{1}{\ell}$), $view_{\mathcal{A}}^{\hat{S}^{F_k}}$ is identical to $view_{\mathcal{A}}$. In particular, in this case, the properties of adaptive completeness and mutual authentication hold in $view_{\mathcal{A}}^{\hat{S}^{F_k}}$ and thus also in $view_{\mathcal{A}}^{\hat{S}^{H}}$ (as $view_{\mathcal{A}}^{\hat{S}^{F_k}}$ and $view_{\mathcal{A}}^{\hat{S}^{H}}$ are indistinguishable). Thus, to show the indistinguishability between $view_{\mathcal{A}}$ and $sview$, it is reduced to show the indistinguishability between $view_{\mathcal{A}}^{\hat{S}^{H}}$ (in case $\hat{S}$ successfully guesses the challenge tag $\mathcal{T}_g$) and $sview$.

In the second hybrid experiment, we consider another polynomial-time algorithm S' that mimics $\hat{S}$, with oracle access to F_k or H, but with the following modifications:

in the second stage of this hybrid experiment, S' essentially mimics the original zk-privacy simulator $\mathcal{S}$. Denote by the view of $\mathcal{A}$ under the run of S' with oracle access to F_k (resp., H) as $view_{\mathcal{A}}^{S'^{F_k}}$ (resp., $view_{\mathcal{A}}^{S'^{H}}$). By the pseudorandomness of F_k, $view_{\mathcal{A}}^{S'^{F_k}}$ and $view_{\mathcal{A}}^{S'^{H}}$ are indistinguishable. We can show that $view_{\mathcal{A}}^{S'^{H}}$ and $view_{\mathcal{A}}^{\hat{S}^{H}}$ are also indistinguishable, and that $view_{\mathcal{A}}^{S'^{F_k}}$ and $sview$ are also indistinguishable (conditioned on S' successfully guesses the challenge tag $\mathcal{T}_g$), which particularly implies that the properties of adaptive completeness and mutual authentication hold also in $sview$. This establishes the indistinguishability between $sview$ and $view_{\mathcal{A}}$.

7 Future Work

One of our future research directions is to analyze existing RFID protocols and design new protocols within the new framework presented in this paper.

Since our framework is formulated w.r.t. the basic scenario of an RFID system, another future research direction is to extend our RFID privacy framework to more sophisticated and practical scenarios which allow compromising of readers, tag cloning (or more feasibly, protocols to prevent swapping attacks) [16,17], tag group authentication, anonymizer-enabled RFID systems, and tag ownership transfer.

Acknowledgment. We are indebted to Andrew C. Yao for many contributions to this work, though he finally declined the coauthorship. The contact author thanks Shaoying Cai for helpful discussions on RFID security and privacy. We thank the anonymous referee for referring us to [16,17].

References

1. Berbain, C., Billet, O., Etrog, J., Gilbert, H.: An Efficient Forward Private RFID Protocol. In: Conference on Computer and Communications Security – CCS 2009 (2009)
2. de Canniere, C., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008)
3. Damgård, I., Ostergaard, M.: RFID Security: Tradeoffs between Security and Efficiency. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 318–332. Springer, Heidelberg (2008)
4. Deng, R.H., Li, Y., Yao, A.C., Yung, M., Zhao, Y.: A New Framework for RFID Privacy. Cryptology ePrint Archive, Report No. 2010/059
5. Garfinkel, S., Juels, A., Pappu, R.: RFID Privacy: An Overview of Problems and Proposed Solutions. IEEE Security and Privacy 3(3), 34–43 (2005)
6. Goldreich, O.: The Foundations of Cryptography. Basic Tools, vol. I. Cambridge University Press, Cambridge (2001)
7. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM 33(4), 792–807 (1986)
8. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof-Systems. In: ACM Symposium on Theory of Computing, pp. 291–304 (1985)
9. Golle, P., Jakobsson, M., Juels, A., Syverson, P.: Universal reencryption for mixnets. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 163–178. Springer, Heidelberg (2004)

10. Ha, J., Moon, S., Zhou, J., Ha, J.: A new formal proof model for RFID location privacy. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 267–281. Springer, Heidelberg (2008)
11. Hell, M., Johansson, T., Meier, W.: The Grain Family of Stream Ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
12. International Standard ISO/IEC 9798 Information technology−Security techniques–Entity authentication–Part 5: Mechanisms using Zero-Knowledge Techniques
13. Hopper, N.J., Blum, M.: Secure human identification protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 52–66. Springer, Heidelberg (2001)
14. Juels, A.: RFID Security and Privacy: A Research Survey. IEEE Journal on Selected Areas in Communications 24(2), 381–394 (2006)
15. Juels, A., Rivest, R.L., Szydlo, M.: The blocker tag: Selective blocking of RFID tags for consumer privacy. In: ACM CCS 2003, pp. 103–111 (2003)
16. Juels, A., Pappu, R.: Squealing Euros: Privacy Protection in RFID-Enabled Banknotes. Financial Cryptography, 103–121 (2003)
17. Juels, A., Syverson, P., Bailey, D.: High-Power Proxies for Enhancing RFID Privacy and Utility. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 210–226. Springer, Heidelberg (2006)
18. Juels, A., Weis, S.: Defining Strong Privacy for RFID. In: International Conference on Pervasive Computing and Communications – PerCom 2007 (2007)
19. Juels, A., Weis, S.: Authenticating pervasive devices with human protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)
20. Ma, C., Li, Y., Deng, R., Li, T.: RFID Privacy: Relation Between Two Notions, Minimal Condition, and Efficient Construction. In: ACM CCS (2009)
21. Yu Ng, C., Susilo, W., Mu, Y., Safavi-Naini, R.: RFID privacy models revisited. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 251–266. Springer, Heidelberg (2008)
22. Yu Ng, C., 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)
23. Paise, R.L., Vaudenay, S.: Muthal Authentication in RFID: Security and Privacy. In: AsiaCCS 2008, pp. 292–299 (2008)
24. Rompel, J.: One-Way Functions are Necessary and Sufficient for Digital Signatures. In: 22nd ACM Symposium on Theory of Computing (STOC 1990), pp. 12–19 (1990)
25. Shamir, A.: SQUASH: A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 144–157. Springer, Heidelberg (2008)
26. Vaudenay, S.: On Privacy Models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)
27. 860 MHz - 930 MHz Class 1 RFID Tag Radio Frequency and Logical Communication Interface Specification Candidate Recommendation Version 1.0.1, Auto-ID Center (2002)

Readers Behaving Badly
Reader Revocation in PKI-Based RFID Systems

Rishab Nithyanand, Gene Tsudik, and Ersin Uzun

Computer Science Department
University of California
Irvine, CA 92697
{rishabn,gts,euzun}@ics.uci.edu

Abstract. Recent emergence of RFID tags capable of performing public key operations motivates new RFID applications, including electronic travel documents, identification cards and payment instruments. In this context, public key certificates form the cornerstone of the overall system security. In this paper, we argue that one of the prominent challenges is how to handle revocation and expiration checking of RFID reader certificates. This is an important issue considering that these high-end RFID tags are geared for applications such as e-documents and contactless payment instruments. Furthermore, the problem is unique to public key-based RFID systems, since a passive RFID tag has no clock and thus cannot use (time-based) off-line methods.

In this paper, we address the problem of reader certificate expiration and revocation in PKI-Based RFID systems. We begin by observing an important distinguishing feature of *personal* RFID tags used in authentication, access control or payment applications – the involvement of a human user. We take advantage of the user's awareness and presence to construct a simple, efficient, secure and (most importantly) feasible solution. We evaluate the usability and practical security of our solution via user studies and discuss its feasibility.

1 Introduction

Radio Frequency Identification (RFID) is a wireless technology mainly used for identification of various types of objects, e.g, merchandise. An RFID tag is a passive device, i.e., it has no power source of its own. Information stored on an RFID tag can be read by special devices called RFID readers, from some distance away and without requiring line-of-sight alignment. Although RFID technology was initially envisaged as a replacement for barcodes in supply chain and inventory management, its many advantages have greatly broadened the scope of possible applications. Current and emerging applications range from visible and personal (e.g., toll transponders, passports, credit cards, access badges, livestock/pet tracking devices) to stealthy tags in merchandise (e.g., clothes, pharmaceuticals and library books). The cost and capabilities of an RFID tag vary widely depending on the target application. At the high end of the spectrum are

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 19–36, 2010.

the tags used in e-Passports, electronic ID (e-ID) Cards, e-Licenses, and contactless payment instruments. Such applications involve relatively sophisticated tags each costing a few (usually < 10) dollars. These tags are powerful enough to perform public key cryptographic operations.

In the "real world", one of the main security issues in using public key cryptography is certificate revocation. Any certificate-based public key infrastructure (PKI) needs an effective revocation mechanism. Revocation can be handled implicitly, via certificate expiration, or explicitly, via revocation status checking. Most PKI-s use a combination of implicit and explicit methods. The latter can be done off-line, using Certificate Revocation Lists (CRLs) [12] and similar structures, or on-line, using protocols such as Open Certificate Status Protocol (OCSP) [27]. However, as discussed below, these approaches are untenable in public key-enabled RFID systems.

Intuitively, certificate revocation in RFID systems should concern two entities: RFID tags and RFID readers. The former only becomes relevant if each tag has a "public key identity". We claim that revocation of RFID tags is a non-issue, since, once a tag identifies itself to a reader, the latter (as the entity performing a revocation check) can use any current revocation method, except perhaps OCSP which requires full-time Internet connectivity. This is reasonable because an RFID reader is a full-blown computing device with its own clock as well as ample power, memory, secondary storage and communication interfaces. Consequently, it can avail itself of any suitable revocation checking technique.

In contrast, revocation of readers is a problem in any public key-enabled RFID system. While a tag may or may not have public key identity, a reader must have one; otherwise, the use of public key cryptography becomes non-sensical. Therefore, before a tag discloses any information to a reader, it must make sure that the reader's public key certificate (PKC) is not expired or revoked.

1.1 Why Bother?

One common and central purpose of all RFID tags and systems is to enable tag identification (at various levels of granularity) by readers. With that in mind, many protocols have been proposed to protect the identification process (i.e., the tag-reader dialog) from a range of attacks. In systems where tags can not perform cryptographic operations or where they are limited to symmetric cryptography, reader revocation is not an issue, since it is essentially impossible. Whereas, in the context of public key-enabled tags, reader revocation is both imperative and possible, as we show later in this paper. It is imperative, because not doing it prompts some serious threats. For example, consider the following events: a reader is *lost*, *stolen*, *compromised* (perhaps without its owner's knowledge), or *decommissioned*.

In all of these cases, if it cannot be revoked effectively, a reader that has fallen into the wrong hands can be used to identify and track tags. In case of personal tags – e.g., ePassports, credit-cards or eIDs – other threats are possible, such as identity theft or credit card fraud.

Thus far, it might seem that our motivation is based solely on the need to detect *explicitly revoked* reader certificates[1]. However, what if a reader certificate naturally expires? This indicates *implicit revocation* and a well-behaved reader would not be operated further until a new certificate is obtained. However, if a reader (or rather its owner) is not well-behaved, it might continue operation with an expired certificate. Without checking certificate expiration, an unsuspecting tag could be tricked into identifying itself and possibly divulging other sensitive information.

In the remainder of this paper, we make no distinction between explicit revocation (i.e., revocation before expiration) and implicit revocation (i.e., certificate expiration) checking. The reason is that both tasks are essential for security and both require current time.

1.2 Why Is Reader Revocation Hard?

When presented with a PKC of a reader, a tag needs to check three things: (1) *signature* of the issuing certification authority (CA), (2) *expiration* and (3) *revocation status*.

The first is easy for any public key-enabled (pk-enabled) tag and has been already incorporated into some reader authentication schemes [6], [14]. However, (2) and (3) are problematic. Note that even a high-end tag is a passive device lacking a clock. Thus, a tag, by itself, has no means of deciding whether a presented certificate is expired.

Revocation checking is even more challenging. First, similar to expiration, offline revocation checking (e.g., CRL-based) requires current time because the tag needs to check the timeliness of the presented proof of non-revocation. Also, communicating a proof of non-revocation entails extra bandwidth from the reader to the tag. For CRLs, the bandwidth is $O(n)$ and, for more efficient CRTs, the bandwidth is $O(log\ n)$ – a non-negligible number for large values of n, where n is the number of revoked readers[2]. Whereas, online revocation checking protocols (such as OSCP) offer constant-size proofs of non-revocation. However, such protocols are unsuitable due to their connectivity and availability requirements ;see Section 3 for further discussion.

1.3 Roadmap

We focus on a class of pk-enabled RFID systems where tags are both personal and attended. This includes e-Passports, e-Licenses and contactless credit cards. *Personal* means that a tag belongs to a human user and *attended* means that a tag is supposed to be activated only with that user's (owner's) consent. Our approach is based on several observations:

[1] "Explicitly" means before the expiration of the PKC.

[2] The problem of the high communication cost of CRL-s in current solutions has been noted by Blundo, et al. [4].

- User/owner presence and (implicit) consent are already required for the tag to be activated.
- Low-cost and low-power flexible display technology is a reality, e.g., e-paper and OLED. In fact, passive RFID tags with small (6-10 digit) displays have been demonstrated and are currently feasible.
- Since certificate revocation and expiration granularity is usually relatively coarse-grained (i.e., days or weeks, but not seconds or minutes), users can distinguish between timely and stale date/time values.

The rest is straight-forward: a display-equipped tag receives, from a reader, a PKC along with a signed and time-stamped proof of non-revocation. After verifying the respective signatures on the reader's PKC and the non-revocation proof, the tag displays the lesser of: (1) PKC expiration time and (2) non-revocation proof expiration time. The user, who is assumed to be reasonably aware of current time, validates the timeliness of the displayed time. If it is deemed to be stale, the user aborts the interaction with the reader. Otherwise, user allows the interaction to proceed.

Organization: We summarize related work in Section 2 and overview some trivial solutions in Section 3. We describe our approach in Section 4, followed by results of the usability study in Section 5. The paper ends with the summary in Section 6.

2 Related Work

There are many ways of handling certificate revocation. Of these, Certificate Revocation Lists (CRLs) are the most commonly used mechanism. Notably, CRLs are used by the X.509 Public Key Infrastructure for the Internet [12]. Some techniques improve the efficiency of revocation checking. Certificate Revocation Trees (CRTs) [19] use Merkle's Hash Trees [23] to communicate a relatively short non-revocation proofs (of size log n). Skip-lists [9] and 2-3 Trees [28] improve on the CRT update procedure through the use of dynamic data structures, offering asymptotically shorter proofs. Online Certificate Status Protocol (OCSP) [27] is an on-line method that reduces storage requirements and provides timely revocation status information. Certificate Revocation System (CRS) [25,24] offers fully implicit certificate revocation by placing the bulk of revocation burden on the prover (certificate owner) and yields compact proofs of certificate validity.

In spite of substantial prior work in both certificate revocation and RFID security, very little has been done with respect to reader revocation and expiration checking. However, the problem has been recognized in previous literature [26,11,15,10,7,30].

3 Trivial Solutions

We now consider some trivial reader revocation techniques and discuss their shortcomings.

3.1 Date Register and Time Stamps

Every PKC has a validity period defined by its effective date (D_{eff}) and expiration date (D_{exp}). During certificate verification, a tag can use the date stored in its register (D_{curr}) to determine whether a certificate has expired. Verification steps are as follows:

1. Tag verifies the CA signature of the reader's certificate.
2. Tag checks that D_{exp} is greater than D_{curr}.
3. If (1) and (2) succeed, the tag accepts the certificate. If D_{eff} is greater than D_{curr}, the tag updates D_{curr} to D_{eff}.

With this approach, the estimate of the current date – D_{curr} – stored by the tag is not guaranteed to be accurate and thus can not always protect it from readers with expired or revoked certificates. This is especially the case for a tag that has not been used for some time. The value of D_{curr} might reflect a date far in the past, exposing the tag to attacks from readers revoked at any point after D_{curr}.

3.2 On-Line Revocation Checking

Online revocation-checking approaches, such as OCSP [27], alleviate client storage requirements by introducing trusted third parties (responders) that provide on-demand and up-to-date certificate status information. To validate a certificate, a client sends an OCSP status request to the appropriate responder and receives a signed status. In its basic form, OCSP requires a clock on the client, as it uses time-stamps to assure freshness. However, an optional OCSP extension supports the use of nonces as an alternative.

Although suitable for a large and well-connected infrastructure, such as a private network or the Internet, OCSP is problematic in RFID systems. Its use would require a tag to generate random challenges and conduct a 2-round (online) challenge-response protocol with an OCSP responder. Random challenges must be generated using a Pseudo-Random Number Generator (PRNG), which requires extra resources on the tag. More importantly, OCSP would necessitate constant infrastructure connectivity for all readers and availability of OCSP responders. Furthermore, the turnaround time for tag-reader interaction would become dependent on external factors, such as congestion of the communication infrastructure (e.g., the Internet) and current load on OCSP responders. Either factor might occasionally cause significant delays and prompt the need for back-up actions.

3.3 Internal Clocks

An internal clock would allow tags to accurately determine whether a certificate is expired and whether a non-revocation proof is current. However, a typical RFID tag is a purely passive device powered by radio waves emitted from a nearby reader. Since a real-time clock needs uninterrupted power, it cannot be sustained by passive tags. One might consider equipping RFID tags with batteries, however, this raises a slew of new problems, such as battery cost, clock synchronization and battery replacement.

4 Proposed Technique

We re-emphasize that our approach is aimed only at pk-based RFID systems. It has one simple goal: secure and reliable revocation checking on RFID tags. In the rest of this section, we discuss our assumptions and details of the proposed solution.

4.1 Assumptions

Our design entails the following assumptions[3]:

1. Each tag is owned and physically attended by a person who understands tag operation and who is reasonably aware of the current date.
2. Each tag is equipped with a one-line alpha-numeric (OLED or ePaper) display capable of showing a 6-8 digit date.
3. Each tag has a mechanism that allows it to become temporarily inaccessible to the reader (i.e., to be "turned off").
4. Each tag is aware of the name and the public key of a system-wide trusted certification authority (CA).
5. The CA is assumed to be infallible: anything signed by the CA is guaranteed to be genuine and error-free.
6. The CA issues an updated revocation structure (e.g., a CRL) periodically. It includes serial numbers of all revoked reader certificates.
7. Each tag knows the periodicity of revocation issuance (i.e., it can calculate the expiration date of revocation status information by knowing its issuance date.)
8. While powered up by a reader, a tag is capable of maintaining a count-down timer.
9. A tag can retain (in its non-volatile storage) the last valid date it encountered.
10. [**Optional**] A tag may have *a single button* for user input.

4.2 Basic Idea

Before providing any information to the reader, a tag has to validate the reader PKC. Recall our assumption that the user is physically near (e.g., holds) his tag during the entire process. Verification is done as follows:

1. The freshly powered-up tag receives the CRL and the reader PKC. Let CRL_{iss}, CRL_{exp}, PKC_{iss} and PKC_{exp} denote issuance and expiration times for purported CRL and PKC, respectively. Let the last valid date stored in the tag be Tag_{Curr}.

[3] Although we use "date" as the revocation/expiration granularity, proposed technique is equally applicable to both coarser- and finer-granular measures, e.g., month or hour.

Fig. 1. A Display and Button Equipped RFID Tag

2. If either CRL_{exp} or PKC_{exp} is smaller than Tag_{curr}, or $CRL_{iss} \geq PKC_{exp}$, the tag aborts.
3. The tag checks whether the CRL includes the serial number of the reader certificate. If so, it aborts.
4. The tag checks the CA signatures of the PKC and CRL. If either check fails, the tag aborts.
5. If CRL_{iss} or PKC_{iss} is more recent than the currently stored date, the tag updates it to the more recent of the two.
6. The tag displays the lesser of the CRL_{exp} and PKC_{exp}. It then enters a countdown stage of fixed duration (e.g., 10 seconds).
7. The user views the date on the display.
   [**OPTION A:**]
   (a) If the displayed date is not in the past, the user does nothing and interaction between the tag and the reader resumes after the countdown stage.
   (b) Otherwise, the user terminates the protocol by initiating an escape action while the tag is still in countdown stage.
   [**OPTION B:**] (If Assumption 10 holds)
   (a) If the displayed date is in the future, the user presses the button on the tag before the timer runs out, and communication with the reader continues normally.
   (b) Otherwise, the timer runs out and the tag automatically aborts the protocol.

4.3 Escape Actions

As evident from the above, an escape action is required whenever the user decides that the displayed date is stale. Although the choice of an escape action is likely to be application-dependent, we sketch out several simple and viable examples.

Using a Button: Recent developments in low-power hardware integration on contactless cards have led to deployment of buttons on RFID tags [20,33]. On such tags, the user can be asked to press a button (within a fixed interval) as a signal of acceptance[4]. If the button is not pressed within that interval, the protocol is automatically terminated by the tag. Thus, the escape action in this case involves no explicit action by the user. We recommend this variant over alternatives discussed below, since it complies with the *safe defaults* design principle, i.e., without explicit approval by the user, the tag automatically aborts its interaction with the reader.

Faraday Cages: A Faraday Cage is a jacket made of highly conductive material that blocks external electric fields from reaching the device it encloses. Since tags are powered by the electric field emitted from a reader, it is theoretically possible to isolate them from all reader access by simply enclosing them in a Faraday cage. For tags that have an enclosing Faraday cage – such as e-Passports that have one inside their cover pages – the natural escape action is simply closing the passport.

Disconnecting Antennas: An RFID tag communicates and receives power through a coil antenna attached to its chip. Disconnecting the antenna from the chip immediately halts communication and shuts down the tag. A simple physical switch placed between a tag and its antenna can be used as an escape action. Similar mechanical actions aimed to halt communication between a tag and a reader are described in [17]. One drawback of such techniques is that physical damage to the tag is possible if the switch is handled roughly.

4.4 Efficient Revocation Checking

Although we hinted at using CRLs earlier in the paper, our approach would work with CRTs or any other off-line revocation scheme. However, both CRLs and CRTs become inefficient as the number of revoked readers increases. CRLs are linear and CRTs – logarithmic, in the number of revoked certificates. Our goal is to minimize bandwidth consumed by revocation information by making it constant, i.e, $O(1)$. To achieve this, we take advantage of a previously proposed modified CRL technique originally intended to provide privacy-preserving revocation checking [29].

In traditional CRLs, the only signature is computed over the hash of the entire list of revoked PKCs. Consequently, the entire list must be communicated to the verifier. To make CRLs bandwidth-optimal, [29] requires the CA or a Revocation Authority to sign each (sorted) entry in a CRL individually and bind it with the previous entry. In more detail, the modified CRL technique works as follows: assume that the CRL is sorted in ascending order by the revoked certificate serial

[4] For tags that have no buttons but built-in accelerometers, gestures (see [8] for more details) can also be used to signal user acceptance.

numbers. For a CRL with n entries, the CA generates a signature for the i-th entry $(1 < i \leq n)$ as follows:

$$Sign(i) = \{h(CRL_{iss}||SN_i||SN_{i-1})\}_{SK_{RA}}$$

where, CRL_{iss} is the issuance date of this current CRL, SN_i is the i-th certificate serial number on the ordered CRL, SN_{i-1} is the immediately preceding revoked serial number, SK_{RA} is the secret key of the CA and h is a suitable cryptographic hash function. To mark the beginning and the end of a CRL, the CA uses two fixed sentinel values: $+\infty$ and $-\infty$.

When authenticating to a tag, a non-revoked reader provides its own PKC as well as the following constant-size non-revocation proof:

$$SN_j,\ SN_{j-1},\ CRL_{iss},\ Sign(j)$$

where reader certificate serial number SN_{rdr} is such that $SN_{j-1} < SN_{rdr} < SN_j$. The reader PKC, along with the above information, allows the tag to easily check that: (1) the range between adjacent revoked certificate serial numbers contains the serial number of the reader PKC, and (2) the signature $Sign(j)$ is valid. If both are true, the tag continues with the protocol by displaying the lesser of the CRL_{exp} and PKC_{exp}, as in step 6 of Section 4.2.

Compared with traditional CRLs, this scheme reduces both storage and communication overhead from $O(n)$ to $O(1)$ for both, readers and tags. On the other hand, the CA has to separately sign each CRL entry. Although this translates into significantly higher computational overhead for the CA, we note that CAs are powerful entities running on resource-rich systems and CRLs are not usually re-issued very frequently, i.e., weekly or daily, but not every minute or even every hour.

4.5 Security Considerations

Assuming that all cryptographic primitives used in the system are secure and the user executes necessary escape actions in case of expired (or revoked) reader certificates, the security of the proposed reader revocation checking mechanism is evident.

We acknowledge that user's awareness of time and ability to abort the protocol (when needed) are crucial for the overall security. To this end, we conducted some usability studies, including both surveys and experiments with a mock implementation. As discussed in section 5, our studies showed that people are reasonably aware of date and also able to execute the protocol with low error rates.

4.6 Cost Assessment

Recent technological advances have enabled mass production of small inexpensive displays (e.g., ePaper) that can be easily powered by high-end RFID tags

aided by nearby readers[5]. The current (total) cost of an ePaper display-equipped and public key-enabled RFID tag is about 17 Euros in quantities of 100,000 and the cost goes down appreciably in larger quantities [33]. Although this might seem high, we anticipate that the cost of cutting-edge passive display technologies (i.e., ePaper and OLED) will sharply decrease in the near future. Moreover, once a display is available, it can be used for other purposes, thus amortizing the expense. We briefly describe some potential alternative uses for display-equipped RFID tags:

Transaction Verification: RFID tags are commonly used as payment and transaction instruments (e.g., credit, ATM and voting cards). In such settings, a direct auxiliary channel between the tag and the user is necessary to verify the details of a transaction. This problem becomes especially apparent with payment applications. A malicious reader can easily fool the tag into signing or authorizing a transaction for an amount different from that communicated to the user. A display on a contactless payment card would solve this problem by showing the transaction amount requested by the reader on its display and waiting for explicit user approval before authorizing it.

Device Pairing: A display may be used for secure pairing of tags with other devices that do not share a CA with the tag. Visual channel-based secure device pairing methods that are proposed for personal gadgets can be used with display-equipped RFID tags (See [21] and [18] for a survey of such methods). The ability to establish a secure ad-hoc connection with arbitrary devices is a new concept for RFID tags that might open doors for new applications, e.g., the use of NFC-capable personal devices (e.g., cell-phones) to change and control settings on personal RFID tags.

User/Owner Authentication: In some scenarios, it might be necessary for a user to authenticate to a tag (e.g., credit card or passport). Currently this can be done only via trusted third party devices such as readers, mobile phones [31], personal computers and wearable beepers [16]. However, in the future, with a display-equipped RFID tag, the need for additional trusted devices might be obviated.

5 Usability

Since the proposed technique requires active user involvement, its usability is one of the key factors influencing its potential acceptance. Also, due to the nature of the protocol, certain type of user errors (i.e., accepting an incorrect or stale date) can result in a loss of security. Thus, we conducted two separate usability studies: online surveys and hands-on usability experiments. The goal of these studies was to answer the following questions:

1. Do everyday users worry about the reader revocation problem?

[5] Power feasibility analysis of integrating a display into a passive RFID tag circuit is discussed in Appendix A.

2. How do these users rate the usability of our solution?
3. Are users reasonably aware of the current date? What are the expected error rates?

5.1 Usability Experiment

In order to assess the usability of our method in the context of real users, 25 subjects were recruited to take part in the usability study. In order to prevent subjects from being explicitly aware of the date during the tests, care was taken to avoid setting up prior test appointments. Instead, subjects were recruited by the test coordinator at various campus venues, e.g., cafés, dorms, classrooms, offices, labs and other similar settings.

Apparatus and Implementation: Our test mock-up was implemented using two mobile phones: a Nokia N95 [2] (simulating the tag) and a Nokia E51 [1] (simulating the reader). These devices were chosen since, at the time of this study, actual RFID tags with displays and buttons could not be ordered in modest quantities. We used *Bluetooth* as the wireless communication medium between the N95 and E51. All implementation code was written in Java Mobile Edition. The time period for the automatic reject was set to 10 seconds.

Subjects: Our study participants were mainly students at the University of California, Irvine. Their age was well distributed among three groups: 36% – 18-24, 32% – 25-29, 32% – 30 +. Gender distribution was controlled for and almost evenly split between male and female (52% and 48%, respectively). On the other hand, 80% of the subjects had a bachelors degree, thus yielding a rather educated sample. We attribute this to the specifics of the study venue (a university campus).

Procedure: To help subjects in understanding the concept of personal RFID tags, the ePassport example was used throughout the test and the questionnaire phases. First, subjects were asked not to consult any source of current date/time before and during the tests. Then, they were given a brief overview of our method and the importance of maintaining natural behavior during the experiments. Next, each subject was presented with a mock-up implementation and was asked to execute the protocol six times. Finally, subject opinions were solicited via the post-test questionnaire.

The set of dates used in the study process was: +/-1 day, -3 days, +7 days, -29 days, and -364 days, from the actual test date (Note that "+" and "-" indicate future and past dates, respectively). All experiments were conducted during the first week of December 2009, and choices of -29 days and -364 days were deliberate so as to make the staleness of these dates more deceiving to the subjects.

Test cases were presented to each subject in random order. The test administrator held the phone simulating the reader and sent dates to the device simulated the tag. After a date was displayed on the "personal tag", the test subject was asked to decide whether to: (1) accept the date by pressing the

button within ten seconds, or (2) reject it by doing nothing. The process was repeated six times for each test-case.

Results

Completion Time and Error Rates: For subjects who accepted displayed dates, the study yielded average completion time of 3.07 seconds, with standard deviation of 1.58 seconds. This shows that subjects were quick in reacting whenever they considered the date to be valid. This also confirms that our choice of a 10-second time-out was appropriate.

Among the 25 subjects, the false negative rate (reject for a date that was not stale) was quite low. No one rejected a date that was one day in future, and only one subject (4% of the sample) rejected the date that was seven days in the future. The false positive rate (accept a stale date) was also low in all cases, except one. When subjects were shown dates that were, respectively: 1, 3 and 29 days earlier, the corresponding observed error rates were 0%, 0% and 4%. However, surprisingly, the error rate spiked up to 40% when subjects were shown a date that was almost a year (364 days) earlier. We discuss this further in Section 5.3 below.

User Opinions: Subjects who tried our mock-up implementation rated its usability at 77% on the original System Usability Scale (SUS) [5], a score that is about 13% higher than that obtained from the on-line survey, where participants rated it solely based on its written description. 84% of the subjects who tested our implementation stated that they would like this system implemented on their own personal tags, while 12% were neutral to the idea (the average score on a 5-point Likert scale was 4.1 with the standard deviation of 0.75).

5.2 On-Line Survey

We created an online survey [3] that was used to anonymously sample 98 individuals. The purpose was to collect information regarding perceived usability and general acceptance of our solution, rather than its actual usability. Participants were given an explanation of the reader revocation problem. Then, they were presented with the detailed description of our approach that included all user interaction.

Survey Results: The proposed technique yielded a score of 68/100 on the system usability scale (SUS). 66% of the participants stated that they would like to see it implemented on their E-passports, while 26% were neutral (the average score on 5-point Likert scale was 3.67 with the standard deviation of 0.87). 84% of the participants were worried about identity theft and 88% stated that they are concerned about revealing personal information to unauthorized parties in general.

In the online survey, we did not ask the subjects for their estimate of the current date or whether a displayed is stale, as this data would have been severely biased owing to the availability of the current date on their computer screen. Instead, participants were asked about their general awareness of the current

date. 40% indicated that they are usually aware of the exact date, 35% were confident to know it with at most one-day error margin and 22% claimed to be within the +/- 3-day range. The remaining 3% indicated that 7 or more days error would be possible on their estimate of the current date.

5.3 Discussion

Based on our usability results, we now attempt to answer the questions raised at the beginning of this section:

Are people concerned with the problem we aim to solve? Among the 123 total participants (98+25, in both studies) 88% are worried about revealing information to unauthorized parties. 70% said that they wanted to see the proposed technique implemented on their personal tags.

How do people rate the usability of our approach? Given the detailed description of the method and required interaction, 98 participants rated its usability at 68% on SUS scale. The usability rating was even higher (77%) for 25 subjects who actually experimented with the mock-up implementation. Both scores are above industry averages [22] and indicate good usability and acceptability characteristics.

Are users aware of current date? As results show, our method very rarely yields false negatives: users are capable of not mistaking valid (future) dates for being in the past. As far as false positives, however, results are mixed. Stale days and months are, for the most part, easily recognized as such. However, with the stale year, the error rate is quite high, at 40%. This deserves a closer look. While we do not claim to know the exact reason(s), some conjectures can be made.

When confronted with a date, most of us are conditioned to first check day and month, e.g., current dates on documents and expiration dates on perishable products. At the same time, users do not tend to pay as much attention to more gross or blatant errors (such as wrong year) perhaps because they consider it to be an unlikely event. Also, we note that among six test-cases for each user, just one had a date with the wrong year. This may have inadvertently conditioned the subjects to pay more attention to the month/day fields of the dates.

On the other hand, we anticipate that year mismatches will be quite rare in practice, since the 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.

Another approach that *may* yield lower error rates is showing *today's date* to the users instead of an expiration date. This approach can be implemented as follows:

1. The reader sends the tag its claimed value for "today's date" (D_{curr}) in addition to its PKC and the most recent CRL.

2. The tag checks that $D_{curr} < PKC_{exp}$ and $D_{curr} < CRL_{exp}$. If either check fails, the tag aborts.
3. The tag displays D_{curr} to the user.
4. The user is now required to verify that the displayed date is indeed "today's date".

We believe more comprehensive user-studies are needed to evaluate whether the above approach or certain changes in date representation and formatting (for e.g., displaying YYYY/MM/DD instead of MM/DD/YY) might help lower user errors.

6 Conclusions

In this paper, we presented a simple and effective method for reader revocation on pk-enabled RFID tags. Our approach requires each tag to be equipped with a small display and be attended by a human user during certificate validation. As long as the user (tag owner) plays its part correctly, our solution eliminates the period of vulnerability with respect to detecting revoked readers.

Recent advances in display technology, such as ePaper and OLED, have already yielded inexpensive display-equipped RFID tags. The low cost of these displays combined with the better security properties and potential new application domains make displays on RFID tags a near reality. Moreover, our usability studies suggest that users find this solution usable and they are capable of performing their roles within reasonable error rates. We believe that display-equipped RFID tags will soon be in mass production and the method proposed in this paper will be applicable to a wide variety of public key-enabled tags.

Acknowledgments. The authors are grateful to Bruno Crispo and Markus Ullman for their valuable comments on the previous version of this paper. This work is supported in part by NSF Cybertrust grant #0831526.

References

1. Nokia e51 specifications, `http://europe.nokia.com/find-products/devices/nokia-e51/specifications`
2. Nokia n95 specifications, `http://www.nokiausa.com/find-products/phones/nokia-n95-8gb/specifications`
3. Display enabled identification and payment instruments (November 2009), `http://sprout.ics.uci.edu/projects/usec/survey.html`
4. Blundo, C., Persiano, G., Sadeghi, A.-R., Visconti, I.: Resettable and Non-Transferable Chip Authentication for ePassports. In: Conference on RFID Security (2008)
5. Brooke, J.: Sus - a quick and dirty usability scale. Usability Evaluation in Industry (1996)
6. Bundesamt fur Sicherheit in der Informationstechnik. Advanced Security Mechanisms for Machine Readable Travel Documents : Version 2.0 (2008)

7. Cheon, J.H., Hong, J., Tsudik, G.: Reducing RFID Reader Load with the Meet-in-the-Middle Strategy. Cryptology ePrint Archive, Report 2009/092 (2009)
8. 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: Computer and Communications Security – CCS (2008)
9. Goodrich, M., Tamassia, R.: Efficient authenticated dictionaries with skip lists and commutative hashing, US Patent App. 10/416,015 (May 7, 2003)
10. Heydt-Benjamin, T., Bailey, D., Fu, K., Juels, A., O'hare, T.: Vulnerabilities in first-generation RFID-enabled credit cards. Financial Cryptography and Data Security (2007)
11. Hoepman, J.-H., Hubbers, E., Jacobs, B., Oostdijk, M., Wichers Schreur, R.: 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)
12. Housley, R., Ford, W., Polk, W., Solo, D.: RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile (January 1999)
13. Infineon Technologies AG, AIM CC. Preliminary Short Product Information: Chip Card and Security IC's (2006)
14. International Civil Aviation Organization. Machine Readable Travel Documents: Specifications for Electronically Enabled Passports with Biometric Identification Capability (2006)
15. Juels, A., Molnar, D., Wagner, D.: Security and privacy issues in e-passports. In: Security and Privacy for Emerging Areas in Communications Networks – SECURECOMM (2005)
16. Kaliski, B.: Future directions in user authentication. In: IT-DEFENSE (2005)
17. Karjoth, G., Moskowitz, P.A.: Disabling rfid tags with visible confirmation: clipped tags are silenced. In: Workshop on Privacy in the Electronic Society – WPES (2005)
18. Kobsa, A., Sonawalla, R., Tsudik, G., Uzun, E., Wang, Y.: Serial hook-ups: a comparative usability study of secure device pairing methods. In: Symposium on Usable Privacy and Security – SOUPS (2009)
19. Kocher, P.C.: On certificate revocation and validation. In: Hirschfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 172–177. Springer, Heidelberg (1998)
20. Kugler, D., Ullman, M.: Contactless security tokens - enhanced security by using new hardware features in cryptographic based security mechanisms. In: Dagstuhl Seminar Proceedings of Foundations for Forgery - Resilient Cryptographic Hardware (July 2009)
21. Kumar, A., Saxena, N., Tsudik, G., Uzun, E.: Caveat eptor: A comparative study of secure device pairing methods (2009)
22. Lewis, J., Sauro, J.: The factor structure of the system usability scale. In: Proceedings of the Human Computer Interaction International Conference (HCII 2009), San Diego CA, USA (2009)
23. Merkle, R.C.: Secrecy, authentication, and public key systems. Technical report, Stanford University (1979)
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, US Patent 5,666,416 (September 1997)
26. Monnerat, J., Vaudenay, S., Vuagnoux, M.: About Machine-Readable Travel Documents. In: Conference on RFID Security (2007)
27. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: Internet public key infrastructure online certificate status protocol- ocsp (1999)

28. Naor, M., Nissim, K.: Certificate revocation and certificate update. Technical report (1999)
29. Narasimha, M., Solis, J., Tsudik, G.: Privacy preserving revocation checking. International Journal of Information Security 8(1), 61–75 (2009)
30. Oren, Y., Feldhofer, M.: A Low-Resource Public-Key Identification Scheme for RFID Tags and Sensor Nodes. In: ACM Conference on Wireless Network Security – WiSec (2009)
31. Saxena, N., Uddin, M. B., Voris, J.: Treat 'em like other devices: user authentication of multiple personal rfid tags. In: SOUPS (2009)
32. Scholz, P., Reihold, C., John, W., Hilleringmann, U.: Analysis of energy transmission for inductive coupled rfid tags. In: International Conference on RFID (2007)
33. Ullman, M.: Personal communication (September 2009)

A Power Feasibility Analysis

The aim of this section is to show that it is completely feasible to integrate low power display technologies on passive RFID tags without any change on reader specifications. We analyze the maximum power requirements of the proposed system and its effect on the (theoretical) maximum working distance with current readers. In the rest of this section, we use ePassports as an example due to their clear tag and reader specifications.

We propose the use of display technologies such as ePaper, OLED, and other such low-power bistable displays. These displays require power of the order of 100mW (for a 2" display unit) during display updates and 0mW of power during standby.

A.1 Power Analysis

ePassport tags such as those supplied by Infineon Technologies, require up to 55mW of power to operate [13] while the display unit requires a maximum power of 100mW to operate. We analyze the power requirements of the proposed system from three aspects:

1. The ePassport tag is operating at maximum power and the display unit is static or non-existent.
2. The ePassport tag is on standby and the display unit is being updated (*i.e.*, refreshed).
3. The ePassport tag is operating at maximum power and the display unit is being updated (*i.e.*, refreshed).

In the first case, the power required by the ePassport circuit to operate will be $\sim$ 55mW (the power required by the display unit at this time is zero). In the second case, the power required by the ePassport circuit to operate will be $\sim$100mW (the power required by the tag during standby is negligible). In the final case, the power required by the ePassport circuit to operate will be $\sim$155mW (the sum of the maximum power required by the tag and display).

The ePassport tag and reader when placed parallel to each other can be represented as a circuit, with circuit parameters set in the manner described by Scholz *et al.* [32].

First, we establish a relationship between the mutual inductance (M) and the distance (x) between the antenna of the tag and the reader.

$$M = \frac{\mu\pi N_1 N_2 (r_1 r_2)^2}{2\sqrt{(r_1^2 + x^2)^3}} \tag{1}$$

Where μ is the Permeability $[H/m]$; N_1 and N_2 are the number of turns in the antennas of the tag and reader; r_1 and r_2 are the radii $[mm]$ of each of these turns. Substituting default values [32] we get the relation

$$M = \frac{1.57 \times 10^{-12}}{x^3} \tag{2}$$

Now we establish a relationship between the power required by the tag (P_{Tag}) and distance (x). This is done through the series of equations below.

$$P_{Tag} = I_1^2 R_T \tag{3}$$

Where I_1 is the current running in the reader circuit $[mA]$ and R_T represents the tag impedance which is given by (4).

$$R_T = \frac{M^2 R_L}{L_2^2} \tag{4}$$

Where L_2 is assigned a value of 168nH [32] and R_L is the load resistance given by (5).

$$R_L = \frac{V_T^2}{P_{Tag}} \tag{5}$$

V_T is the voltage required in the tag circuit (5.5 Volts). The value of R_L is 195.1 Ω in the case that the ePassport tag and display unit operate at maximum power together (case 3). R_L is 302.5 Ω in the case that the ePassport tag is on standby when the display unit is refreshed (case 1). Finally, by combining equations 2 through 5, we can get a relationship between x and P_{Tag}.

$$x^6 = \frac{(1.57 \times 10^{-12})^2 \times (I_1)^2 \times (R_L)}{P_{Tag} \times (L_2)^2} \tag{6}$$

Making the necessary substitutions, we get the following values for x, where x represents the maximum possible operating distance:

– An ePassport tag without a display unit or with display on stand-by (*i.e.*, not refreshing):

$$P_{Tag} = 55\ mW,\ R_L = 550\ \Omega \Longrightarrow x = .097\ m \tag{7}$$

– An ePassport display unit while refreshing output when the tag is in standby mode:

$$P_{Tag} = 100\ mW,\ R_L = 302.5\ \Omega \Longrightarrow x = .080\ m \tag{8}$$

– An ePassport tag and the display unit operating at maximum power:

$$P_{Tag} = 155\ mW,\ R_L = 195.1\ \Omega \Longrightarrow x = .069\ m \tag{9}$$

From the above results it is clear that even with the current reader and antenna specification, adding a display reduces the maximum operating distance between the tag and reader only by 2.8 cm. Therefore, adding a display unit to the current ePassport circuit is feasible and doesn't require any changes over the power specifications in the original proposal [6]. If longer operating distances (over 6.9 cm) are needed, it can be achieved with small modifications on the RFID antenna design or by increasing power of a reader.

Privacy-Preserving, Taxable Bank Accounts

Elli Androulaki, Binh Vo, and Steven Bellovin

Columbia University
{elli,binh,smb}@cs.columbia.edu

Abstract. Current banking systems do not aim to protect user privacy. Purchases made from a single bank account can be linked to each other by many parties. This could be addressed in a straight-forward way by generating unlinkable credentials from a single master credential using Camenisch and Lysyanskaya's algorithm; however, if bank accounts are taxable, some report must be made to the tax authority about each account. Assuming a flat-rate taxation mechanism (which can be extended to a progressive one) and using unlinkable credentials, digital cash, and zero knowledge proofs of knowledge, we present a solution that prevents anyone, even the tax authority, from knowing which accounts belong to which users, or from being able to link any account to another or to purchases or deposits.

1 Introduction

One of the hardest realms in which to achieve privacy is finance. Apart from the obvious — few transactions are made via cash or other anonymous payment mechanisms — society often requires that other information about bank accounts be disclosed. In the U.S., for example, banks and other financial institutions are required to report interest or dividend payments, since they are generally considered to be taxable income. Some jurisdictions require that a portion of the interest be paid directly to the government; other jurisdictions impose taxes on actual balances. These requirements conflict with a desire for privacy and suggesting a way to combine the two is the topic of this paper.

Pseudonymity as Privacy Mechanism. One particular aspect of the conflict concerns a very common technique for achieving transactional privacy: pseudonymity. In pseudonymous systems, an individual has a multitude of separate, unlinkable identities that can be used as desired. A separate pseudonym can be used for each peer, thus preventing linkage between different sorts of activities.

We claim that pseudonymity may be adopted in the banking system to achieve privacy, as, at least for tax purposes, neither banks nor the government need to know who owns a particular bank account. In fact, there are both security and privacy benefits to having multiple pseudonymous accounts. Often, knowledge of a "routing number" (effectively, the bank's identity) and an account number are sufficient to *withdraw* money from an account as well as deposit money into

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 37–54, 2010.

it. Having multiple pseudonymous accounts — and closing those created for a special purpose when they are no longer needed — could prevent such incidents.

The Challenge. Although the identity of the account owners is not a functional requirement of the banking system, pseudonymity may harden authorization and encourage fairness attacks as it lacks accountability. Banks need to know that only authorized parties withdraw money from accounts; governments need ensure that balances and income are properly reported, and taxes paid. An ideal system would be one where an individual could open a bank account without disclosing his or her real identity; nevertheless the relevant tax authorities would receive accurate reports of relevant information.

Our Contribution. We present a solution that accomplishes these goals. Individuals need present strong identification credentials only to obtain a single membership to the bank, after which he may open an arbitrary number of anonymous or nominal accounts, without anyone being able to link those accounts to their owner or one to the other. Periodically, appropriate information on taxable items is supplied; the tax authority can verify that all accounts are properly reported. Our protocols consider non-progressive taxes, can easily cover progressive ones, and are secure under the strong RSA assumption.

Organization. In Section 2, we give a more precise statement of the architecture and requirements for the system. The protocols are described in Section 3. We explain why we believe this system to be secure in Section 4. Section 5 discusses related work; Section 6 has concluding thoughts.

2 System Architecture

Our goal is to build a bank account mechanism, where an honest individual is able to own and handle multiple anonymous bank accounts not entirely connected to his identity, while being taxed fairly and privately. More specifically, we consider an architecture consisting of *Users*, i..e, individuals who open bank accounts and must pay taxes, *Banks*, who allow users to open accounts for the purpose of storing cash and handling financial transactions, while responsible for reporting interest for income tax purposes, and the *Tax Authority* (TA), which is responsible for ensuring that correct income taxes are paid by all users. Tax Authority corresponds to the U.S.'s Internal Revenue Service (IRS), the Canada Revenue Agency, the U.K.'s HM Revenue & Customs, etc.

Accountability requires that a single identity should be accounted for repetitive misbehaviors of the same user. To restrict registration entries to one per user, banks require that users present at Registration strong unforgeable identification credentials. Users may then open and manage two types of bank accounts: *nominal* accounts, which carry the identity their owner and whose activity can be fully traced, and *anonymous* accounts, where the identity of the owner is concealed to the bank. To handle the the data related to these accounts, the bank maintains $\mathsf{D}_{\mathsf{reg}}$ and D_{α} databases, respectively. Anonymous accounts may be opened

(AccountOpen operation) after special user-requests. To manage their accounts, users are required to create pseudonyms strongly (but secretly) connected to their bank identity. On an annual basis, all accounts are withheld taxes which are reported in a four stage procedure: the TaxReportIssue, where the account owner and the bank issue an anonymous account's report, the TaxReportTranform, where the user transforms his report into a different but valid form, the TaxReportDeposit, where the user deposits the tax report to the tax authority and TotalTaxCalculation, where all the reports of the user are used to extract the overall tax withheld. Auxiliary operations of the system are the Bkeygen and Ukeygen, the key generation algorithms for the bank and the users respectively. See [AVB10] for more details.

Threat Model. We make the following assumptions:

1. Users may try to cheat. A user trying to avoid paying taxes may lie regarding the tax withheld, attempt any type of forgery in tax reporting. Also, malicious users may try to collaborate in order to minimize the reported balance, as long as they do not endanger their funds.
2. Banks are "honest but curious". Aiming to maintain their clientele, banks are trusted to perform all their functional operations correctly, i.e., they issue credentials, open and update accounts as instructed by their customers. However, they may use the information they possess for other reasons, i.e., to sell credit card based profiles to advertising companies, while they may collaborate with tax authority to reveal the identity behind an anonymous account.
3. Tax Authority is considered "honest but curious". Although we assume that it is operated by the government who wants to protect honest users, it, however, is not assumed to protect privacy; indeed, there have been a number of incidents in the U.S. of privacy violations by tax authorities or by unscrupulous individuals employed by the tax authorities.

Requirements. *Privacy* and *security* are the core requirements of our system. Privacy refers to the user, while security guarantees the proper operation of the entire system. Privacy requires that the activity of an honest individual is untraceable to other users, banks, the tax authority or collaborations between them, and, thus, translated to:

1. Account-Account-owner Unlinkability. It can be inducted to account-pseudonym — account-owner unlinkability and account – account tax-report unlinkability. The first requires that given a pseudonym $\mathsf{P_U}$ that does not belong to a corrupted party, the adversary can learn which user owns $\mathsf{P_U}$ no better than guessing at random among all non-corrupted users that appear consistent with $\mathsf{P_U}$. Account – account tax-report unlinkability requires that given a tax report T that does not belong to a corrupted party, the adversary can learn which user owns T no better than guessing at random among all non-corrupted users in D_α. In addition, given a transformed tax report TT, that does not belong to a corrupted party,

the adversary can learn which pseudonym (and thus which account) it corresponds to no better than guessing at random among all pseudonyms (accounts) of non-corrupted users in D_α.

2. Account-Account Unlinkability. It can similarly be analyzed to account-pseudonym – account-pseudonym unlinkability and account-tax report – account tax-report unlinkabilty. The first requires that given two pseudonyms $\mathsf{P}^1_\mathsf{U}, \mathsf{P}^2_\mathsf{U}$ that do not belong to corrupted parties, the adversary has no advantage in telling whether $\mathsf{P}^1_\mathsf{U}, \mathsf{P}^2_\mathsf{U}$ belong to the same user or not. The second, that given two recently generated tax reports T^1, T^2 that do not belong to corrupted parties, the adversary has no advantage in telling whether they belong to the same user or not. There should be no way for any entity or collaboration of entities, including the bank and tax authority, to link different accounts of the same user.

Security is defined as the combination of the following properties:

1. Fairness. Suppose that n users $\mathsf{U}_1, \ldots, \mathsf{U}_n$ collude together. Let the sum of the tax withheld by all of them together is $\mathsf{SumTax} = \sum_{i=1\ldots n} TotalTax^i$, where $TotalTax^i$ is U_i's tax amount withheld. Fairness requires that the group of users may report in total at least SumTax. We also require that the following hold

a. Tax Report non-Transferability. No user should be able to use (deposit) the tax report of another user. Assuming two corrupted users U_1 and U_2, where U_1 has issued T^1. Tax Report non-Transferability requires that there is no valid transformation TT^1 of T^1 (through $\mathsf{TaxReportTransform}$) for which the following happens with non negligible probability: if U_2 attempts to deposit TT^1 in honest B through $\mathsf{TaxReportDeposit}$, B accepts.
b. Tax Report Unforgeability. No user or coalition of users should be able to construct a valid tax report for his accounts, i.e., a tax report for which $\mathsf{TaxReportDeposit}$ is accepted by the tax authority or the bank.

2. Accountability. Users who attempt to avoid paying taxes for their accounts are traced and punished.

3 Taxation Protocol

Accountability poses important a form of "privacy-preserving" *centralization* of user activity inside the bank. Thus, each user can be privately authenticated by demonstrating knowledge of a master secret, ms_U, which he generates at the registration procedure. Users are highly motivated not to share their secret, which they use to open and manage their anonymous accounts. More, specifically the user utilizes his ms_U to issue single use, bank (blindly) authorized permissions perm_α, which he later deposits anonymously to open accounts. To manage them, the anonymous user generates account pseudonyms, which are secretly, but provably, connected to their owner's ms_U. Users deposit annually to the tax authority an number of tax reports equal to the number of accounts they own. Tax reporting consists of two phases:

1. Tax Report Generation. It involves three stages:

1. *Tax-Report-number Acquisition*, where account owners obtain one valid tax-report-number (TRN) per account. It is important to note that TRNs are not linked to the accounts they are used for.
2. *Actual Report Generation*, where the account pseudonym proves that it is the owner of the account — by demonstrating knowledge of the corresponding $\mathsf{ms_U}$ — and provides a verifiable commitment to both his $\mathsf{ms_U}$ and TRN. The bank *signs* it and produces the prime version of the account's tax report: $(T^\sigma, T^M) = (\mathsf{Sig}^x_\mathsf{B}(\text{TaxInfo}), \text{TaxInfo})$, where TaxInfo $= Tax \mid\mid \text{Commit}(\mathsf{TRN}, \text{Master-Secret})$, Tax is the tax withheld from the user's account and by $\mathsf{Sig}^x_\mathsf{B}(M)$, we denote a complicated procedure which involves bank's (x-multiple) signature on M. The exact number of bank signatures applied on M is not revealed to the user. However, the bank provides the user with a randomized token SigInfo which contains that information, in a form only readable by the taxation authority, along with re-randomization information SITranform for the user to make SigInfo unlinkable to its initial form.
3. *Tax Report Transformation*, where the account owner, applies a transformation function F to both, the bank-signed tax report T^σ, and the corresponding unsigned message, ending up to the depositables $TT^\sigma = F(T^\sigma)$, and $TT^M = F(T^M)$. The account owner also transforms SigInfo through SITranform.

2. Tax Report Deposit. Each user deposits all of his tax reports to the bank. The deposit of tax reports includes three stages:

1. *Deposit of all the unused* perm_α. In this way, the bank can accurately compute the number of anonymous accounts of each user.
2. *Deposit of the depositable tax report pairs*, $(TT^{\sigma,i}, TT^{M,i})$ corresponding to each account of the accounts owned by the user.The user proves that each tax report pair is valid, i.e., that it corresponds to bank signature(s) (according to the transformed version of SigInfo), that was constructed using the same user's master secret and that it has not been deposited before.
3. *Tax Amount Calculation procedure*. The bank collaborates with the user to calculate the overall tax withheld the latter's accounts..

3.1 Building Blocks — Primitives for the Suggested Solution

Ecash. An E-Cash [CHL05] system consists of three types of players: the *bank*, *users*, and *merchants*. The input and output specifications of the basic operations are as follows. For convenience, we will assume that the operations take place between a merchant M, a user U and the Bank B.

- $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{EC.BKeyGen}(1^k, params)$ and $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{EC.UKeyGen}(1^k, params)$, which are the key generation algorithm for the bank and the users respectively.
- $\langle W, \top \rangle \leftarrow \mathsf{EC.Withdraw}(\mathsf{pk_B}, \mathsf{pk_U}, n)$ $[\mathsf{U}(\mathsf{sk_U}), \mathsf{B}(\mathsf{sk_B})]$. U withdraws a wallet W of n coins from B.

- $\langle W', (S, \pi) \rangle \leftarrow \mathsf{EC.Spend}(\mathsf{pk_M}, \mathsf{pk_B}, n)$ $[\mathsf{U}(W), \mathsf{M}(\mathsf{sk_M})]$. U spends a coin with serial S from his wallet W to M. W is then reduced to W', M obtains a coin (S, π), where π is a proof of a valid spent coin with a serial S.
- $\langle \top/\bot, L' \rangle \leftarrow \mathsf{EC.Deposit}(\mathsf{pk_M}, \mathsf{pk_B})$ $[\mathsf{M}(\mathsf{sk_M}, S, \pi), \mathsf{B}(\mathsf{sk_B}, L)]$. M deposits a coin (S, π) into its account in B. If successful, M's output will be $\top$ and the B's list L of the spent coins will be updated to L'.
- $(\mathsf{pk_U}, \Pi_G) \leftarrow \mathsf{EC.Identify}(params, S, \pi_1, \pi_2)$. It outputs the public key of the violator U, who spent a coin with serial S twice, producing validity proofs π_1 and π_2, and a proof of guilt Π_G.
- $\top/\bot \leftarrow \mathsf{EC.VerifyGuilt}(params, S, \mathsf{pk_U}, \Pi_G)$. This algorithm, given Π_G publicly verifies the violation of $\mathsf{pk_U}$.
- $\{(S_i, \Pi_i)\}_i \leftarrow \mathsf{EC.Trace}(params, S, \mathsf{pk_U}, \Pi_G, D, n)$. This algorithm provides the list of serials S_i of the coins a violator $\mathsf{pk_U}$ has issued, with the corresponding ownership proofs Π_i.
- $\top/\bot \leftarrow \mathsf{EC.VerifyOwnership}(params, S, \Pi, \mathsf{pk_U}, n)$. Given a ownership proof Π it verifies that a coin with serial number S belongs to a user with public key $\mathsf{pk_U}$.

Security Properties: *(a) Correctness. (b) Balance.* No collection of users and merchants can ever spend more coins than they withdrew. *(c) Identification of Violators.* Given a violation and the corresponding proofs of guilt, the violator's public $\mathsf{pk_U}$ key is revealed such that EC.VerifyGuilt accepts. *(d) Anonymity of users.* The bank, even when cooperating with any collection of malicious users and merchants, cannot learn anything about a user's spendings. *(e) Exculpability.* An honest user U cannot be accused for conducting a violation such that EC.VerifyGuilt accepts. *(f) Violators' Traceability.* Given a violator U with a proof of violation Π_G, this property guarantees that EC.Trace will output the serial numbers of all coins that belong to U with the corresponding ownership proofs.

Pseudonym Systems. Pseudonym systems have three types of players: *users, organizations, and verifiers.* Users are entities that receive credentials. Organizations are entities that grant and verify the credentials of users. Finally, verifiers are entities that verify credentials of the users. See [LRSW99, CL01] for more details. The standard operations supported are the following:

- $(\mathsf{pk_O}, \mathsf{sk_O}) \leftarrow \mathsf{PS.OKeyGen}(1^k)$. This procedure generates a public/secret key pair for an organization. We denote a key pair for an organization O by $(\mathsf{pk_O}, \mathsf{sk_O})$.
- $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{PS.UKeyGen}(1^k)$. This procedure generates a public/secret key pair for a user. We denote a key pair for a user U by $(\mathsf{pk_U}, \mathsf{sk_U})$. Sometimes we refer to the secret key of a user as a master secret key for the user.
- $\langle (N, \mathsf{NSecr}_N), (N, \mathsf{NLog}_N) \rangle \leftarrow \mathsf{PS.FromNym}(\mathsf{pk_O})$ $[\mathsf{U}(\mathsf{pk_U}, \mathsf{sk_U}), \mathsf{O}(\mathsf{sk_O})]$. This interactive procedure between a user and an organization generate a pseudonym (or simply nym). The common input is the public key of the organization O. The output for the user is a nym N and some secret information NSecr_N, and for the organization the nym N and some secret information NLog_N.
- $\langle \mathsf{cred}_N, \mathsf{CLog}_{\mathsf{cred}_N} \rangle \leftarrow \mathsf{PS.GrantCred}(N, \mathsf{pk_O})$ $[\mathsf{U}(\mathsf{pk_U}, \mathsf{sk_U}, \mathsf{NSecr}_N), \mathsf{O}(\mathsf{sk_O}, \mathsf{NLog}_N)]$. This interactive procedure between a user and an organization generate a credential for a nym N. The common input is N and $\mathsf{pk_O}$. The output for the user is the credential cred_N for the nym N. The output for the organization is some secret information $\mathsf{CLog}_{\mathsf{cred}_N}$ for the credential.
- $\langle \top, \top \rangle / \langle \bot, \bot \rangle \leftarrow \mathsf{PS.VerifyCred}(\mathsf{pk_O})$ $[\mathsf{U}(N, \mathsf{cred}_N), V]$. In this interactive procedure between a user and a verifier, the user proves that he has a credential on the nym N issued by the organization O.

– $\langle \top, \top \rangle / \langle \bot, \bot \rangle \leftarrow$ $\mathsf{PS.VerifyCredOnNym}\ (N, \mathsf{pk_O}, \mathsf{pk}_{O_1})\ [\mathsf{U}(N_1, \mathsf{cred}_{N_1}), \mathsf{O}(\mathsf{NLog}_N)]$. In this interactive procedure between a user and the organization O, the user proves that N is his valid nym of the organization O and that he has a credential cred_{N_1} on the nym N_1 issued by the organization O_1.

Security Properties. (a) Unique User for Each Pseudonym. Even though the identity of a user who owns a nym must remain unknown, the owner should be unique. *(b) Unlinkability of Pseudonyms.* Nyms of a user are not linkable at any time better than by random guessing. *(c) Unforgeability of Credentials.* A credential may not be issued to a user without the organization's cooperation. *(d) Non-Transferability.* Whenever a user U_1 discloses some information that allows a user U_2 to use her credentials or nyms, U_1 is effectively disclosing her master secret key to him.

Commitment Schemes. In a typical commitment scheme, there are provers (let each be P) who are required to commit to a particular value towards verifiers (let each be V), who may be able to see the committed value when provers decide to. The procedures supported are the following:

- $(params) \leftarrow \mathsf{CS.Setup}(1^k)$, which outputs the parameters of a commitment scheme.
- $(\mathsf{C}/false) \leftarrow \mathsf{CS.Commit}(params)[\mathsf{P}(r, m)]$. It outputs either the commitment itself to a value m or *not-completed*. P's input is the message m and randomness r.
- $\langle \top/\bot, m/\bot \rangle \leftarrow \mathsf{CS.Open}(\mathsf{C})[\mathsf{P}(m), \mathsf{V}]$. In this operation the P shows the committed value m to V. V accepts it if m is the value matching C.

Security Properties: (a) Binding. It should be computationally impossible for P, after having committed to m, to generate another message m' that has the same commitment value C. *(b) Hiding.* It should be computationally impossible for a verifier who knows C to get any information regarding m.

Blind Signatures. In a typical blind signature scheme, there are signers (let each be S) who produce blind signatures on messages of users (let each be U). The following procedures are supported:

- $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{BS.KeyGen}(1^k)$. This is a key-generation algorithm that outputs a public/secret key-pair $(\mathsf{pk_S}, \mathsf{sk_S})$.
- $\langle \top/\bot,\ \sigma/\bot \rangle \leftarrow \mathsf{BS.Sign}(\mathsf{pk_S})[\mathsf{S}(\mathsf{sk_S}),\ \mathsf{C}(m)]$. At the end of this interactive procedure, the output of the S is either *completed* or *not-completed* and the output of U is either the signature (σ) or a failure sign $(\bot)$.
- $\langle \top/\bot \rangle \leftarrow \mathsf{BS.Verify}(m, \sigma, \mathsf{pk_S})$ is a verification algorithm.

Security Properties: (a) Unforgeability. No one but the signer should be able to produce a valid signature σ on a blinded message m. *(b) Blindness* S does not learn any information about the message m on which it generates a signature σ.

Zero Knowledge Proof of Knowledge Protocols. In a typical zero knowledge proof of knowledge(ZKPOK) scheme there are two types of players, the provers who need to prove possession of one or more secret number(s), that satisfy a particular property to one or more verifiers and the verifiers. In what follows, we will use the notation introduced by Camenisch and Stadler in [CS97] for the various proofs of knowledge of discrete logarithms and proofs of the

validity of statements about discrete logarithms. In particular, $PK\{(\alpha, \beta, \gamma) : y_1 = g_1^{\alpha} h_1^{\beta} \wedge y_2 = g_2^{\alpha} h_2^{\gamma} \wedge (u \leq \alpha \leq u)\}$ denotes a "zero-knowledge-proof-of-knowledge" of integers α, β and γ such that $y_1 = g_1^{\alpha} h_1^{\beta}$ and $y_2 = g_2^{\alpha} h_2^{\beta}$, where $u \leq \alpha \leq u$ and $y_1, g_1, h_1, y_2, g_2, h_2$ are all elements of two groups G_1 and G_2 respectively. We make use of the following ZKPoK schemes:

A proof of knowledge of a representation of an element $y \in G$ with respect to bases $z_1, \ldots, z_v \in G$ [CEVDG88], i.e.,
$PK\{(\alpha_1, \ldots, \alpha_v) : y = z_1^{\alpha_1} \cdot \ldots \cdot z_v^{\alpha_v}\}$.
A proof of equality of discrete logarithms of $y_1, y_2 \in G$ to the bases $g, h \in G$ respectively, [C91, CP93] i.e., $PK\{(\alpha) : y_1 = g^{\alpha} \ \wedge \ y_2 = h^{\alpha}\}$.
A proof of knowledge of a discrete logarithm of $y \in G$ with base $g \in G$ such that $log_g y$ lies in the interval [a,b], [B00],i.e.,
$PK\{(\alpha) : y = g^{\alpha} \wedge \alpha \in [a, b]\}$.
Proof of knowledge that the discrete logarithms of two group elements $y_1 \in G_1$, $y_2 \in G_1$ to the bases $g_1 \in G_1$ and $g_2 \in G_2$ in the different groups G_1 and G_2 are equal [BCDG88, CM99], i.e.,
$PK\{(\alpha, \beta) : y_1 =^{G_1} g_1^{\alpha} \ \wedge \ y_2 =^{G_2} g_2^{\alpha} \wedge C =^{G} g^{\alpha} h^{\beta} \ \wedge \ \alpha \in [0, min(q_1, q_2)]\}$, where q_1, q_2 are the order of the groups G_1, G_2 respectively, $G =< g >=< h >$ is a group to which the commitment C of α, β is computed.

Security Properties.(a) Correctness. (b) Zero-Knowledge. The verifier learns nothing other than that the prover knows the relevant values. *(c) Proof of Knowledge.* The protocol accepts iff the prover knows the secret value he claims to know.

3.2 Detailed Protocol Description

As mentioned before, the bank manages two different registries: one handling users' non-anonymous information (reg-setting) and accounts and another one handling anonymous accounts (α-setting). As each setting is realized as organizations in pseudonymous systems (see [CL01] for more details), the bank runs PS.OKeyGen twice, once for the reg-setting and once for the α setting. In particular, the bank:

- generates all the common system parameters (see [CL01]): the length of the RSA moduli ℓ_n, the integer intervals $\Gamma =]-2^{\ell_\Gamma}, 2^{\ell_\Gamma}[$, which is basically the interval master-secrets belong to, $\Delta =]-2^{\ell_\Delta}, 2^{\ell_\Delta}[$, $\Lambda =]-2^{\ell_\Lambda}, 2^{\ell_\Lambda + \ell_\Sigma}[$, such that $\ell_\Delta = \epsilon(\ell_n + \ell_\Lambda) + 1$, where ϵ is a security parameter, and $\ell_\Lambda > \ell_\Sigma + \ell_\Delta + 4$.
- chooses two pairs (one for each setting) of random $\ell_n/2$-bit primes: p'_x, q'_x, such that $p_x = 2p'_x + 1$ and $q_x = 2q'_x + 1$ are prime and sets modulus $n_x = p_x \cdot q_x$, where $x = \mathsf{reg}, \alpha$.
- chooses random elements $a_x, b_x, d_x, g_x, h_x \in QR_{n_x}$, where $x = \mathsf{reg}, \alpha$. In addition to the standard organization setup procedure of [CL01], the bank chooses for the α-setting random $k_\alpha, l_\alpha, m_\alpha, s_\alpha, z_\alpha \in QR_{n_\alpha}$.

Thus, the Bank's public-secret information for the two settings are

- $\{(n_{\mathsf{reg}}, a_{\mathsf{reg}}, b_{\mathsf{reg}}, d_{\mathsf{reg}}, g_{\mathsf{reg}}, h_{\mathsf{reg}}), (p_{\mathsf{reg}}, q_{\mathsf{reg}})\}$, for the reg-setting, and
- $\{(n_\alpha, a_\alpha, b_\alpha, d_\alpha, g_\alpha, h_\alpha, k_\alpha, l_\alpha, m_\alpha, s_\alpha, z_\alpha), (p_\alpha, q_\alpha)\}$, for the α-setting.

In addition to the aforementioned parameters, the bank generates a blind signature key pair $(\mathsf{pk}_{\mathsf{B}}^{b}, \mathsf{sk}_{\mathsf{B}}^{b})$ and an RSA signature key pair, $(sk_{\mathsf{B}}, pk_{\mathsf{B}}) = (\{d,\ p_\alpha,\ q_\alpha\}, \{e,\ n_\alpha\})$, based on the α RSA-parameters and $1 < e < \phi(p_\alpha q_\alpha)$ and $de = 1(mod(\phi(p_\alpha q_\alpha)))$. e is given to the taxation authority (TA). On the other hand, TA generates an encryption key pair ($\mathsf{pk}_{\mathsf{TA}}$, $\mathsf{sk}_{\mathsf{TA}}$) of a known randomized (and re-randomizable) encryption scheme (Paillier etc) and provides the bank with the encryption key (see Appendix A.3 or [PP99] for more details).

Registration. Assuming collaborations between a user U and a bank B, in the registration procedure, U contacts B *in person* to create an entry in B's $\mathsf{D}_{\mathsf{reg}}$ registry. In particular

1. U → B: strong identification credential, i.e., passport, id card etc.
2. U:runs PS.UKeyGen to obtains a bank-oriented master secret ms_{U} and a public/secret key pair $\{\mathsf{pk}_{\mathsf{U}}^{\mathsf{B}}, \mathsf{sk}_{\mathsf{U}}^{\mathsf{B}}\}$ connected to his ms_{U}.
3. U: runs PS.FormNym using the reg-parameters to generate a registration pseudonym $\mathsf{P}^{\mathsf{reg}}$, connected to ms_{U} in zero knowledge fashion.
4. U ↔ B: execute EC.Withdraw procedure for U (see 3.1 for more details) to withdraw a wallet $\mathsf{WAcc}_{\mathsf{U}}^{\mathsf{B}}$ of perm_{α} (ecoins). $\mathsf{WAcc}_{\mathsf{U}}^{\mathsf{B}}$ will later authorize U to open anonymous accounts in B. Consequently, the size of the wallet withdrawn depends on the maximum number of anonymous accounts U is eligible for.
5. U ↔ B: execute PS.GrantCred procedure so that U obtains a registration credential $\mathsf{cred}_{\mathsf{U}}^{\mathsf{B}}$ for having registered in $\mathsf{D}_{\mathsf{reg}}$, which is provably connected to ms_{U}.
6. U stores in his database his secret key ($\mathsf{sk}_{\mathsf{U}}^{\mathsf{B}}$), the information related to his pseudonym($\mathsf{pubP}^{\mathsf{reg}}$, $\mathsf{secP}^{\mathsf{reg}}$) and credentials ($\mathsf{pubcred}_{\mathsf{U}}^{\mathsf{B}}$, $\mathsf{seccred}_{\mathsf{U}}^{\mathsf{B}}$), while B stores only the public information.

Account Opening. To open an anonymous account, user U contacts B initially anonymously. Both, B and U make use of the α-parameter group. The following interactions take place:

1. U(anonymous) ↔ B: run EC.Spend for U to spend an ecoin (S, π) (perm_{α}) from his $\mathsf{WAcc}_{\mathsf{U}}^{\mathsf{B}}$ wallet. If the ecoin used has been spent before, B runs the EC.Identify and EC.Trace procedures to recover U's identity($\mathsf{pk}_{\mathsf{U}}^{\mathsf{B}}$) and activity ($\mathsf{sk}_{\mathsf{U}}^{\mathsf{B}}$).
2. U: runs PS.FormNym, to generate a pseudonym P^{i} for managing his new account α^i. The pseudonym created has the form of $\mathsf{P} = a_\alpha^{\mathsf{ms}_{\mathsf{U}}} b_\alpha^s$, where s is a U-B-generated value known only to U (see [CL01]).
3. U(anonymous) ↔ B: run PS.VerifyCredOnNym, where U demonstrates ownership of $\mathsf{cred}_{\mathsf{U}}^{\mathsf{B}}$ and B verifies both, that $\mathsf{cred}_{\mathsf{U}}^{\mathsf{B}}$ and P^{i} are bound to the same ms_{U} (user) and that their owner has registered to the bank with a reg-pseudonym which is bound to the same ms_{U} as P^{i}.
4. U stores in his database the public/secret information related to his account-pseudonym ($\mathsf{pubP}^{\mathsf{i}}$, $\mathsf{secP}^{\mathsf{i}}$). B stores ($\mathsf{pubP}^{\mathsf{i}}$, S, π).

Tax Report Issue. This is a procedure taking place between the owner U of an account α^i, who participates through his pseudonym P^{i} and the bank B. It consists of three stages:

1. *Tax Report Number Acquisition.* The account pseudonym P^{i} collaborates with B in a $\mathsf{BS.Sign}$ procedure, for the former to obtain a (blind towards B) TRN related ticket trtick^i. U deposits *in person* to B the trtick^i to receive a tax-report-number TRN^i. B sends to TA the tuple $(\mathsf{U}, \mathsf{TRN}^i)$ and stores it in its $\mathsf{D_{reg}}$. Tax report numbers are chosen from a range $\mathsf{Range_T} = [\ell_\tau, u_\tau]$.
2. *Tax Report Generation.* The following take place:
   (a) P^{i}: using $\mathsf{secP}^{\mathsf{i}}$ proves that he is the owner of P^{i}, by engaging in the ZKPoK: $PK\{(\beta, \gamma) : (\mathsf{P}^{\mathsf{i}})^2 = (a_\alpha^2)^\beta \cdot (b_\alpha^2)^\gamma\}$ (see [CL01]).
   (b) $\mathsf{P}^{\mathsf{i}} \rightarrow \mathsf{B}$: $C = \mathsf{Com}(\mathsf{ms_U}, \mathsf{TRN}^i, r^i) = k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i}$, where Com is a tax report related commitment scheme, $\mathsf{ms_U}$ U's master-secret, TRN^i, the single-use tax-report-number, which U acquired anonymously, and r^i is a U-generated randomness.
   (c) $\mathsf{P}^{\mathsf{i}} \leftrightarrow \mathsf{B}$: execute the following ZKPoK protocol for P^{i} to show in zero knowledge fashion that C was computed correctly, i.e., that the committed master secret matches the master secret used in the construction of P^{i} $(\mathsf{ms_U})$ and that the exponent of l_α (TRN^i) is among the specified range:
   $PK\{(\gamma, \delta, \varepsilon, \eta) : (\mathsf{P}^{\mathsf{i}})^2 = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \ \wedge \ C^2 = (k_\alpha^2)^\gamma (l_\alpha^2)^\eta (m_\alpha^2)^\varepsilon \ \wedge$
   $\gamma \in \Gamma \ \wedge \ \delta \in \Delta \ \wedge \ \eta \in \mathsf{Range_T}\}$.
   (d) P^{i} *ra* B: a random r_x; if B has received r_x before, the procedure is repeated.
   (e) B: decides x based on r_x. It then computes $h_\alpha^{\mathsf{tax}_i}$ and uses his RSA signature key to *sign* $T^{M,i} = h_\alpha^{\mathsf{tax}_i} \cdot C$ x times into $T^{\sigma,i}$. B provides U with an x-related the secret piece of information $\mathsf{SigInfo} = \mathsf{Enc_{TA}}(x)$, where $x \in \mathsf{Range_x}$. $T^{\sigma,i}$ is then:
   $T^{\sigma,i} = h_\alpha^{d^x \mathsf{tax}_i} \cdot k_\alpha^{d^x \mathsf{ms_U}} \cdot l_\alpha^{d^x \mathsf{TRN}^i} \cdot m_\alpha^{d^x r^i} (modn_\alpha)$.
   (f) $\mathsf{B} \rightarrow \mathsf{U}$: $T^{\sigma,i}$, $\mathsf{SigInfo}$ and $\mathsf{SigInfo}$ re-randomization information $\mathsf{SITranform}$.
3. *Tax Report Transformation.* In this case, after having obtained his signed tax reports, U applies the transformation function F, so that — although provably valid — the modified tax reports are unlinkable to their initial form. In our scheme $F(M)$ is instantiated by adding an extra factor to M. In particular, U:
   (a) transforms both $T^{\sigma,i}$ and $T^{M,i}$ using $F(M, r) = M \cdot s_\alpha^{r_1} \cdot z_\alpha^{r_2}$, where M is the message to be transformed and $r = r_1 || r_2$ is a U-specified randomness. Thus, we get the following for the signed tax report and the corresponding message, respectively,
   $TT^{\sigma,i} \leftarrow F(T^{\sigma,i}, r^{\sigma,i}) \leftarrow h_\alpha^{d^x \mathsf{tax}_i} \cdot k_\alpha^{d^x \mathsf{ms_U}} \cdot l_\alpha^{d^x \mathsf{TRN}^i} \cdot m_\alpha^{d^x r^i} \cdot s_\alpha^{r_1^{\sigma,i}} \cdot z_\alpha^{r_2^{\sigma,i}}$
   $TT^{M,i} \leftarrow F(T^{M,i}, r^{M,i}) \leftarrow h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i} \cdot s_\alpha^{r_1^{M,i}} \cdot z_\alpha^{r_2^{M,i}}$.
   (b) re-randomizes the encryption of $\mathsf{SigInfo}$ according to $\mathsf{SITranform}$

Tax Report Deposit. Each user U(using a real identity) sends to the TA all the tax reports he has acquired, $(TT^{\sigma,1}, TT^{M,1}), \ldots, (TT^{\sigma,N}, TT^{M,N})$, where N is the number of U's accounts. U then proves that each one of these pairs were constructed in a correct way and that they correspond to *his* accounts. The tax report validation consists of two steps:

1. *Signature Validation*, where U shows that $(TT^{\sigma,i}, TT^{M,i})$, for all $i = 1 \ldots N$, correspond to transformations of bank-signatures:
   (a) TA: decrypts $\mathsf{SigInfo}$, reads x and raises all $TT^{\sigma,i}$s to B's signature verification key e, x times using $(modn_\alpha)$:
   $TT^{M',i} \leftarrow (TT^{\sigma,i})^{e^x} \leftarrow h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i} \cdot s_\alpha^{e^x r_1^{\sigma,i}} \cdot z_\alpha^{e^x r_2^{\sigma,i}}$.
   (b) $\mathsf{U} \leftrightarrow \mathsf{TA}$: interact in the following ZKPoK protocol to prove that in each pair, $TT^{M,i}$ and $TT^{M',i}$ correspond to the same $TaxInfo$, i.e., that in both cases the exponents of $h_\alpha, k_\alpha, l_\alpha, m_\alpha$ are the same, or that $\frac{TT^{M,i}}{TT^{M',i}}$ is a factor of powers of s_α and z_α:
   $PK\{(\theta, \eta)\} : (\frac{TT^{M,i}}{TT^{M',i}})^2 = (s_\alpha^2)^\theta (z_\alpha^2)^\eta$.
2. *Tax Report Ownership and non-Repetition Proof.* where U proves to the tax authority TA that each one of the tax reports he deposits had been created through his collaboration with B and that he has not deposit the same tax report twice. The latter is achieved through the one-time-use TRN s. For each one of $TT^{M,i}$s (or $TT^{M',i}$), U reveals the TRN^i to the TA, while he engages to a ZKPoK protocol for the TA to verify that the exponent of k_α in $TT^{M,i}$ (and thus,$TT^{M',i}$) matches the $\mathsf{ms_U}$ used in $\mathsf{P^B}$, i.e.,

$$PK\{(\gamma, \delta, \tau, \varepsilon, \theta, \eta) : (\mathsf{P^{reg}})^2 = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \ \wedge \ \frac{TT^{M,i}}{l_\alpha^{\mathsf{TRN}^i}} =$$

$$= h_\alpha^\tau \cdot k_\alpha^\gamma \cdot m_\alpha^\varepsilon \cdot s_\alpha^\theta \cdot z_\alpha^\eta \ \wedge \gamma \in \Gamma \ \wedge \ \delta \in \Delta\}.$$

Total Tax Calculation. In this operation, TA confirms that U has deposited tax reports for all of his accounts and then uses them to extract the overall tax amount withheld by U's accounts. In particular, TA and U collaborate in an EC.Spend procedure for the latter to spend his unused ecoins from $\mathsf{WAcc_U^B}$ wallet. TA estimates the exact number of U's accounts, computes the overall tax withheld, and based on that the overall balance of U in banks (progressive-tax formulas may then apply):

1. TA: computes the product of all $TT^{M,i}$ $(TT^{M,all})$, which because of the homomorphism of the commitment scheme used, equals to

$$\Pi_{i=1,\ldots,N} TT^{M,i} = \Pi_{i=1,\ldots,N} (h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{trni} \cdot m_\alpha^{r^i} \cdot s_\alpha^{r_1^{M,i}} \cdot z_\alpha^{r_2^{M,i}}) =$$

$$h_\alpha^{TotalTax} \cdot k_\alpha^{N\mathsf{ms_U}} \cdot l_\alpha^{R_t} \cdot m_\alpha^{\sum_{i=1,\ldots,N} r^i} \cdot s_\alpha^{\sum_{i=1,\ldots,N} r_1^{M,i}} \cdot z_\alpha^{\sum_{i=1,\ldots,N} r_2^{M,i}}.$$

2. U reveals $TotalTax = \sum_{i=1\ldots N} \mathsf{tax}_i$, which is the overall tax withheld.

3. U and TA collaborate in a ZKPoK protocol to prove that $\frac{TT^{M,all}}{h_\alpha^{TotalTax} \cdot l_\alpha^{R_t}}$ is correctly created and thus prove that $TotalTax$ is the required amount (note that TA knows R_t):
$PK\{(\beta, \gamma, \delta, \zeta, \eta) : (\mathsf{P}^{\mathsf{reg}})^2 = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \wedge$
$\frac{TT^{M,all}}{h_\alpha^{TotalTax} l_\alpha^{R_t}} = (k_\alpha^N)^\gamma \cdot m_\alpha^\varepsilon \cdot s_\alpha^\zeta \cdot z_\alpha^\eta \ \wedge \ \gamma \in \Gamma \ \wedge \ \delta \in \Delta\}$.

4 Discussion

The following theorem states the correctness, privacy and security of our general scheme: if the underlying primitives (anonymous credential system, e-cash system, blind signatures, commitments and ZKPoK) are secure, then our scheme satisfies *correctness*, *account–account unlinkability*, *account–account-owner unlinkability*, *fairness in tax reporting*, *tax report non transferabiliy* and *unforgeability*, and *accountability*. We use prove this theorem with the following lemmas. We have omitted the correctness definition and solution for space reasons (see appendix A, [AVB10]).

Lemma 2. If the underlying primitives (anonymous credential system, ecash system, and ZKPoK) are secure, then our scheme satisfies *account-account unlinkability*.

Proof. Account-account unlinkability is maintained in the Account Open procedure through the *unlinkability* property of the ecash scheme used for perm_α and the *unlinkability of pseudonyms* property of the underlying anonymous credential system. Account-account unlinkability is also maintained through the tax reporting: Let α^1 and α^2 two accounts of U for which he obtains tax reports T^1, T^2 respectively. Then T^1 and T^2 are unlinkable one to the other because of the *hiding* property of the commitment scheme used to generate them and the *zero knowledge* property of the ZKPoK scheme used to prove their correct construction.

Lemma 3. If the underlying primitives (anonymous credential system, ecash system, blind signatures, commitment and ZKPoK, transformation function F, Paillier cryptosystem) are secure, then our scheme satisfies *account–account-owner unlinkability*.

Proof. Let α^i an anonymous account of user U managed by pseudonym P^i. Let T and TT be the tax report for α^i and its transformed version. Unlinkability of α^i and U at the AccountOpen procedure is achieved through the *anonymity* property of the ecash scheme realizing perm_αs and of and pseudonym system used for the generation of P^i as well as through the *blindness* of the blind signature scheme used for the acquisition of TRNs. T is unlinkable to U through the *hiding* property of the commitment scheme, which "hides" the ms_U committed in T and the security (*zero knowledge*) of the ZKPoK protocol used to validate the construction of T: no information is leaked neither TRN nor for ms_U contained

in T. TT on the other hand, does not reveal anything regarding T or the account because of the *hiding* property of transformation function F (see appendix for more details) used for its construction, the zero knowledge property of the ZKPoK protocol used at its validation and the re-randomization property of the Paillier cryptosystem used for blinding SigInfo.

Lemma 4. If the underlying primitives (anonymous credential system, digital signatures, commitment) are secure, then our scheme satisfies *Tax Report Unforgeability.*

Proof. Let that user U manages an account α_{U} through a pseudonym P and generates tax report $T^{\sigma/M}$, which is later transformed to $TT^{\sigma/M}$, through $F()$. We need to prove that the tax report remains unforgeable at all stages. T is an RSA-signature-based function on a commitment on TRN, tax_i and ms_{U}. To avoid B-signature forgeries exploiting RSA homomorphism, apply the signature scheme on T^M x number of times, while the RSA-signature verification key and x are kept secret to U. x is only revealed to TA only at the TT deposit procedure through SigInfo. We assume that there are very few x-es w.r.t. the total number of tax reports so that x-based linkability attacks do not apply. U has no incentive to alter SigInfo. To avoid such a forgery using the same tax report, we make use of TRN s, B-chosen numbers of a pre-specified range such that summations of two numbers in $\mathsf{Range}_{\mathsf{T}}$ result in an invalid number. *bindness* property of the commitment scheme used in T generation guarantees that as long as the RSA signature is unforgeable, U cannot dispute the "TaxInfo" he has committed to in T^M.

Lemma 5. If the underlying primitives (anonymous credential system, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Tax Report non transferability.*

Proof. In our system users are highly motivated not to share their ms_{U}. Thus, assuming that they are not doing so, Tax-Report non transferability is achieved through the need to prove knowledge of the ms_{U} at each step of the tax reporting. More specifically, account pseudonyms are required to show that their ms_{U} matches the one committed in T, which is then signed and -thus- cannot change (unforgeability of the signature scheme). The *proof of knowledge* property of the ZKPoK scheme used when depositing TT, guarantees that user depositing TT knows the corresponding ms_{U}, which should match the ms_{U} used in all tax reports deposited by the same user, as well as his registration pseudonym.

Lemma 6. If the underlying primitives (anonymous credential system, ecash, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Fairness.*

Proof. Because of Tax Report Unforgeability and non-transferability, users cannot change the tax reported in each report or use other users' tax reports. Because of the *Identification of Violators* and *Violators' Traceability* property of ecash implementing perm_{α}s, users cannot lie to the bank regarding the number of the accounts they have opened: if they try to prove they opened fewer

accounts, some of the perm_αs in $\mathsf{WAcc}^{\mathsf{B}}$ will be doublespent. At the same time, because of the TRNs, users cannot avoid a tax report, by depositing another one twice.

Lemma 7. If the underlying primitives (anonymous credential system, ecash, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Accountability.*

Proof. Because of the Identification of Violators and Violators' Traceability property of ecash implementing perm_αs, users who lie regarding the anonymous accounts they opened are identified. Because of the *proof of knowledge* property of the ZKPoK protocols, the *non-transferability of credentials* property of the underlying pseudonym system and the *non-transferability* property of tax reports, users trying to use other users' tax reports are detected.

5 Related Work

Legal issues related to anonymous payments, including purchase receipts have been technically addressed in [PWP00, BP89]. [JY96], [LMP96] are cases of protocols providing conditionally anonymous payments from user issued bank accounts. However, their work is different from ours as there is either a third trusted party involved for anonymity revocation purposes, or they do not offer privacy against coalitions of banks. In [AB09], the authors provide privacy in the management of anonymous accounts, even w.r.t. the bank through the use of anonymous credit cards. However, we take an additional step in addressing tax reporting for bank accounts, which is not an issue in credit-only systems.

Taxation has been addressed in the past in the stock market. In [XYZ00], the authors propose a scheme addressing a similar problem to ours: anonymous and taxable stock market trading accounts. As in our system, users are using a generated anonymous credential from a public credential to validate anonymous stock-transaction. However, their system differs from our own in two major ways. First, they only allow for each user to own one anonymous account, because of the extra complications to tax reporting the multiple accounts would cause. Addressing these complications is one of our major contributions. Secondly, they do not aim to prevent the Tax Authority from learning which accounts the reports are coming from. Thus if the TA were to collaborate with the Stock Exchange Center, they could re-link the users with their anonymous accounts. Preventing this is another contribution of our system.

6 Conclusion

In this paper we presented a privacy preserving bank account system, where individuals may open arbitarily anonymous and unlinkable accounts w.r.t. the bank and tax authority collaborations. All accounts are ultimately and in zero knowledge fashion connected to their owner. We emphasize on the bank account taxation mechanism, where individual users report the aggregated amount of tax all of their accounts have been withheld in a fair and accountable way.

References

[AB09] Androulaki, E., Bellovin, S.: An anonymous credit card system. In: Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G. (eds.) TrustBus 2009: Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business. LNCS, vol. 5695, pp. 42–51. Springer, Heidelberg (2009)

[AVB10] Androulaki, E., Vo, B., Bellovin, S.M.: Taxable, privacy-preserving bank accounts. Technical Report CUCS-005-10, Computer Science Dept., Columbia University (2010), http://www.cs.columbia.edu/research/publications

[B00] Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)

[BCDG88] Brickell, E.F., Chaum, D., Damgård, I., Graaf, J.v.d.: Gradual and verifiable release of a secret. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 156–166. Springer, Heidelberg (1988)

[BP89] Brk, H., Pfitzmann, A.: Digital payment systems enabling security and unobservability. Computers & Security 8 (1989)

[C91] Chaum, D.: Zero-knowledge undeniable signatures. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 458–464. Springer, Heidelberg (1991)

[CEVDG88] Chaum, D., Evertse, J.-H., van de Graaf, J.: An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 127–141. Springer, Heidelberg (1988)

[CHL05] Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005)

[CL01] Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)

[CM99] Camenisch, J., Michels, M.: Separability and efficiency for generic group signature schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 413–430. Springer, Heidelberg (1999)

[CP93] Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)

[CS97] Camenisch, J., Stadler, M.: Effcient group signature schemes for large groups. In: Sommer, G., Daniilidis, K., Pauli, J. (eds.) CAIP 1997. LNCS, vol. 1296, pp. 410–424. Springer, Heidelberg (1997)

[JY96] Jakobsson, M., Yung, M.: Revokable and versatile electronic money (extended abstract). In: CCS 1996: Proceedings of the 3rd ACM Conference on Computer and Communications Security, pp. 76–87. ACM, New York (1996)

[LMP96] Low, S., Maxemchuk, N.F., Paul, S.: Anonymous credit cards and its collusion analysis. IEEE Transactions on Networking (December 1996)

[LRSW99] Lysyanskaya, A., Rivest, R., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H.M., Adams, C.M. (eds.) SAC 1999. LNCS, vol. 1758, pp. 184–199. Springer, Heidelberg (2000)

[PP99] Paillier, P., Pointcheval, D.: Efficient public-key cryptosystems provably secure against active adversaries. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 165–179. Springer, Heidelberg (1999)

[PWP00] Pfitzmann, B., Waidner, M., Pfitzmann, A.: Secure and anonymous electronic commerce: Providing legal certainty in open digital systems without compromising anonymity. Technical Report 93278, IBM Research RZ3232 (2000)

[XYZ00] Xu, S., Yung, M., Zhang, G.: Scalable, tax evasion-free anonymous investing (2000)

A Complementary Security Proofs - Primitives Instantiation

A.1 Correctness Analysis

Definition. *Correctness* requires that if an honest user U, who is eligible for opening anonymous accounts with an honest bank B, runs AccountOpen with B, then none will output an error message. Also, if honest U, has opened accounts $\alpha_1, \ldots, \alpha_N$ with honest B, and runs TaxReportIssue, then no one will output an error message, while when the user tries to deposit them and thus runs with TA TaxReportDeposit and TotalTaxCalculation no entity will output error message and they will output the aggregated tax withheld by honest U's accounts.

Lemma 1. If the underlying primitives (anonymous credential system, e-cash system, commitments and ZKPoK) are secure, then our scheme satisfies *Correctness.*

Proof. The first condition of correctness is satisfied directly through the *correctness* of the underlying schemes of ecash and anonymous credentials and according to which if U is honest neither EC.Spend procedure of perm_α nor PS.VerifyCredOnNym (which take place at the Account Open will output an error message. The correctness and verifiability of the RSA signature scheme, its homomorphism and the correctness of the used ZKPoK protocols used to confirm that U is the owner of all tax reports and guarantee that TaxReportDeposit will not output an error message.

A.2 Security of Auxiliary Functions

Lemma 8. The transformation function F, defined on $D_M x \mathbb{Z}$, where:

- $F(M, r) = M \cdot s_\alpha^{r_1} \cdot z_\alpha^{r_2} (mod n_\alpha)$, $n_\alpha = p_\alpha \cdot q_\alpha$, p_α, q_α safe primes, $s_\alpha, z_\alpha \in QRn_\alpha$, $r = r_1 || r_2$ is a random number and M the message to be blinded;

– $D_M = \{x : \exists y, z, w, j : x = h_\alpha^y \cdot k_\alpha^z \cdot l_\alpha^w \cdot m_\alpha^j (mod n_\alpha)\}$, where $h_\alpha, k_\alpha, l_\alpha, m_\alpha \in QR_{n_\alpha}$ are system parameters;

is computationally non-invertible and provides output indistinguishability w.r.t. M-inputs. More specifically, we claim that F supports:

– *Non Invertibility* Given an output f of $F()$ it is computationally impossible to compute $M \in D_M$ and r such that $F(\mathrm{M,r}) = \mathrm{f}$.
– *Input-Output Unlinkability* Given two messages M_1 and M_2 and an output f of $F()$ which corresponds to one of the messages, it is computationally hard to decide which message corresponds to f with a better probability than $1/2$.

Proof. Both properties derive directly from the discrete log assumption modulo a safe prime product and strong RSA assumption.

Lemma 9. The function Com used, defined on $(\mathbb{Z}x\mathbb{Z})x\mathbb{Z}$, where

$$\mathsf{Com}(x, y; r) = k_\alpha^x \cdot l_\alpha^y \cdot m_\alpha^r (mod n_\alpha),$$

$n_\alpha = p_\alpha \cdot q_\alpha$, $p_\alpha,\ q_\alpha$ safe primes, $k_\alpha, l_\alpha, m_\alpha \in QR_{n_\alpha}$ is a commitment scheme on x, y with randomness r.

Proof. Function Com satisfies both properties *bindness* and *hiding* which derives from the discrete log assumption modulo a product of safe primes and factoring assumption.

A.3 Paillier Encryption

The Paillier cryptosystem is s a probabilistic asymmetric algorithm for public key cryptography and bases its security on the decisional composite residuosity assumption (see [PP99] for details). Assuming the system is meant for a user U to be able to receive messages confidentially, the operations supported are as in every cryptosystem the following:

– $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{Pail.KeyGen}(1^k)$, where U generates his encryption key pair. In particular, U chooses two safe large prime numbers p and q, such that $gcd(p-1, q-1) = 2$, computes $n = pq$ and chooses $g \in \mathbb{Z}_{n^2}^*$, such that n divides the order of g. $\mathsf{pk_U} = (n, g), \mathsf{sk_U} = (p, q)$.
– $\langle\ \mathsf{C}/\perp\ \rangle \leftarrow \mathsf{Pail.Encrypt}(\mathsf{pk_U}, m)$, where anyone may use $\mathsf{pk_U}$ to generate ciphertext C on a mesage m: $\mathsf{C} = g^m \cdot r^n (mod n^2)$, where r is randomly chosen.
– $\langle\ m/\perp\ \rangle \leftarrow \mathsf{Pail.Decrypt}(\mathsf{sk_U}, \mathsf{C})$, where U uses his secret key to receive the plaintext.

It is apparent that a particular plaintext may have many ciphertexts, depending on r. We make use of this property in the encryption of x in two ways: (a) two users will not be able to distinguish whether they have the same x or not,

and are thus unable to know whether they are able to exploit RSA homomorphism; (b) for re-randomization of SigInfo: users who know n can simply compute $\mathsf{C} \cdot (r')^n (mod n^2)$ and generate another ciphertext of x unlinkable to C. Thus in this case of encryption algorithm, SITranform is n.

Security Properties: Semantic security against chosen plaintext attacks (IND-CPA), i.e. given pk, two messages m_1, m_2 and a ciphertext corresponding c to one of them, it is impossible to guess which of the messages corresponds to c with a better probability than 1/2.

Formal Analysis of Privacy for Vehicular Mix-Zones

Morten Dahl[1,2], Stéphanie Delaune[2], and Graham Steel[2]

[1] Department of Computer Science, Aalborg University
[2] LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

Abstract. Safety critical applications for recently proposed vehicle to vehicle ad-hoc networks (VANETs) rely on a beacon signal, which poses a threat to privacy since it could allow a vehicle to be tracked. Mix-zones, where vehicles encrypt their transmissions and then change their identifiers, have been proposed as a solution to this problem.

In this work, we describe a formal analysis of mix-zones. We model a mix-zone and propose a formal definition of privacy for such a zone. We give a set of necessary conditions for any mix-zone protocol to preserve privacy. We analyse, using the tool ProVerif, a particular proposal for key distribution in mix-zones, the CMIX protocol. We show that in many scenarios it does not preserve privacy, and we propose a fix.

Keywords: Privacy, VANETs, Mix-Zones, Security Protocols.

1 Introduction

Road traffic accidents are the most common cause of death in young adults in industrialized countries [13]. To improve road safety, a vehicle-to-vehicle communication platform is currently being developed by consortia of car manufacturers and legislators [15,17]. Safety-related applications such as collision warning systems and high speed toll payment are envisaged. Dubbed vehicular ad-hoc networks (VANETs), the platform is based on decentralised mobile ad-hoc networks in order to retain scalability despite the high average speed of vehicles, and the large size of the network. As a consequence, the protocols used within the network are designed to use few steps, short messages, and not rely heavily on infrastructure for e.g. obtaining trust. To facilitate safety-critical applications there is a consensus that all vehicles must periodically broadcast a beacon message consisting of the vehicle's location (in the form of a GNSS coordinate), velocity, and identifier. Broadcasting this data several times per second raises privacy issues.

Fortunately, many of the envisioned applications, including collision avoidance, do not need a real-world identifier such as the vehicle's license plate, but can instead make do with a random identifier known as a pseudonym. However, long term tracking may still reveal the real-world identity of the driver. One can change pseudonym from time to time, but for this to have any effect the vehicles must change pseudonyms under the right circumstances. It seems preferable to

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 55–70, 2010.

change pseudonyms e.g. at intersections where several vehicles are close together and their paths unpredictable. This mimics the ubiquitous computing idea of a *mix-zone*, where beacon signals are turned off in a mixing area [3]. Vehicles cannot turn off beacon messages since many accidents happen at intersections, hence the idea is to have all vehicles encrypt their beacon signals when inside the zone [11].

Related Work. Several papers discuss the background to the VANET privacy problem and the merits of the pseudonymous authentication solution [10,12,14]. Previous analysis work aims to evaluate the effectiveness of (a larger network of) general mix-zones in terms of the probability of the attacker correctly linking two pseudonyms based on assumed prior known statistics about vehicles movement [6], when the effectiveness of each single mix-zone is already assumed. Privacy for mobile devices with RFID tags has recently been treated formally [2,5,18] . It is not clear how the definitions of privacy in these papers relate to each other, and even less so to our own definition. We, for instance, have to exclude scenarios where privacy is broken independently of the key establishment protocol and must moreover require synchronised behaviour of vehicles. These requirements for obtaining privacy are closer to the requirements made for electronic voting protocols [8].

Our contributions. In this paper, we investigate formally the effectiveness of vehicular mix-zone proposals. We model the network traffic inside a mix-zone, and examine under which conditions it is reasonable to expect any gain in privacy. We use the formal notion of indistinguishability to formalise the privacy property for a mix-zone. We analyse the CMIX protocol [11] that has been proposed to distribute keys to vehicles entering the mix-zone. We report some scenarios in which the use of the CMIX protocol can prevent privacy from being achieved. These scenarios have been discovered with the aid of the protocol analysis tool ProVerif [4]. We propose a fix to the protocol. We believe this is the first work to investigate the privacy property of an encrypted mix-zone, in particular when the key distribution protocol is also taken into account.

Paper outline. In the next section, we present the concept of a mix-zone and we give a description of the CMIX protocol. Then, we give our formal model (see Section 3) and we explain our formal definition of mix-zone privacy, which corresponds to an indistinguishability property (Section 4). In Section 5, we give our results obtained on mix-zones, first assuming an ideal key distribution protocol, and then using the CMIX protocol. Finally, we evaluate the protocol and our modelling approach, we propose a fix, and we give conclusions.

2 Mix-Zones and CMIX Protocol

This section describes mix-zones, and in particular the CMIX protocol used to distribute keys to vehicles entering a zone.

2.1 Mix-Zones

As discussed in the previous section, mix-zones are needed for the change of pseudonyms to have any effect in preserving privacy. However, changing pseudonyms while close to other vehicles is not sufficient to guarantee 'unlinkability', which we define informally as the property that an attacker cannot know that the old and new pseudonym belong to the same vehicle. To obtain this, pseudonyms must also be changed synchronously from the point of view of the attacker. More precisely, by synchronously, we mean that once one vehicle has started broadcasting using a new pseudonym then all future broadcasts heard by the attacker from at least one other vehicle must be using a new pseudonym as well.

If two vehicles in a mix-zone can agree on a precise point in time to change their pseudonyms (for instance by one of the vehicles broadcasting the time of when it is going to change its pseudonym) then synchronised change of pseudonym is sufficient for unlinkability. In practice however, for several reasons, one might want to allow a larger time interval for pseudonym change, e.g. to have a better chance that another vehicle is nearby to synchronise with, to ensure a certain level of unpredictability of trajectory, to account for clock differences, etc. Using a longer time interval has the undesirable effect of causing a radio-silence period during which none of the safety beacon messages can be broadcast. Encrypted mix-zones are suggested to remedy these short-comings: beacon messages can still be broadcast during the synchronisation time interval as long as the attacker cannot read them. A recent proposal argues that turing off radios at intersctions might be a worthwhile trade-off for privacy [7], but in this paper we concentrate on investigating what privacy can be achieved when beacon signals are required to be left on.

2.2 The CMIX Protocol

The CMIX protocol [11] distributes keys for encrypting beacon messages while in the mix-zone with the goal of preventing an attacker from linking the pseudonym of an in-coming vehicle with the pseudonym it uses when leaving. Every vehicle is equipped with a tamper-resistant device (TRD) allowing access to its contents only through its API. An offline Certification Authority (CA) run by a trusted third party is responsible for issuing certificates cryptographically binding a pseudonym P together with the public part ($\mathsf{pub}(K)$) of an asymmetric key K. Every vehicle has a fresh non-empty set of these key-pseudonym pairs stored in its TRD. One pair is marked as current, to be used when sending messages.

Vehicles entering the mix-zone (part A of Figure 1) are alerted of the presence of a road-side unit (RSU) by a radio broadcast. This triggers the vehicles to initiate a key establishment session.

1. $V \rightarrow RSU : \mathsf{sign}_{K_V}(\mathsf{request}, T_s),\ \mathsf{sign}_{K_{CA}}\big(P_V, \mathsf{pub}(K_V)\big)$
2. $RSU \rightarrow V : \mathsf{aenc}_{\mathsf{pub}(K_V)}\big(\mathsf{sign}_{K_{RSU}}(P_V, zk, T_s)\big),\ \mathsf{sign}_{K_{CA}}\big(P_{RSU}, \mathsf{pub}(K_{RSU})\big)$
3. $V \rightarrow RSU : \mathsf{sign}_{K_V}(\mathsf{ack}, T_s),\ \mathsf{sign}_{K_{CA}}\big(P_V, \mathsf{pub}(K_V)\big)$

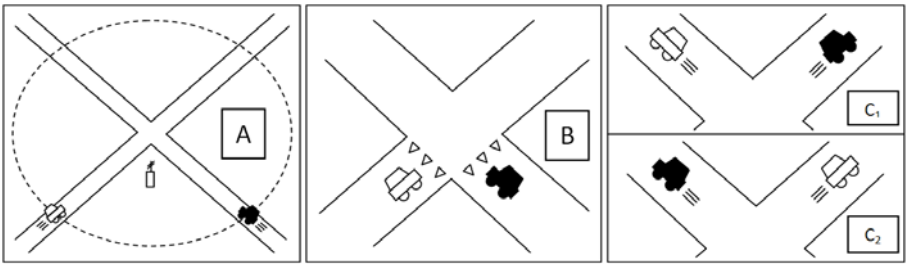


Fig. 1. Intended usage of encrypted mix-zones

The first message is a signed timestamp T_s together with the constant request used as a tag. The reply made by the RSU contains the zone encryption key zk encrypted under the public key $\mathsf{pub}(K_V)$ associated with the vehicle's current pseudonym P_V. The corresponding private key is assumed to be only known by the vehicle's on board tamper-resistant cryptographic device, which can decrypt the packet, store the zone key, and make available an encryption and decryption service using this key. In this way, the zone key remains unknown to everyone, including an attacker with a vehicle and a tamper-resistant device of his own. The last message is an acknowledgement sent by the vehicle. Every message is appended with the principal's current certificate.

The zone key is then used to encrypt and decrypt beacon messages while inside the geographical area dictated by the RSU. During their journey through the mix-zone, the vehicles will come in close enough proximity that the attacker is assumed unable to distinguish their locations (part B of Figure 1). Before leaving the mix-zone the vehicles change their pseudonyms leaving the attacker unable to determine if they leave according to part C_1 or part C_2 of Figure 1.

In the CMIX proposal [11], it is not specified whether a deterministic or probabilistic encryption scheme is used to encrypt beacon messages. Probabilistic encryption might seem the best solution, but due to the tight size constraints of messages in VANETs, it may be preferable to use a deterministic scheme. Deterministic schemes might still prevent the easy comparison of ciphertexts due to the rapidly changing content of beacon messages (such as the coordinate). Since this would depend on the exact cipher mode, beacon message format, etc, and this is not yet fixed [17], we consider both types of encryption scheme in our analysis.

A short informal analysis of the CMIX protocol is provided by Freudiger et al. [11]. The threat scenario they consider is unclear: they first state that their adversary is a passive outsider [11, §2.2] but then describe the resistance of the protocol to attacks where the adversary sends messages to try to impersonate an RSU [11, §3.2]. In general, VANET protocols are assumed to be required to withstand attack by active adversaries, as described e.g. by Raya and Hubaux (a subset of the authors of the CMIX paper) [14]. In this paper, therefore, we consider both the passive attacker and an active attacker that can forge and broadcast messages, but not prevent messages from being received.

We will explain during our analysis under what assumptions particular attacks would be effective. Note that our adversary is assumed to have no visual contact as he would otherwise be able to track a vehicle using e.g. the license plate.

3 Formal Modelling

The process calculus of Blanchet *et al.* [4] used by the ProVerif tool is a variant of the applied pi calculus [1], a process calculus for formally modelling concurrent systems and their interactions. We recall the basic ideas and concepts of this calculus that are needed for our analysis.

3.1 Messages

To describe messages, we start with a set of *names* (which are used to name communication channels and other atomic data), a set of *variables*, $x, y, \ldots$ and a signature Σ formed by a finite set of *function symbols* each with an associated arity. Function symbols are distinguished by two categories: *constructors* and *destructors*. We use standard notation for function application, i.e. $f(M_1, \ldots, M_n)$. Constructors are used for building messages. Destructors represent primitives for taking messages apart and can visibly succeed or fail (while constructors always succeed). Messages $M, N, \ldots$ are obtained by repeated application of constructors on names and variables whereas a term evaluation D can also use destructors. The semantics of a destructor g of arity n is given by a set of rewrite rules of the form $g(M_1, \ldots, M_n) \to M_0$ where $M_0, \ldots, M_n$ are messages that only contains constructors and variables. Given a term evaluation D, we write $D \Downarrow M$ when D can be reduced to M by applying some destructor rules.

In the following, we consider constructors to model signatures and different kinds of encryptions (symmetric/asymmetric and deterministic/probabilistic). The symbol pub is a constructor representing the public key associated to the private key given in argument. The semantics of our destructors are given below:

$$\mathsf{checksign}(\mathsf{sign}(x, y), \mathsf{pub}(y)) \to x \qquad \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x$$
$$\mathsf{getmsg}(\mathsf{sign}(x, y)) \to x \qquad \mathsf{rsdec}(\mathsf{rsenc}(x, y, z), y) \to x$$
$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), y) \to x$$

We model probabilistic encryption by $\mathsf{rsenc}(m, k, r)$ where the r component is fresh for every encryption, thus preventing comparison. We model a signature scheme by two rewrite rules: the first one is used to verify a signature and the second one models the fact that the signature scheme is not message concealing.

3.2 Processes

Processes are built from the grammar described below, where N is a message, D is a term evaluation, a is a name, c is a channel name, and x a variable.

$P, Q, R ::=$	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
new $a; P$	name restriction
let $N = D$ in P else Q	term evaluation
$\mathrm{in}(c, N); P$	message input
$\mathrm{out}(c, N); P$	message output

The process "let $N = D$ in P else Q" tries to evaluate D; if this succeeds and if the resulting message matches the term N then the variables in N are bound and P is executed; if not then Q is executed. The rest of the syntax is quite standard. To ease the presentation, we will use tuples of messages, denoted by parentheses, while keeping the reduction rules for these tuples implicit. We will omit "else Q" when the process Q is 0.

An *evaluation context* is a context, that is a process with a hole, built from $[\_]$, $C \mid P$, $P \mid C$ and new $a; C$. We obtain $C[P]$ as the result of filling $C[\_]$'s hole with P. A process P is *closed* if all its variables are bound through an input or a let construction.

The RSU process. To illustrate the calculus used throughout this paper, we give below a description of the RSU part of the CMIX protocol. We follow the description given in the previous section. The RSU sends and receives all messages using some public channel c and holds a freshly generated zone key zk. We also model its pseudonym p_{rsu} and its private key k_{rsu} by fresh names. We assume that the RSU already knows its certificate $\mathsf{sign}((p_{rsu}, \mathsf{pub}(k_{rsu})), k_{ca})$. Below, we only model the reception of the first message with its decomposition. After some checks, the reply to the vehicle containing zk is constructed and sent. We do not model the reception of the acknowledgement.

$$
\begin{aligned}
\mathsf{RSU}_{\mathsf{CMIX}} \stackrel{\mathsf{def}}{=}\ & \mathrm{in}(c, (x^s, x^c));\\
& \text{let } (x_{pv}, x_{pkv}) = \mathsf{checksign}(x^c, \mathsf{pub}(k_{ca})) \text{ in}\\
& \text{let } (\mathsf{request}, x_T) = \mathsf{checksign}(x^s, x_{pkv}) \text{ in}\\
& \text{let } y^s = \mathsf{sign}((x_{pv}, zk, x_T), k_{rsu}) \text{ in}\\
& \text{let } y^c = \mathsf{sign}((p_{rsu}, \mathsf{pub}(k_{rsu})), k_{ca}) \text{ in}\\
& \mathrm{out}\ (c, (\mathsf{aenc}(y^s, x_{pkv}), y^c)); \ldots
\end{aligned}
$$

The operational semantics of processes in the calculus of ProVerif , are essentially defined by two relations, namely *structural equivalence* $\equiv$ and *reduction* $\rightarrow$. We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. *Structural equivalence* is the smallest equivalence relation on processes that is closed under application of evaluation contexts and some other standard rules such as associativity and commutativity of the parallel operator. *Reduction* is the smallest relation closed under structural equivalence and application of evaluation contexts such that:

RED I/O	$\text{out}(c, M).Q \mid \text{in}(c, N).P \rightarrow Q \mid P\sigma$	
RED FUN 1	let $N = D$ in P else $Q \rightarrow P\sigma$	if $D \Downarrow M$
RED FUN 2	let $N = D$ in P else $Q \rightarrow Q$	if there is no M such that $D \Downarrow M$
REPL	$!P \rightarrow P \mid !P$	

where σ is the substitution defined on the variables that occur in N and such that $M = N\sigma$. In case such a substitution does not exist, the resulting process will be $Q \mid \text{in}(c, N).P$ for RED I/O rule and Q for the RED FUN 1 rule.

3.3 Observational Equivalence

The notion of observational equivalence was introduced in [1]. We write $P{\downarrow_c}$ when P emits a message on the channel c, that is, when $P \equiv C[\text{out}(c, M); Q]$ for some evaluation context C that does not bind c and some process Q.

Definition 1. Observational equivalence $\sim$ *is the largest symmetric relation* $\mathcal{R}$ *on closed processes such that* $P\ \mathcal{R}\ Q$ *implies:*

1. *if* $P{\downarrow_c}$ *then* $Q{\downarrow_c}$*;*
2. *if* $P \rightarrow P'$ *then there exists* Q' *such that* $Q \rightarrow^* Q'$ *and* $P'\ \mathcal{R}\ Q'$*;*
3. $C[P]\ \mathcal{R}\ C[Q]$ *for all evaluation contexts* C*.*

Intuitively, a context may represent an attacker, and two processes are observationally equivalent if they cannot be distinguished by any attacker. Note that such an attacker is too powerful for our purpose since the nature of broadcast communication does not allow him to block all messages. When performing the analysis we will exclude attacks that are not possible for our attacker; as we will see, the attacks we find do not rely on the attacker blocking messages.

ProVerif is not able to check observational equivalence directly but actually checks a stronger notion that implies observational equivalence [4]. However, this notion is too strong in many situations. This problem has recently been studied in and a method has been proposed to extend the class of equivalences which ProVerif is able to verify [9]. We will use this method to overcome the limitations of ProVerif and to automatically verify the equivalences allowing us to model our privacy property.

4 Privacy for Vehicular Mix-Zones

In this section we show how the privacy property informally described in Section 2 can be formalised in our setting. We build on the classical approach of formalising privacy properties as some kind of observational equivalence in a process algebra or calculus [8,16], and extend this to take into consideration mix-zones and vehicle mobility.

4.1 Mix-Zones

In the previous sections we have informally used the term mix-zone to describe a place suitable for vehicles to change their pseudonym by being able to mix

or hide among each other. We formally define a mix-zone as consisting of five locations $entry_L$, $entry_R$, *proximity*, $exit_L$, and $exit_R$. We use public channels to model these locations. If two messages are emitted on different channels, then our attacker will be able to see a difference. This corresponds to the fact that he is able to tell that they were transmitted from geographically different locations. Note that messages sent on a public channel can be received on another public channel with the help of our active attacker. Vehicles enter the mix-zone by one of the entry locations and exit by one of the exit locations. The *proximity* location models a stretch within the mix-zone where vehicles are so close to each other that our attacker cannot tell them apart geographically.

Beacon messages are defined as consisting only of a pseudonym p_v modelled by a fresh name. This pseudonym is signed using the vehicle's current key k_v and appended with the CA signed certificate binding the pseudonym together with the public part of k_v. Formally a beacon message is defined as $\Big(\mathsf{sign}(p_v, k_v),\ \mathsf{sign}((p_v, \mathsf{pub}(k_v)), k_{ca})\Big)$ where k_{ca} is the private key of the CA. Note that all the location data in beacon messages are modelled by the channel on which they are sent.

4.2 Privacy

The formal privacy property aims to capture the fact that an attacker cannot track a vehicle. We assume that the attacker can listen on the entire network and hence on all public channels. Thus, in order to achieve privacy, we need to suppose the presence of at least two vehicles.

We consider a single mix-zone with two vehicles V_A and V_B, as in Figure 1. V_A will always start in $entry_L$ and V_B always in $entry_R$. Going through the mix-zone, each vehicle emits a series of beacon messages. They can do this in two different ways:

1. The vehicle V_A moves from $entry_L$ to *proximity* to $exit_L$ while V_B moves from $entry_R$ to *proximity* to $exit_R$ (as in part C_1 of Figure 1).
2. The vehicle V_A moves from $entry_L$ to *proximity* to $exit_R$ while V_B moves from $entry_R$ to *proximity* to $exit_L$ (as in part C_2 of Figure 1).

Intuitively, we achieve privacy if an attacker cannot tell the two cases apart. Formally, let $\mathsf{V}(entry, exit)$ stand for the vehicle that moves from *entry* to *proximity* to *exit*. Privacy holds if the following equivalence holds:

$$C\Big[\mathsf{V}(entry_L, exit_L) \mid \mathsf{V}(entry_R, exit_R)\Big] \sim C\Big[\mathsf{V}(entry_L, exit_R) \mid \mathsf{V}(entry_R, exit_L)\Big].$$

The next section presents the analysis we have performed, including the definition of the vehicles processes, and also the C contexts with which the analysis has been performed.

5 Privacy Analysis

The analysis is performed in two models: an ideal model where the vehicles are assumed to know the mix-zone encryption key and a CMIX model where

this key is distributed using the CMIX protocol. From our ideal model analysis, we extract a set of scenarios where it is possible for a 'perfect' key distribution protocol to guarantee privacy. We then evaluate the CMIX protocol with respect to these scenarios.

5.1 Privacy in the Ideal Model

In the ideal model the vehicles magically know the mix-zone encryption key, the attacker does not know it, and the only communications are the beacon messages. As discussed in previous sections, we consider both deterministic and probabilistic encryption of beacon messages.

Experimental Analysis. We model each vehicle using a fixed sequence of beacon message emissions $\overline{p_v^1}$; $\{\overline{p_v^1}\}_{zk}$; $\{\overline{p_v^2}\}_{zk}$; $\overline{p_v^2}$ where:

- $\overline{p_v^i} \stackrel{\mathsf{def}}{=} \mathsf{sign}(p_v^i, k_v^i)$, $\mathsf{sign}((p_v^i, \mathsf{pub}(k_v^i)), k_{ca})$, and
- $\{\overline{p_v^i}\}_{zk} \stackrel{\mathsf{def}}{=} \mathsf{senc}(\overline{p_v^i}, zk)$ or $\mathsf{rsenc}(\overline{p_v^i}, zk, r)$ depending on whether we are considering respectively deterministic or probabilistic encryption. In this last case, each occurrence of r represents a fresh nonce.

From this fixed sequence we generate a set of relevant scenarios by adding two changes of location, from *entry* to *proximity* and from *proximity* to exit, and we perform a geographical synchronisation either coming into or going out of the *proximity* location. We allow each vehicle to emit each beacon three times, so it is possible to change locations at any position in the sequence. The first $\overline{p_v^1}$ is always emitted at an *entry* location and the last $\overline{p_v^2}$ is always emitted at an *exit* location. We then investigate whether we can prove privacy if two vehicles in the mix-zone conform to this pattern.

We write each scenario as a process. For instance, the scenario where all $\overline{p_v^1}$ beacon messages are emitted at the *entry* location, the $\{\overline{p_v^1}\}_{zk}$ spread out over *entry* and *proximity*, the $\{\overline{p_v^2}\}_{zk}$ over *proximity* and *exit*, and the $\overline{p_v^2}$ at *exit* with deterministic encryption and synchronisation before leaving the *proximity* location, is represented by:

$$
\begin{aligned}
\mathsf{Vehicle}(\mathit{entry}, \mathit{exit}) \stackrel{\mathsf{def}}{=} & \text{ new } p_v^1;\ \text{new } k_v^1;\ \text{new } p_v^2;\ \text{new } k_v^2;\ \text{out}(\mathit{entry}, \overline{p_v^1});\\
& \mathit{(*\ key\ establishment\ *)}\\
& \text{out}(\mathit{entry}, \{\overline{p_v^1}\}_{zk});\\
& \text{out}(\mathit{proximity}, \{\overline{p_v^1}\}_{zk});\ \text{out}(\mathit{proximity}, \{\overline{p_v^2}\}_{zk});\\
& \mathit{(*\ geographical\ synchronisation\ *)}\\
& \text{out}(\mathit{exit}, \{\overline{p_v^2}\}_{zk});\ \text{out}(\mathit{exit}, \overline{p_v^2})
\end{aligned}
$$

where for sake of clarity we have removed duplicate instructions. The *(* key establishment *)* marker is left empty since we consider an ideal model where the vehicles magically know the mix-zone encryption key. The *(* geographical synchronisation *)* marker indicates that the two vehicles will have to synchronise

at this point. In other words, a vehicle can execute the instructions after this point only once all the instructions before this point have been executed by both vehicles.

Having turned the scenario into a process, we instantiate this process twice using different values for *entry* and *exit* to obtain the two Vehicle processes needed for the equivalence checking. We consider the context

$$C_{\mathsf{ideal}} = \mathsf{new}\ k_{ca}; \mathrm{out}(c, \mathsf{pub}(k_{ca})); \mathsf{new}\ zk; \_$$

and ask ProVerif to try to prove observational equivalence. To overcome the limitations due to the ProVerif tool, we perform *data swapping* as described in [9].

From previous discussions it is clear that geographical synchronisation is a necessary condition for privacy, i.e. that two vehicles either enter or exit the mix-zone at the same time. More precisely, the necessary condition is that no message is sent from an *entry* location after a message has been sent from an *exit* location. If this is not satisfied then the attacker can trivially link $\overline{p_v^1}$ with $\overline{p_v^2}$, so we did not include any such scenarios in our experiments.

Results. All the scenarios we consider are listed in Figure 2 along with the obtained results. Each row is a scenario with the first columns showing where the beacon messages in the sequence are emitted. The columns to the right of the sequence show the results in the different encryption models: the first two give the results for when deterministic encryption is used and the last two for when probabilistic encryption is used. In each encryption model, the left column shows the result if the vehicles synchronise before going into the *proximity* location and the right column if they synchronise before leaving. A minus symbol ($-$) indicates that ProVerif could not prove equivalence (and found an attack trace) and a plus symbol ($+$) means that it could.

Analysis. Our results show a second necessary condition for privacy: that vehicles do not change pseudonym too early or too late. This is shown by Scenario 1 and 2 where the vehicles are still sending unencrypted beacon messages using the first pseudonym at the exit location. Similarly, Scenario 31 and 32 show that privacy is lost if they move too late; in this case the second pseudonym is used in an unencrypted beacon message at the entry location.

In the deterministic encryption model, we only have privacy in scenarios where geographical synchronisation coincides with a change of message. This condition is illustrated by Scenarios 10-14. In this group, ProVerif can prove privacy if the synchronisation is before the *proximity* location since the link between $\overline{p_v^1}$ and $\{\overline{p_v^1}\}_{zk}$ is broken. However, in Scenarios 15-20 we see from ProVerif's counterexamples that when synchronisation is before the *proximity* location, the attacker can link $\overline{p_v^1}$ and $\{\overline{p_v^1}\}_{zk}$ since they are both emitted at the same *entry* location.

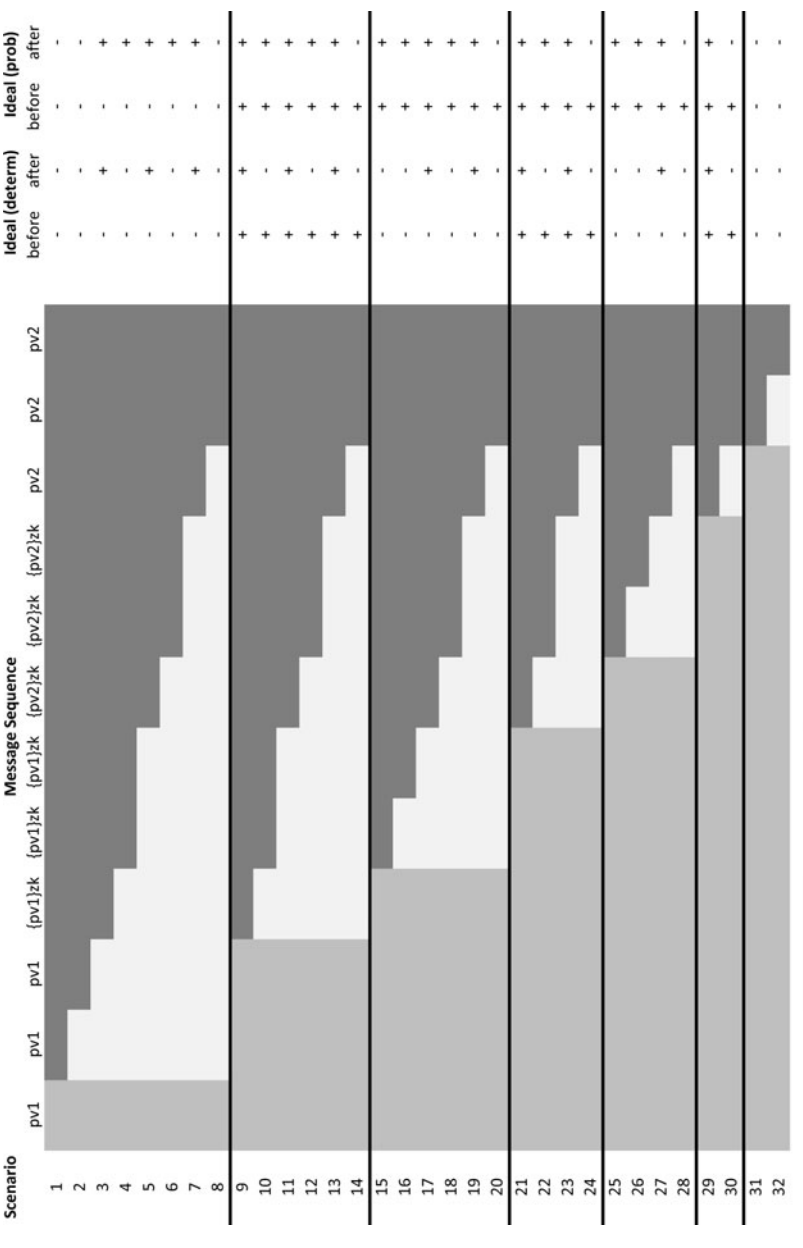


Fig. 2. Result of analysis in the CMIX model

After the synchronisation one vehicle can move to an *exit* location and emit $\overline{p_v^2}$ while the other is still at *proximity* and emitting $\{\overline{p_v^1}\}_{zk}$. By comparing ciphertexts the attacker will know which vehicle has "fallen behind" and which vehicle is at the *exit* location, in turn allowing him to link $\overline{p_v^1}$ and $\overline{p_v^2}$.

The situation changes when probabilistic encryption is used. In this case we have that ProVerif can prove equivalence for all the cases where deterministic encryption allows privacy, and in addition, scenarios where the geographical synchronisation is between two encrypted messages, e.g. Scenarios 4, 6, 15, 16. This is an important result, since it means that two vehicle only need to get into a mix zone and encrypt at the same time as another vehicle, then change the pseudonym before leaving. It seems clear that an encryption scheme that renders encrypted beacons incomparable must be used.

As a final remark we note that the results show that in our model, use of encryption is not necessary to obtain privacy: if the vehicles agree on when to change their pseudonym then no encryption is needed. This is best illustrated in Scenario 21. Although encryption is used, it has no effect since beacon messages can be trivially linked with their encryption by the location where they are emitted. Furthermore, no messages are emitted at the *proximity* location.

5.2 Privacy in the CMIX Model

Based on the conclusions of the previous section, we consider only probabilistic encryption when analysing the CMIX key distribution protocol. We consider all scenarios where privacy is provable in the ideal model. First, we add one session of the CMIX protocol to both vehicle processes, to be executed before entering the *proximity* zone. We found that in all cases where privacy was possible in the ideal model, it was also possible here[1].

We recall that according to the CMIX paper [11], a key request message is triggered in the vehicle when it either receives a message that it cannot decrypt, or when it receives an alert message from the RSU. The former situation could be used by an active attacker to trigger a second CMIX session (after the first was finished). The nearby presence of other mix-zones or simply a corrupted broadcast might also trigger a second session in the presence of a passive attacker. Hence we consider all variations of the scenarios obtained by interleaving two sequential sessions of the key establishment protocol. One session is always at the *entry* location using the first pseudonym and before emitting any encrypted beacon messages, but the location of the second session is varied between *proximity* and *exit*, and further by which of the pseudonyms it uses.

To illustrate the modelling of subscenarios we consider the variation of the scenario from the previous subsection obtained by placing the second key establishment session at the *exit* location after changing pseudonym. The process for this subscenario is similar to the vehicle process given in Section 5.1 expect that $\mathsf{Vehicle}_{\mathsf{CMIX}}(entry, p_v^1, k_v^1)$ defined in Figure 3 replaces the marker *(* key establishment *)* and $\mathsf{Vehicle}_{\mathsf{CMIX}}(exit, p_v^2, k_v^2)$ is inserted just after the *(* geographical*

[1] Full results can be found online at `http://www.cs.aau.dk/~dahl/mixzoneprivacy/`

$$
\begin{aligned}
\mathsf{Vehicle}_{\mathsf{CMIX}}(c, p_v, k_v) \stackrel{\mathsf{def}}{=}\ & \text{new } t_s; \\
& \text{let } x^s = \mathsf{sign}((\mathsf{request}, t_s), k_v) \text{ in} \\
& \text{let } x^c = \mathsf{sign}((p_v, \mathsf{pub}(k_v)), k_{ca}) \text{ in} \\
& \mathsf{out}(c, (x^s, x^c)); \\
& \quad \mathsf{in}(c, (y^e, y^c)); \\
& \quad \text{let } (x_{prsu}, x_{pkrsu}) = \mathsf{checksign}(y^c, \mathsf{pub}(k_{ca})) \text{ in} \\
& \quad \text{let } y^s = \mathsf{adec}(y^e, k_v) \text{ in} \\
& \quad \text{let } (p_v, x_{zk}, t_s) = \mathsf{checksign}(y^s, x_{pkrsu}) \text{ in} \\
& \quad \text{let } z^s = \mathsf{sign}((\mathsf{ack}, t_s), k_v) \text{ in} \\
& \quad \text{let } z^c = \mathsf{sign}((p_v, \mathsf{pub}(k_v)), k_{ca}) \text{ in} \\
& \quad \mathsf{out}\ (c, (z^s, z^c))
\end{aligned}
$$

Fig. 3. Vehicle's part of CMIX key establishment protocol

*synchronisation *)* marker. Note that to make the analysis practical the operations of the TRD are inlined.

For the analysis, we place the two instantiated vehicle processes in the context given by:

$$
\begin{aligned}
C_{\mathsf{CMIX}} \stackrel{\mathsf{def}}{=}\ & \text{new } k_{ca}; \mathrm{out}(c, \mathsf{pub}(k_{ca})); \\
& \text{new } k_{rsu}; \text{new } p_{rsu}; \mathrm{out}(c, (p_{rsu}, \mathsf{pub}(k_{rsu}))); \text{new } zk; \left(!\mathsf{RSU}_{\mathsf{CMIX}} \mid \_\right).
\end{aligned}
$$

which, contrary to the context used in the ideal model, includes the RSU.

Results. The experiments show that the CMIX key establishment protocol as described in the paper can break privacy in scenarios where it is assured in the ideal model. The reason is that the pseudonym is sent in clear in the request message. More precisely, the experiments show that if a key establishment session is triggered at the *exit* location then there is an attack when the vehicle has not yet changed its pseudonym: the key establishment session reveals the first pseudonym which can be link to the second pseudonym by the location. Perhaps less obviously, if a key establishment session is triggered at the *proximity* location then there is also an attack when the geographical synchronisation does not separate it from the unencrypted beacon messages sent using the other pseudonym. This attack is an instance of the general "fallen behind" attack that arises when both pseudonyms are revealed in locations not separated by a geographical synchronisation.

Contrary to the analysis in the ideal model, where the running time of ProVerif on a 2.5 GHz Intel Xero processor was less than a few minutes for each variation, the running time in the CMIX model ranged between a few seconds and 3 hours for each scenario.

5.3 Fixing the Key Establishment Protocol

A simple fix to the CMIX key establishment protocol that does not increase the number of rounds is to encrypt the request and the acknowledgement message

under the RSU's public key. This assumes vehicles know the certificate of the RSU before performing a key request, which could be ensured by, for instance, including the certificate in the messages broadcast from the RSU to inform vehicles about the mix-zone.

We modelled this revised protocol in ProVerif and retried all the scenarios. For most of them ProVerif was able to prove privacy in the CMIX model when there was privacy in the ideal model, but in a fraction of the scenarios (1/13) a false attack was reported. The false attack seems to be due to the stronger equivalence that ProVerif tried to prove, and arises when two key establishment sessions using the same pseudonyms are separated by a geographical synchronisation. By recording the RSU's response in the first session with the vehicle using key kv and replaying this message to a vehicle during the second session, the vehicle not using kv will fail at decryption whereas the vehicle using kv will correctly decrypt but fail at a different step in the process, namely when comparing time stamps. The observations are the same, but the processes execute differently, so ProVerif is unable to prove equivalence.

6 Conclusion

In this paper, we have proposed a formal notion of privacy for mix-zones based on classical ideas of equivalence: if the equivalence is satisfied then no attacker can link the pseudonyms used by two vehicles entering a mix-zone with the pseudonyms they use when exiting. We have seen that for an idealised vehicular mix-zone to achieve privacy requires geographical and pseudonym change synchronisation. Our experiments on a variety of scenarios suggest that probabilistic encryption gives a significantly better chance of achieving privacy than deterministic encryption. We have analysed the CMIX proposal for key distribution in mix-zones, and shown that the use of the protocol can inadvertently prevent privacy from being achieved in many scenarios. We have shown that the CMIX protocol can be modified to preserve privacy.

As future work it seems natural to examine to what extent our experiments on a fixed series of beacon signals identical for both vehicles captures the space of possible scenarios satisfactorily. Although some cases of vehicles performing different scenarios are captured by our experiments, the case where one vehicle changes pseudonym at the entry location while the other changes at the exit location is for instance not captured. Another limitation of our modelling is that the messages of a key establishment session cannot be emitted across several locations. If the attacker can identify to which session messages belong then a session spanning across a geograpical synchronisation might break privacy, even against a passive attacker. Capturing this type of attack is also left for future work.

We plan to examine the API of the on board tamper-resistant cryptographic device to see how it might prevent insider attacks, i.e. attacks by an adversary who owns a legitimate vehicle. We also plan to investigate more fully the properties of our modelling approach, by e.g. comparing our notion of privacy to existing notions of anonymity, untraceability and unlinkability in the literature.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proc. 28th ACM Symposium on Principles of Programming Languages (POPL 2001), pp. 104–115. ACM Press, New York (2001)
2. Arapinis, M., Chothia, T., Ritter, E., Ryan, M.: Analysing unlinkability and anonymity using the applied pi calculus. In: Proc. 23rd IEEE Computer Security Foundations Symposium, CSF 2010 (to appear, 2010)
3. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Computing 2(1), 46–55 (2003)
4. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming 75(1), 3–51 (2008)
5. Brusó, M., Chatzikokolakis, K., den Hartog, J.: Formal verification of privacy for RFID systems. In: Proc. 23rd IEEE Computer Security Foundations Symposium, CSF 2010 (to appear, 2010)
6. 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)
7. Buttyán, L., Holczer, T., Weimerskirch, A., Whyte, W.: SLOW: A practical pseudonym changing scheme for location privacy in VANETs. In: IEEE Vehicular Networking Conference (VNC), Tokyo, Japan, October 2009, pp. 1–8 (2009)
8. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17(4), 435–487 (2009)
9. Delaune, S., Ryan, M.D., Smyth, B.: Automatic verification of privacy properties in the applied pi-calculus. In: Karabulut, Y., Mitchell, J., Herrmann, P., Jensen, C.D. (eds.) Proc. 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM 2008), Trondheim, Norway, June 2008. IFIP Conference Proceedings, vol. 263, pp. 263–278. Springer, Heidelberg (2008)
10. Doetzer, F.: Privacy issues in vehicular ad hoc networks. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 197–209. Springer, Heidelberg (2006)
11. Freudiger, J., Raya, M., Félegyházi, M., Papadimitratos, P., Hubaux, J.-P.: Mix-zones for location privacy in vehicular networks. In: Proc. of ACM Workshop on Wireless Networking for Intelligent Transportation Systems, WiN-ITS 2007 (2007)
12. Parno, B., Perrig, A.: Challenges in securing vehicular networks. In: Proc. 4th Workshop on Hot Topics in Networks (November 2005)
13. Sleet, D., Peden, M., Scurfield, R.: World report on traffic injury prevention. World Health Organization Report (2004)
14. Raya, M., Hubaux, J.-P.: The Security of Vehicular Ad Hoc Networks. In: Proc. 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2005), pp. 11–21 (2005)
15. Safespot project (2006-2010), http://www.safespot-eu.org/
16. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)

17. IEEE standard. IEEE Trial-Use Standard for Wireless Access in Vehicular Environments – Security Services for Applications and Management Messages (approved June 8, 2006)
18. van Deursen, T., Mauw, S., Radomirovic, S.: Untraceability of RFID protocols. In: Onieva, J.A., Sauveron, D., Chaumette, S., Gollmann, D., Markantonakis, K. (eds.) WISTP 2008. LNCS, vol. 5019, pp. 1–15. Springer, Heidelberg (2008)

IntPatch: Automatically Fix Integer-Overflow-to-Buffer-Overflow Vulnerability at Compile-Time

Chao Zhang, Tielei Wang, Tao Wei, Yu Chen, and Wei Zou

Institute of Computer Science and Technology, Peking University
Key Laboratory of Network and Software Security Assurance (Peking University), Ministry of Education
{zhangchao,wangtielei,weitao,chenyu,zouwei}@icst.pku.edu.cn

Abstract. The Integer-Overflow-to-Buffer-Overflow (*IO2BO*) vulnerability is an underestimated threat. Automatically identifying and fixing this kind of vulnerability are critical for software security. In this paper, we present the design and implementation of IntPatch, a compiler extension for automatically fixing IO2BO vulnerabilities in C/C++ programs at compile time. IntPatch utilizes classic type theory and dataflow analysis framework to identify potential IO2BO vulnerabilities, and then instruments programs with runtime checks. Moreover, IntPatch provides an interface for programmers to facilitate checking integer overflows. We evaluate IntPatch on a number of real-world applications. It has caught all 46 previously known IO2BO vulnerabilities in our test suite and found 21 new bugs. Applications patched by IntPatch have a negligible runtime performance loss which is averaging about 1%.

1 Introduction

The Integer Overflow to Buffer Overflow vulnerability (*IO2BO* for short), defined in Common Weakness Enumeration (CWE-680 [7]), is a kind of vulnerability caused by integer overflows, i.e. an integer overflow occurs when a program performs a calculation to determine how much memory to allocate, which causes less memory to be allocated than expected, leading to a buffer overflow.

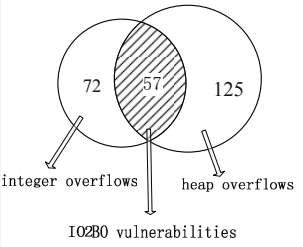
For instance, figure 1(a) shows a typical IO2BO vulnerability which existed in the old version of Faad2 [11]. In this code snippet, the argument `mp4ff_t *f` represents a mp4-file stream. Routine `mp4ff_read_int32(f)` at line 467 reads an integer value from external file `f` without any checks. The unchecked integer value (e.g. $0x80000001$) is then used in a memory allocation function at line 469. If an overflow occurs there, a smaller than expected memory (e.g. $0x80000001 * 4 = 4$) will be allocated. At line 483, some values read from external file without any checks will be written to the allocated memory chunk. Because the allocated memory is smaller than expected, these writes will corrupt the heap and may lead to arbitrary code execution [40].

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 71–86, 2010.

```
458. static int32_t mp4ff_read_ctts(mp4ff_t *f)
459. {
460.   // sth. omitted ...
467.   p_track->ctts_entry_count = mp4ff_read_int32(f);
468.
469.   p_track->ctts_sample_count =
       (int32_t*) malloc (p_track->ctts_entry_count * sizeof(int32_t));
470.   p_track->ctts_sample_offset =
         (int32_t*) malloc (p_track->ctts_entry_count * sizeof(int32_t));

481.   for (i = 0; i < p_track->ctts_entry_count; i++)
482.   {
483.     p_track->ctts_sample_count[i] = mp4ff_read_int32(f);
484.     p_track->ctts_sample_offset[i] = mp4ff_read_int32(f);
485.   }
486.   return 1;
488. }
```

(a)



(b)

Fig. 1. (a)A real-world IO2BO vulnerability in Faad2. (b)Number of vulnerabilities reported by NVD from April 1, 2009 to April 1, 2010. There are 129 (=72+57) integer overflows and 182 (=57+125) heap overflows. More than 44% (=57/129) of integer overflows are IO2BO vulnerabilities.

IO2BO is an underestimated threat. In recent years, we have witnessed that IO2BO is being widely used by attackers, such as bypassing the SSH authentication in [30] and the heap corruption attack in [40]. Moreover, according to the statistical data (from April 2009 to April 2010) in the National Vulnerability Database (NVD [17]), nearly a half of integer overflow vulnerabilities and one third of heap overflow vulnerabilities are IO2BO, as shown in Fig. 1(b).

The main reason that IO2BO is so popular is that many programmers have not yet realized the danger brought by integer overflows. Even for those who are aware of integer overflows, fixing these bugs is tedious and error-prone. For example, CUPS [4], a well-known open source printing system, has an IO2BO vulnerability in the function `_cupsImageReadPNG` [6]. CUPS first released a patch, but the initial patch failed to fix the vulnerability properly [5]. The developers had to release another patch to completely fix this vulnerability. Moreover, the C99 standard [12] specifies that signed overflow is considered as an undefined behavior, thus some patches that work properly in some compiler environments may fail in others.

Some compilers or compiler extensions such as RICH [25] have the ability to insert extra code to capture integer overflows at runtime. For example, with `-ftrapv` option, GCC can insert additional code to catch each overflow at runtime. However, there exists benign integer overflows deliberately used in random numbers generating, message encoding/decoding or modulo arithmetic [25], and thus such full instrumentation inevitably generates false positives. Furthermore, the instrumented programs usually suffer from a non-trivial performance overhead.

There are a number of integer overflow detection studies, such as [41] [38] [29] [28]. For the static-analysis-based tools, false positives are non-negligible.

Manually analyzing and patching the potential integer overflows is still error-prone. For the dynamic-analysis-based tools, the main disadvantage is their false negatives. Although many dynamic analysis systems (such as KLEE [26], EXE [27], CUTE [39], DART [35]) use symbolic execution techniques to improve code coverage and can be extended for detecting integer overflows, the analysis results are not sound.

In this paper, we present IntPatch, a tool capable of identifying potential IO2BO vulnerabilities and fixing them automatically. First, we use a type analysis to detect potential IO2BO vulnerabilities. Then, for each candidate vulnerability, another analysis pass is made to locate the points to fix at.

In the type analysis process, we consider each variable's taintedness and whether it overflows. If a tainted (thus untrusted) and maybe overflowed variable is used in a memory allocation function, there is a potential IO2BO vulnerability. In the locating and patching process, we use backward slicing [42] technique to identify those related vulnerable arithmetic operations and then insert check statements after them to catch vulnerability at runtime.

We implement IntPatch based on LLVM (Low Level Virtual Machine [36,37]) and evaluate its performance on a number of real-world open-source applications. Experiments show that IntPatch has caught all 46 previously known IO2BO vulnerabilities and it helps us find 21 zero-day bugs. These zero-day bugs are in the process of being submitted. Compared to their original versions, the patched applications have a negligible runtime performance loss which is averaging about 1%. Thus, IntPatch is a powerful and lightweight tool which can efficiently capture and fix IO2BO vulnerabilities. It could help programmers accelerate software development and greatly promote programs' security.

Contributions. This paper presents an automatic tool for efficiently protecting against IO2BO vulnerabilities. Specially, we:

- Survey 46 IO2BO vulnerabilities and compare some of them with their patched versions. We figure out that fixing IO2BO is tedious and error-prone.
- Construct a type system to model IO2BO vulnerabilities and present a framework for automatically identifying and fixing them at compile time.
- Provide an API for programmers who want to fix IO2BO vulnerabilities manually.
- Implement a tool called IntPatch. It inserts dynamic check code to protect against IO2BO. The patched version's performance overhead is low, on average about 1%. Experiments also show that IntPatch is able to capture all previously known IO2BO vulnerabilities.
- Identify 21 zero-day bugs in open-source applications with IntPatch.

Outline. We first describe what an IO2BO-type vulnerability is and how complicated it is when we try to fix it in Sect. 2. Our system overview and the type system we used to model IO2BO vulnerability are shown in Sect. 3. In Sect. 4,

we discuss our system's implementation, including the interface provided for programmers. Section 5 evaluates our work, and shows the performance and false positives. Related work and conclusion are discussed in Sect. 6 and Sect. 7.

2 Background

Although integer overflows may cause many other vulnerability types [23,25], the most typical case is IO2BO. In this section, we will discuss in detail what an IO2BO vulnerability is and what difficulties programmers may meet when they try to fix it.

2.1 What Is an IO2BO Vulnerability?

An IO2BO vulnerability, as defined in CWE [7], is a kind of vulnerability caused by integer overflow. Specifically, when an overflowed value (smaller than expected) is used as the size of a memory allocation, subsequent reads or writes on this allocated heap chunk will trigger a heap overflow vulnerability. A typical instance has been shown in the previous section.

Characteristics of IO2BO Vulnerabilities. We have surveyed 46 IO2BO vulnerabilities consisting of 17 bugs found by IntScope [41] and 29 bugs reported in CVE [2], Secunia [21], VUPEN [22], CERT [1] and oCERT [18].

According to the survey, we find that an exploitable IO2BO vulnerability has many significant features, similar to those presented in [41]. First, the program reads some user-supplied thus untrusted input. Then, the input value is used in an arithmetic operation to trigger an integer overflow. Finally, the overflowed value is propagated to the memory allocation function, and thus a smaller than expected memory is allocated.

Overflow in the Context of IO2BO Cannot be Benign. As mentioned in the introduction, it is difficult to distinguish integer overflow vulnerabilities from benign overflows. However, we argue that, in a context of IO2BO, the involved integer overflow cannot be benign.

More precisely, if an untrusted value triggers an integer overflow and then the overflowed result is used in memory allocation, the involved integer overflow is a real vulnerability. Usually, the overflowed result is smaller than its expected value. Besides, allocating a small memory chunk rather than a huge one doesn't cause any warnings or failures. Thus, programmers have no idea that the allocated memory is smaller than expected. It is note worthy that, further actions such as read/write will still be taken on the expected memory chunk, and then trigger buffer overflows. So, the involved integer overflow is a real vulnerability.

With this argument, we can conclude that, if an integer overflow in the context of IO2BO is caught at runtime, this overflow should be a real vulnerability. Thus, it is possible to construct a fixing mechanism with a low false positive rate for protecting against IO2BO.

2.2 How to Fix IO2BO Vulnerabilities?

Among the 46 IO2BO vulnerabilities, we investigate 18 patches of them. We find that manually fixing integer overflows is tedious. Even worse, some patches cannot fix integer overflows correctly.

Input Validation. Fixing integer overflows is essentially an input validation problem. Incomplete input validation is the origin of IO2BO vulnerability.

The widely used method for checking integer overflow in practice looks like:

$$\text{if } (b \neq 0 \quad \&\& \quad (a * b)/b \neq a) \qquad \text{MSG("overflow occurs")};$$

However, this method has some problems when programs are compiled with GCC. We will discuss later.

On assembly language level, to check an integer overflow is also an annoying work. For example, on x86 architecture, methods for checking overflows in signed/unsigned multiplications/additions are different. Instructions `jo`, `jc`, and `js` should be used in combination to check those overflows [13].

Fallibility and Complexity. Fixing integer overflow manually is **error-prone**. Figure 2 illustrates an erroneous patch in CUPS. Field `img->ysize` is propagated from the argument `height` which is read from external. If this field is given a big enough value, operation `img->ysize*3` may overflow first, then it will make the check in this patch useless. For example, let `img->xsize=2` and `img->ysize=0x60000000`, then `img->ysize*3` will be equal to $0x20000000$ (overflowed). Then the product of `img->xsize`, `img->ysize` and 3 overflows but this overflow cannot be caught by the check in this patch.

```
    png_get_IHDR(pp, info, &width, &height,          // untrusted source read from file
          &bit_depth, &color_type, &interlace_type, &compression_type, &filter_type);

    img->xsize = width;                               // propagate
    img->ysize = height;

-   in = malloc(img->xsize * img->ysize * 3);         // overflow occurs, and
                                                      // used in sensitive operation
+   {
+     bufsize = img->xsize * img->ysize * 3;
+     if ((bufsize / (img->ysize * 3)) != img->xsize)   // incorrect patch
+         fprintf(stderr, "…" );
+   }
+   in = malloc(bufsize);
```

Fig. 2. Incorrect Patch in CUPS-1.3 for vulnerability whose ID is CVE-2008-1722 [6]

The correct method for checking overflow in this expression will take two steps. First, check whether expression `img->ysize*3` overflows. Then, check whether expression `product*img->xsize` overflows, where `product` is the product of `img->ysize` and 3.

Suppose we want to check an overflow in a long expression such as `a*b*c*d*e*f`, it follows that five sub-expressions should be checked separately. Since methods for checking each sub-expression are similar, it is too **tedious** for a programmer to manually fix integer overflows.

Compiler Problem. In this section, we will explain why the widely used method for checking integer overflow listed above will be useless when programs are compiled with GCC.

The C99 standard [10] specifies that signed overflow is considered as undefined behavior, thus implementation specific. And the GCC developers think that programmers should detect an overflow before an overflow is going to happen rather than using the overflowed result to check the existence of overflow. The detailed discussion between programmers and GCC developers can be found in [9].

As a result, the condition statement `if(a*b/b!=a)` in the widely used method may be removed totally when the program is compiled with GCC, especially when it is compiled with optimization options. The Python interpreter is a victim of this problem. Python developers use a check like `if(x>0 && x+x<0)` to test whether `x+x` (where `x` is a signed int variable) could overflow. However, the check may be optimized and discarded by GCC compiler [20], so that the code is still vulnerable. See [20] for more information.

So, freeing programmers from fixing integer overflows is necessary. Compilers should be responsible for fixing integer overflows.

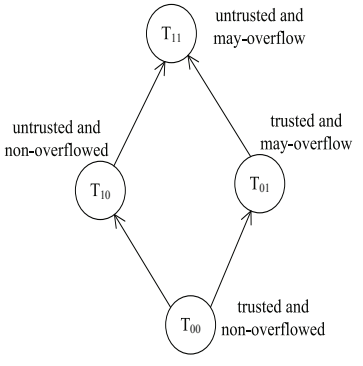
3 System Overview

In this section, we describe the overview of our system which is aimed at fixing IO2BO vulnerabilities automatically. To fix IO2BO vulnerabilities, we must identify them first. According to the features of IO2BO vulnerabilities, we use a type analysis to detect them. Then another analysis is made upon those candidate vulnerabilities to decide which points to fix at. Finally, runtime check statements are inserted at those points.

3.1 Identify Potential IO2BO Vulnerabilities

As mentioned above, an IO2BO vulnerability has some significant features. Thus, properties of variables, such as whether they are trusted and whether they may be overflowed, are considered. Then a type system is constructed and a type analysis to identify potential IO2BO vulnerabilities is made.

Type System. Figure 3(a) shows our type system. Our type system forms a lattice. The bottom of the lattice is type T_{00}. Variables with this type are trusted, i.e. their values are not from program input, and non-overflowed. The top of this lattice is type T_{11}, which represents for untrusted and may-overflow. Variables with this type origins from program input, and origins from some variables possibly overflowed. Our type system also has another two types T_{10} and T_{01} which respectively represents for untrusted and may-overflow.



$$\frac{\Gamma \vdash v_1 : \tau \qquad v_2 := v_1}{\Gamma \vdash v_2 : \tau} \text{ (assignment)}$$

$$\frac{\Gamma \vdash v_1 : \tau_1 \qquad \Gamma \vdash v_2 : \tau_2 \qquad v := v_1 \otimes v_2}{\Gamma \vdash v : (\tau_1 \vee \tau_2 \vee T_{01})} \text{ (arith-ops)}$$

$$\frac{\Gamma \vdash v_1 : \tau \qquad *v := v_1}{\Gamma \vdash *v : \tau \qquad tp\_v = tp\_v \vee \tau} \text{ (store operation)}$$

$$\frac{v_1 \sim \mathbf{V} \qquad v_2 := *v_1}{\Gamma \vdash v_2 : (\bigvee_{v \in V} tp\_v \ \vee \ tp\_v_1)} \text{ (load operation)}$$

(b)

Fig. 3. (a)Our type system, (b)type inference rules in our system

If a variable with type T_{11} is assigned to a variable which expects type T_{00}, there is a type conflict, which means there is a potential IO2BO. Due to the characteristics of IO2BO vulnerabilities, other type casting are allowed.

Type Initialization. Our type system is different from embeded type system of the C/C++ programming language. So, when applying our type system on programs, we must assign each variable with a type. It is impossible to assign each variable with a type manually. We just assign variables at key points with specific types. For example, if a variable is read from program input (called **sources**), then type T_{10} will be assigned to it. If a variable is used in memory allocation (called **sinks**), it will be assigned with type T_{00}. Then, following type inference rules are used to decide the remainder variables' types.

Type Inference. Figure 3(b) shows our type inference rules.

Assignment Statement. The right-hand side variable's type will be directly assigned to the left-hand side variable.

Arithmetic Operation. Overflow could only occurs in addition, subtraction, multiplication or left shift operation. So, the listed rule for arithmetic operation covers only these four kinds of operations. The result's type is joined by the

two operands' types and T_{01}. It means that, the result may overflow, and is untrusted if any one of its operand is untrusted.

Store Operation. Type inference rule for memory store operation is a little complex. In order to make a conservative analysis, for each pointer variable v, we record an additional type information tp_v, which represents the *possible Type of those memory chunks Pointed by* v. If variable v_1 with type τ is stored into a memory pointed by v, the target memory will be assigned with type τ, and the memory's type information will be joined into tp_v.

Load Operation. If variable v_2 is loaded from memory pointed by v_1, it may have a type same as any memory pointed by v_1. Besides, if pointer v_1 alias to pointers in set V (denoted as $v_1 \sim \mathrm{V}$), then variable v_2's type may also be same as any memory pointed by any pointer v in V. Thus, variable v_2's type is the upper bounds of tp_v_1 and tp_v for each pointer v in V.

Misc. Remaining operations' type inference rules are straightforward. Thus they are not listed here.

Type Analysis Process. For each application to be analyzed, a configuration file which defines sources (i.e. functions which read input) and sinks (i.e. memory allocation functions) is manually provided. This configuration file is read in and used to initialize our type system. Then, a dataflow analysis applying our type inference rules is made. As explained above, type T_{00} is expected at sinks. If the type inferred from the dataflow analysis is T_{11}, there is a type conflict, i.e. there is a potential IO2BO vulnerability.

3.2 Locate Vulnerable Arithmetic Operations and Patch

After the type analysis, some candidate IO2BO vulnerabilities are generated. The type analysis is conservative, and thus it is sound (i.e. there is no false negatives). However, this type analysis is path-insensitive, thus there may be many infeasible paths which are reported as IO2BO vulnerabilities. Besides, the alias analysis in LLVM we used is conservative, it may also introduce additional false positives. Leaving all these candidate vulnerabilities for programmers to validate is terrible. In this section, we introduce an automatic fixing mechanism which can reduce false positives and protect programs against IO2BO vulnerabilities.

First, our approach identifies those related vulnerable arithmetic operations (i.e. overflow occurs here will further triggers the IO2BO vulnerability). Then, for each vulnerable arithmetic operation, statements for checking overflow at runtime are automatically inserted after it.

To locate vulnerable arithmetic operations, a backward analysis is made for each candidate IO2BO vulnerability. Variables at each vulnerable sink are focused. Techniques like backward slicing [42] are then used to find other variables which may affect the focused variable. If a variable found by slicing is with type T_{11} and the corresponding statement is an arithmetic operation, this statement is thought as a vulnerable arithmetic operation. Finally, statements for checking overflow at runtime are inserted after those vulnerable arithmetic operations.

As argued in Sect. 2.1, integer overflows in the context of IO2BO are usually vulnerable. Thus, integer overflows caught by this fixing mechanism at runtime are real vulnerabilities, i.e. this fixing mechanism can reduce false positives.

4 Implementation

In this section, we present the implementation of our system. We implement our system as a tool IntPatch based on LLVM. Figure 4 shows the structure of IntPatch.

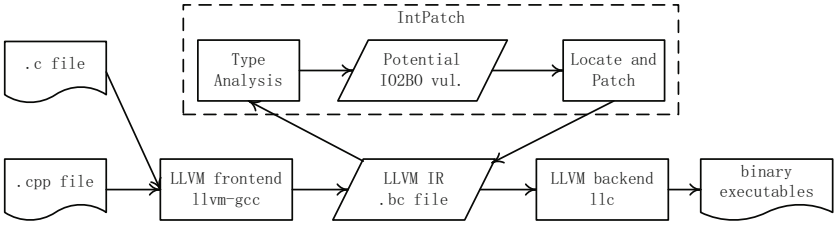


Fig. 4. Structure of IntPatch

IntPatch first makes an classic dataflow analysis to analyze each variable's type and identify potential IO2BO vulnerabilities. Then, for each potential vulnerability, it makes a slicing to find the vulnerable arithmetic operations. Finally, check statements are inserted after those vulnerable operations to catch runtime bugs.

4.1 LLVM

LLVM [36,37] is a compiler infrastructure which supports effective optimization and analysis at compile time, link-time, run-time and offline. IntPatch utilizes some useful features or interfaces provided by LLVM.

For example, LLVM provides us an easy-to-use CFG which facilitates iterating over whole program. All memory accesses are explicitly using load and store instructions in LLVM. Thus, our type inference rule for load and store operation is easy to be applied. LLVM's intermediate representation (IR) is in SSA (Static Single Assignment [32]) form and thus facilitates our dataflow analysis. In addition, LLVM provides some intrinsic instructions for catching integer overflows. LLVM also provides some classic alias analysis pass for us to use, which helps us a lot when we make type analysis.

4.2 Type Analysis

IntPatch uses a type analysis to identify potential IO2BO vulnerability. In LLVM, all kinds of instructions and operands are instances of class `llvm::Value`.

A value which represents an instruction could be used as another instruction's operand. That is to say, a value representing an instruction also represents the result of the instruction, thus can be thought as a variable.

We maintain a map from such variables to types. Because LLVM's IR is in SSA form, each variable has only one definition point. Thus, the type information of any variable won't change.

A predefined file which annotates what are sources and sinks is read in to initialize the mapping relationship between variables and types. Then we use classic dataflow analysis method [24] to analyze each variable's type. Type inference rules are applied on each instruction. At each basic block's entry, there may be some phi-nodes [32], which are introduced by SSA. For each of these phi-nodes, such as $v = \phi(v_1,\ v_2,\ ...,\ v_n)$, we join types of variable $v_1,\ v_2,\ ...,\ v_n$ together and assign it to variable v.

When the dataflow analysis analyzes variables at sinks, we do a type check here. If variables at sinks are with type T_{11} according to the analysis's result, there is a type conflict, and thus a potential IO2BO vulnerability exists.

This type analysis process is implemented as a pass in LLVM and its result can be used by other passes. Because our analysis is interprocedural, our analysis pass is an instance of `llvm::ModulePass` and needs to be invocated at link-time.

4.3 Locate Vulnerable Arithmetic Operations and Patch

The type analysis can identify potential IO2BO vulnerabilities. Our remainder task is to fix IO2BO vulnerabilities automatically. Fixing should be complete, i.e. if a bug is caught at runtime, it should be a real bug. In other word, a mechanism is needed to reduce false positive rates. Otherwise, users will complain about the program's quality.

We implement another analysis pass to identify those vulnerable arithmetic operations. This analysis uses classic slicing method [42] to find related variables. If the related variable's type is T_{11} and the variable (i.e. instruction) is an arithmetic operation, a check statement is inserted after that instruction. We use intrinsic instructions provided by LLVM such as `llvm.sadd.with.overflow` to check integer overflow. If an overflow occurs, we redirect the control flow to a predefined function. By default, this function blocks the program and waits for user debugging. This function can also be specified by programmers.

Using these two analysis pass, IntPatch is able to automatically identify and fix IO2BO vulnerabilities over full programs with a reasonable false positive rate.

4.4 Another Compiler Interface

However, in some situations, programmers still want to fix IO2BO vulnerabilities manually. In order to shield programmers from the tedious and error-prone fixing work, IntPatch also provides an easy-to-use interface. With this interface, programmers can specify what expressions to be monitored and what actions will be taken when overflow occurs in these expressions.

This interface, named `IOcheck(int exp, void (*f)())`, is implemented as an API. Programmers pass the expression to be monitored into the first argument, and pass the overflow handler function into the second argument. The second argument is default set to NULL, which means we will use a handler predefined in IntPatch.

In order to support this API, we need to make a few modifications to the original analysis. In the type analysis process, we just treat the first argument of function `IOcheck()` as sinks. And in the slicing process, we just need to change the inserted overflow handler function to the handler specified by programmers. Besides, we provide an library for function `IOcheck()` which does nothing in fact. This library will be linked by LLVM.

5 Evaluation

We evaluate IntPatch with several real-world open-source applications, including libtiff [14], ming [15], faad2 [11], dillo [8], gstreamer [12] and so on. The evaluation was performed on an Intel Core2 2.40GHz machine with 2GB memory and Linux 2.6.27.25 kernel.

5.1 Check Density

We first measure how many checks IntPatch inserts into programs. Table 1 shows, for each benchmark program, the number of total instructions in the program (in LLVM IR form), the number of arithmetic operations in the program, and the number of checks inserted by the IntPatch. Then the checking ratio is calculated, i.e. (number of checks)/(number of arithmetic operations).

Table 1. Number of checks inserted

application	# inst	# arith-ops	# checks	ratio
libtiff-3.8.2	781212	20739	1751	8.44%
faad2-2.7	37993	1189	150	12.6%
ming-0.4.2	35901	1375	241	17.5%
dillo-2.0	641574	8053	345	4.28%
gstreamer-0.8.5	2060335	10683	1067	9.98%

Results show that, there are lots of arithmetic operations (about one tenth) which may affect memory allocations. In fact, this ratio is a little bit higher than that in regular applications, because most of the test suites are image-related applications which needs to allocate a lot of memory. Compared to results in [28] and [25], the checking ratio is very low.

5.2 Performance Overhead

In this section, we present the performance overhead of IntPatch. Our experiments show that the overhead is quite low, on average about 1%. Table 2 shows

Table 2. Performance of IntPatch

application	original (s)	patched (s)	overhead
ming-0.4.2	236.143	239.549	1.44%
libtiff-3.8.2	127.571	129.123	1.01%
dillo-2.0	3.762	3.805	1.14%
faad2-2.7	361.163	364.478	0.91%

the overhead of applications patched by IntPatch relative to the uninstrumented versions (both compiled with the same options).

We test *ming*, a library for generating Macromedia Flash files (.swf), with benchmark PNGSuite [19]. PngSuite is a test-suite containing 157 different PNG format images for PNG applications. These PNG files are converted into flash files using *ming* and the consumed time is recorded.

For *dillo*, we test its CSS rendering speed using a CSS benchmark devised by *nontroppo* [3]. *Libtiff* is tested with a pack of TIFF format files distributed together with it. These tiff files are compressed to JPEG format files using *libtiff* and the consumed time is recorded. For faad2, we use it to decode 100 MPEG-4 format videos randomly downloaded from Mp4Point [16].

5.3 False Positives and False Negatives

As mentioned above, our type analysis is conservative, and thus our analysis is sound (i.e. no false negatives). In other words, any vulnerability that satisfies IO2BO's features will be caught by the type analysis.

In order to evaluate the false positive rate of IntPatch, we test these applications instrumented by IntPatch with normal and malicious inputs. Each application is fed with normal inputs described in Sect. 5.2 and with $2 \sim 3$ malicious inputs (e.g. crafted image files). Results show that all normal inputs don't trigger the runtime check and while malicious inputs both trigger the check. That is to say, no false positives exist. However, the test is not sufficient and the the code coverage rate is low, and thus IntPatch may still has false positives.

In fact, our type analysis and slicing analysis are path-insensitive, infeasible paths may bring false positives to IntPatch. The conservative alias analysis in LLVM we used also brings some false positives.

In addition, integer overflow checks (called sanitization routine) inserted by programmers will also lead to false positives. That is because the sanitization routine will untaint the variable, but our type analysis process hasn't considered this semantic effection on the type propagation. On the other hand, sanitization routines are at semantic level and hard to be detected. One possible solution is that programmers give up customized sanitization routines and use the interface `IOcheck()` provided by IntPatch only.

5.4 Zero-Day Bugs

The type analysis pass in IntPatch has generated many candidate IO2BO vulnerabilities. Of course, there are many false positives. With manual validation, we can identify real vulnerabilities. During our unfinished time-consuming validation process, we discover 21 new IO2BO vulnerabilities in 6 applications, as shown in Table 3.

Table 3. Zero-Day Bugs detected by IntPatch

application	swftools	Inkscape	gnash	ming	faad2	libtiff
version	0.9.0	0.46	0.8.5	0.4.2	2.7	3.8.2
# bugs	2	4	5	3	3	4

For example, we found a vulnerability in function `readPNG` in ming-0.4.2. Value `png.height` is read from an input PNG file. This value then multiplies a constant without any checks. The result of the multiplication is further used in function `malloc`. Finally, data from the input PNG file is read into the allocated memory. It is a typical IO2BO vulnerability.

We have submitted some of these zero-day vulnerabilities to security service provider such as Secunia [21] and oCert [18]. Some of the submissions, such as the vulnerability in libtiff (CVE-2009-2347), have been confirmed. Corresponding patches from vendors has been released or are in progress. Considering that other vulnerabilities are still in the process of being submitted or fixed, we do not want to provide further detailed information here.

5.5 Limitation

Our work is based on LLVM, which is still in development stage. So certain applications might have troubles being compiled with LLVM. Furthermore our analysis pass is time-consuming. These drawbacks limit the domain of IntPatch's applications.

In our implementation, IntPatch depends heavily on alias analysis. However, alias analysis is a well-known problem in static analysis. Its accuracy and performance will affect IntPatch's results.

Programmers' sanitization routines are not encouraged as mentioned above. This limitation is not friendly to programmers.

6 Related Work

Many efforts have been made on integer overflow vulnerabilities. Followings are some representative works.

Shuo Chen et al. presented a FSM-based method [29] and uses finite state machines (FSM) to identify integer overflows. Experts summarize a finite state machine representing the integer overflow vulnerability first. Then a tool is used to check whether there are integer overflow vulnerabilities. It needs a lot of expert's effort and the FSM for distinct applications may be different. Thus, it is not a general solution.

Ramkumar Chinchani et al. [31] describe each arithmetic operation formally and then utilize architecture characteristics to check each arithmetic operation and catch integer overflow at runtime [31]. This method doesn't pay much attention on distinguishing benign and unexpected overflows, thus there are lots of false positives.

The sub-typing method presented by Brumley et al. [25] formalizes the semantics for safe integer operations in C. Overflow checks are inserted after each arithmetic operations to capture runtime overflows. It protects against many kinds of integer errors, including signedness error, interger overflow/underflow or truncation error. They implement a prototype called RICH and found several zero-day bugs too. However, benign and unexpected overflows are not distinguished either.

The method presented by Ceesay [28] utilizes type qualifiers theory [33] and a tool CQUAL [34] to detect type conflicts. Their work is implemented in the preprocessing step. They extend traditional type system with new type qualifier `trusted` similar to embeded type qualifier `const`. Then a type analysis is made and find all type conflicts. Each type conflict is reported as a potential vulnerability.

Both of these methods treat all kinds of integer overflow vulnerabilities, and suffer from the indistinguishability between benign overflows and unexpected overflows. Thus, their false positive rates are high.

Our paper focus on the most typical integer overflow vulnerability and tries to present a sound solution. Our type system is more complex and effective than Ceesay's. The final result shows that our method is effective.

7 Conclusion

This paper surveys many IO2BO vulnerabilities, and presents a framework to model and automatically fix this kind of vulnerability. A prototype tool IntPatch is implemented based on LLVM. Experiments show that IntPatch is powerful and lightweight and can effectively defend against IO2BO vulnerabilities. Twenty-one zero-day vulnerabilities are found as a byproduct.

References

1. Carnegie Mellon University's Computer Emergency Response Team, `http://www.cert.org/advisories/`
2. Common vulnerabilities and exposures, `http://cve.mitre.org`

3. Cssbench: a css benchmark devised by nontroppo, http://www.howtocreate.co.uk/csstest.html
4. CUPS: a standards-based, open source printing system developed by Apple Inc., http://www.cups.org/
5. Cups' erroneous patch, http://www.cups.org/str.php?L2974
6. CUPS Vulnerability, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1722
7. Cwe-680: Io2bo vulnerabilities, http://cwe.mitre.org/data/definitions/680.html
8. Dillo: a lightweight browser, http://www.dillo.org
9. Discussion between programmers and gcc developers, http://gcc.gnu.org/bugzilla/show_bug.cgi?id=30475#c2
10. Draft of the c99 standard with corrigenda tc1, tc2, and tc3 included, http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf
11. FAAD2: A MPEG-4 and MPEG-2 AAC Decoder, http://www.audiocoding.com/faad2.html
12. GStreamer: a framework for streaming media applications, http://gstreamer.freedesktop.org/
13. Intel 64 and ia-32 architectures software developer's manuals, http://www.intel.com/products/processor/manuals/
14. libtiff: TIFF Library and Utilities, http://www.libtiff.org/
15. Ming: a library for generating Macromedia Flash files, http://www.libming.org/
16. Mp4point: a source for free mp4 / mpeg-4 video movie clips, http://www.mp4point.com/
17. National vulnerability database, http://nvd.nist.gov/
18. oCERT: Open Source Computer Emergency Response Team, http://www.ocert.org/
19. Pngsuite: The "official" test-suite for png applications like viewers, converters and editors, http://www.schaik.com/pngsuite/
20. Python interpreter suffers from gcc's behavior, http://bugs.python.org/issue1608
21. Secunia: a Danish computer security service provider, http://secunia.com/
22. Vupen: a company providing security intelligence, http://www.vupen.com/english/
23. Ahmad, D.: The rising threat of vulnerabilities due to integer errors. IEEE Security and Privacy 1(4), 77–82 (2003)
24. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Princiles, Techniques, and Tools, 2nd edn. Addison-Wesley, Reading (2006)
25. Brumley, D., Chiueh, T.c, Johnson, R., Lin, H., Song, D.: Rich: Automatically protecting against integer-based vulnerabilities. In: Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007) (2007)
26. Cadar, C., Dunbar, D., Engler, D.: Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008), San Diego, CA, USA (2008)
27. Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: Exe: automatically generating inputs of death. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006 (2006)

28. Ceesay, E., Zhou, J., Gertz, M., Levitt, K., Bishop, M.: Using type qualifiers to analyze untrusted integers and detecting security flaws in c programs. Detection of Intrusions and Malware & Vulnerability Assessment (2006)
29. Chen, S., Kalbarczyk, Z., Xu, J., Iyer, R.K.: A data-driven finite state machine model for analyzing security vulnerabilities. In: IEEE International Conference on Dependable Systems and Networks, pp. 605–614 (2003)
30. Chen, S., Xu, J., Sezer, E.C., Gauriar, P., Iyer, R.K.: Non-control-data attacks are realistic threats. In: Proceedings of the 14th Conference on USENIX Security Symposium, p. 12 (2005)
31. Chinchani, R., Iyer, A., Jayaraman, B., Upadhyaya, S.: Archerr: Runtime environment driven program safety. In: 9th European Symposium on Research in Computer Security, Sophia Antipolis (2004)
32. Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Efficiently computing static single assignment form and the control dependence graph (1991)
33. Foster, J.S., Fähndrich, M., Aiken, A.: A theory of type qualifiers. In: PLDI 1999: Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation, pp. 192–203. ACM, New York (1999)
34. Foster, J.S., Terauchi, T., Aiken, A.: Flow-sensitive type qualifiers. In: PLDI 2002: Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation, Berlin, Germany, pp. 1–12 (2002)
35. Godefroid, P., Klarlund, N., Sen, K.: Dart: directed automated random testing. In: PLDI 2005: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 213–223 (2005)
36. Lattner, C.: LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL (December 2002), http://llvm.cs.uiuc.edu
37. Lattner, C., Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In: Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO 2004), Palo Alto, California (March 2004)
38. Molnar, D., Li, X.C., Wagner, D.A.: Dynamic test generation to find integer bugs in x86 binary linux programs. In: Proceedings of the 18th USENIX Security Symposium (2009)
39. Sen, K., Marinov, D., Agha, G.: Cute: a concolic unit testing engine for c. In: ESEC/FSE-13: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 263–272 (2005)
40. Sotirov, A.: Heap feng shui in javascript. In: Proceedings of Blackhat Europe (2007)
41. 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, San Diego, CA (February 2009)
42. Weiser, M.: Program slicing. In: Proceedings of the 5th International Conference on Software Engineering (1981)

A Theory of Runtime Enforcement, with Results

Jay Ligatti and Srikar Reddy

University of South Florida
Department of Computer Science and Engineering
{ligatti,sreddy4}@cse.usf.edu

Abstract. This paper presents a theory of runtime enforcement based on mechanism models called MRAs (Mandatory Results Automata). MRAs can monitor and transform security-relevant actions and their results. Because previous work could not model monitors transforming results, MRAs capture realistic behaviors outside the scope of previous models. MRAs also have a simple but realistic operational semantics that makes it straightforward to define concrete MRAs. Moreover, the definitions of policies and enforcement with MRAs are significantly simpler and more expressive than those of previous models. Putting all these features together, we argue that MRAs make good general models of runtime mechanisms, upon which a theory of runtime enforcement can be based. We develop some enforceability theory by characterizing the policies MRAs can and cannot enforce.

Keywords: Security models, enforceability theory.

1 Introduction

Runtime enforcement mechanisms work by monitoring untrusted applications, to ensure that those applications obey desired policies. Runtime mechanisms, which are often called runtime/security/program *monitors*, are quite popular and can be seen in operating systems, web browsers, spam filters, intrusion-detection systems, firewalls, access-control systems, stack inspection, etc. Despite their popularity and some initial efforts at modeling monitors formally, we lack satisfactory models of monitors in general, which prevents us from developing an accurate and effective theory of runtime enforcement.

1.1 Related Work

It has been difficult to model runtime mechanisms generally. Most models (e.g., [16,18,11,9,8,1,5]) are based on *truncation automata* [16,12], which can only respond to policy violations by immediately halting the application being monitored (i.e., the *target* application). This constraint simplifies analyses but sacrifices generality. For example, real runtime mechanisms often enforce policies that require the mechanisms to perform "remedial" actions, like popping up a window to confirm dangerous events with the user before they occur (to confirm a web-browser connection with a third-party site, to warn the user before

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 87–100, 2010.

downloading executable email attachments, etc). Although real mechanisms can perform these remedial actions, models based on truncation automata cannot—at the point where the target attempts to perform a dangerous action, truncation automata must immediately halt the target.

To address the limitations of truncation automata, in earlier work we proposed edit automata, models of monitors that can respond to dangerous actions by quietly suppressing them or by inserting other actions [12]. By inserting and suppressing actions, edit automata capture the practical ability of runtime mechanisms to *transform* invalid executions into valid executions, rather than the ability of truncation automata to only *recognize* and halt invalid executions. Edit automata have served as the basis for additional studies of runtime enforcement (e.g., [19,17,4]).

Unfortunately, while truncation automata are too limited to serve as general models of runtime mechanisms, edit automata are too powerful. The edit-automata model assumes monitors can predetermine the results of all actions without executing them, which enables edit automata to safely suppress any action. However, this assumption that monitors can predetermine the result of any action is impractical because the results of many actions are uncomputable, nondeterministic, and/or cannot tractably be predicted by a monitor (e.g., actions that return data in a network buffer, the cloud cover as read by a weather sensor, or spontaneous user input). Put another way, the edit-automata model assumes monitors can buffer—without executing—an unbounded number of target-application actions, but such buffering is impractical because applications typically require results for actions before producing new actions. For example, the echo program `x=input(); output(x)` cannot produce its second action until receiving a result, which is unpredictable, for the first. Because the echo program invokes an action that edit automata cannot suppress (due to its result being unpredictable), this simple program, and any others whose actions may not return predictable results, are outside the edit-automata model.

1.2 Contributions

This paper presents a theory of runtime enforcement based on mechanism models called MRAs (Mandatory Results Automata). Their name alludes to the requirement that, unlike edit automata, MRAs are obligated to return a result to the target application before seeing the next action it wishes to execute. In the MRA model, results of actions may or may not be predeterminable.

Conceptually, we wish to secure a system organized as in Figure 1a, with an application producing actions, and for every action produced, the underlying executing system (e.g., an operating system, virtual machine, or CPU) returning a result to the target application. Results may be exceptions or void or unit values, so all actions can be considered to produce results. For simplicity, this paper assumes all actions are synchronous; after the application produces an action a, it cannot produce another action until receiving a result for a. In contrast, the edit-automata model can be viewed as one in which all actions

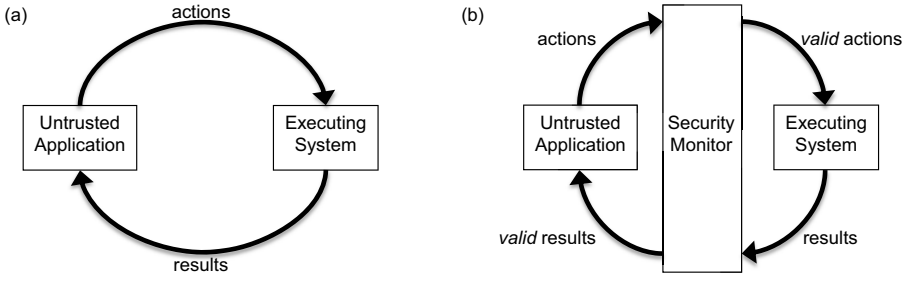


Fig. 1. In (a), an untrusted application executes actions on a system and receives results for those actions. In (b), a security monitor interposes on, and enforces the validity of, the actions executed and the results returned.

are fully asynchronous (because edit automata can buffer, without executing, an unbounded number of actions).

Figure 1b shows how we think of a monitor securing the system of Figure 1a. In Figure 1b, the monitor interposes on and transforms actions and results to ensure that the actions actually executed, and the results actually returned to the application, are valid (i.e., *satisfy* the desired policy). The monitor may or may not be inlined into the target application.

The ability of MRAs to transform *results* of actions is novel among general runtime-enforcement models, as far as we are aware. Yet this ability is crucial for enforcing many security policies, such as privacy, access-control, and information-flow policies, which may require (trusted) mechanisms to sanitize the results of actions before (untrusted) applications access those results. For example, policies may require that system files get hidden when user-level applications retrieve directory listings, that email messages flagged by spam filters do not get returned to clients, or that applications cannot infer secret data based on the results they receive. Because existing frameworks do not model monitors transforming results of actions, one cannot use existing models to specify or reason about enforcing such *result-sanitization policies*.

The semantics of MRAs enables simple and flexible definitions of policies and enforcement—significantly simpler and more flexible than those of previous work. In particular, the definition of executions presented here allows policies to make arbitrary requirements on how monitors must transform actions and results. Consequently, this paper's definition of enforcement does not need an explicit notion of *transparency*, which previous work has considered essential for enforcement [7,9,12]. Transparency constrains mechanisms, forcing them to permit already-valid actions to be executed. The MRA model enables policies to specify strictly more and finer-grained constraints than transparency, thus freeing the definition of enforcement from having to hardcode a transparency requirement.

After defining MRAs and the precise circumstances under which they can be said to enforce policies, this paper briefly characterizes the sets of policies MRAs can enforce soundly, completely, and precisely.

Summary of Contributions. This paper develops a theory of runtime enforcement, in which monitors may transform both actions and results. It contributes:

- A simple but general model of runtime mechanisms called MRAs. MRAs appear to be the first general model of runtime mechanisms that can transform results and enforce result-sanitization policies.
- Definitions of policies and enforcement that, because they can reason about how monitors transform actions and results, are significantly simpler and more expressive than existing definitions.
- A brief analysis of the policies MRAs can enforce soundly, completely, and precisely.

2 Background Definitions and Notation

This section briefly lays out some basic definitions of, and notation for specifying, systems and traces. The definitions and notation presented here are extended versions of definitions and notation in previous work (extended to include results of actions) [2,16,12].

We define a system abstractly, in terms of (1) the actions it can execute to perform computation and (2) the possible results of those actions. The system's interface determines its action set; for example, if the executing system is an operating system then actions would be system calls; if the executing system is a virtual machine then actions would be virtual-machine-code instructions (e.g., bytecode, including calls to API libraries integrated with the virtual machine); and if the executing system is machine hardware then the actions would be machine-code instructions. We use the metavariable A to represent the (nonempty, possibly countably infinite) set of actions on a system and R (disjoint from A) to represent the (nonempty, possibly countably infinite) set of results. An *event* is either an action or a result, and we use E to denote the set of events on a system; $E = A \cup R$.

An *execution* or *trace* is a possibly infinite sequence of events; it is the sequence of events that occur during a run of a monitored application and executing system. Adopting a monitor-centric view of Figure 1b, executions include events related to the monitor (1) inputting an action from the target, (2) outputting an action to the executing system, (3) inputting a result from the executing system, and (4) outputting a result to the target. To be explicit about exactly how a monitor is behaving, we subscript every event in an execution with i or o to indicate whether the monitor has input or output that event. When writing executions, we separate events by semicolons. For example, an execution could be:

$$\texttt{shutdown}_i \,;\, \texttt{popupConfirm}_o \,;\, \texttt{OK}_i \,;\, \texttt{shutdown}_o$$

This execution represents the sequence of events in which an application attempts to execute a shutdown action (so that action gets input to the monitor), to which the monitor responds by outputting a window-popup action that, when executed (e.g., by an operating system), confirms the shutdown with the user.

The user OKs the shutdown, so an `OK` result gets input to the monitor, allowing the monitor to then output the shutdown action after all. This example illustrates the alternating input-output nature of monitors that arises from their role as event transformers [12].

The set of all well-formed, finite-length executions on a system with event set E is E^*; the set of all well-formed, infinite-length executions is E^ω; and $E^\infty = E^* \cup E^\omega$. The special symbol $\cdot$ refers to the empty execution, that is, an execution in which no events occur. In general, we use $\cdot$ to refer to an absence of events; at times we use $\cdot$ to denote the absence of a single action or result. The metavariable e ranges over events, a over actions, r over results, x over executions, and $\mathcal{X}$ over sets of executions (i.e., subsets of E^∞). Sometimes it will also be convenient to use α to refer to a "potential action", that is, either $\cdot$ or an action. Similarly, ρ ranges over $\{\cdot\} \cup R$. The notation $x; x'$ denotes concatenation of two executions x and x', the result of which must be a well-formed execution (in E^∞). Finally, when x is a finite prefix of x' we write $x \preceq x'$.

3 Mandatory Results Automata

We model monitors that behave as in Figure 1b as MRAs.

3.1 Definition of MRAs

An MRA M is a tuple (E, Q, q_0, δ), where E is the event set over which M operates, Q is the finite or countably infinite set of possible states of M, q_0 is M's initial state, and δ is a (deterministic) transition function of the form $\delta : Q \times E \to Q \times E$, which takes M's current state and an event being input to M (either an action the target is attempting to execute or a result the underlying system has produced) and returns a new state for M and an event to be output from M (either an action to be executed on the underlying system or a result to be returned to the target). In contrast to earlier work [12], we do not require δ to be decidable; δ may not halt on some inputs. This ability of MRAs to diverge accurately models the abilities of real runtime mechanisms.

We call $\begin{smallmatrix}\alpha_i\\ \rho_o\end{smallmatrix}\Big|q\Big|\begin{smallmatrix}\alpha_o\\ \rho_i\end{smallmatrix}$ a *configuration* of MRA M, where q is M's current state, α_i is either $\cdot$ or the action being input to M (by the target program), α_o is either $\cdot$ or the action being output by M (to the executing system), ρ_i is either $\cdot$ or the result being input to M (by the executing system), and ρ_o is either $\cdot$ or the result being output by M (to the target program). Because MRAs process events one at a time, at most one of α_i, α_o, ρ_i, and ρ_o will ever be nonempty. Our notation for writing configurations mimics the graphic representation of monitors' inputs and outputs in Figure 1b.

We do not bother writing dots in configurations, so $\begin{smallmatrix}a\\ \cdot\end{smallmatrix}\Big|q\Big|\begin{smallmatrix}\cdot\\ \cdot\end{smallmatrix}$ is the same as $\begin{smallmatrix}a\\ \cdot\end{smallmatrix}\Big|q\Big|\begin{smallmatrix}\cdot\\ \cdot\end{smallmatrix}$. The starting configuration of an MRA is $\Big|q_0\Big|$ because the monitor begins executing in its initial state with no events yet input or output.

$$\frac{next_T = a}{{}_{\rho}\left|q\right| \xrightarrow{a_i} {}^{a}\left|q\right|} \textit{(Input-Action)} \qquad \frac{next_S = r}{\left|q\right|^{a} \xrightarrow{r_i} \left|q\right|_{r}} \textit{(Input-Result)}$$

$$\frac{\delta(q,a) = (q',a')}{{}^{a}\left|q\right| \xrightarrow{a'_o} \left|q'\right|^{a'}} \textit{(Output-Act-for-Act)} \qquad \frac{\delta(q,r) = (q',a)}{\left|q\right|_{r} \xrightarrow{a_o} \left|q'\right|^{a}} \textit{(Output-Act-for-Res)}$$

$$\frac{\delta(q,a) = (q',r)}{{}^{a}\left|q\right| \xrightarrow{r_o} {}_{r}\left|q'\right|} \textit{(Output-Res-for-Act)} \qquad \frac{\delta(q,r) = (q',r')}{\left|q\right|_{r} \xrightarrow{r'_o} {}_{r'}\left|q'\right|} \textit{(Output-Res-for-Res)}$$

Fig. 2. Single-step semantics of mandatory results automata

We define the operational semantics of MRAs with a labeled single-step judgment whose form is $C \xrightarrow{e}_M C'$. This judgment indicates that MRA M takes a single step from configuration C to configuration C' while extending the current trace by event e (which will be tagged as either an input or output event). Because M will always be clear from context, we henceforth omit it from the judgment.

The definition of MRAs' single-step semantics appears in Figure 2. Six inference rules define all possible MRA transitions:

1. *Input-Action* enables the MRA to receive a new input action from the target ($next_T$ is the next action generated by the target). Because ρ ranges over $\{\cdot\} \cup R$, the MRA can receive a new input action when in its initial configuration $\left|q_0\right|$ or when in a configuration of the form ${}_{r}\left|q\right|$ (in which case the MRA has most recently returned a result r to the target, so it is ready for another input action).
2. *Output-Act-for-Act* enables the MRA, immediately after inputting action a, to output a possibly different action a'.
3. *Output-Res-for-Act* enables the MRA, immediately after inputting action a, to return a result r for a to the target.
4. *Input-Result* enables the MRA to receive a new input result r for its most recent output action a ($next_S$ is the next result generated by the system).
5. *Output-Act-for-Res* enables the MRA, immediately after inputting result r, to output another action a.
6. *Output-Res-for-Res* enables the MRA, immediately after inputting result r, to return a possibly different result r' to the target for the action it most recently tried to execute.

Although many alternatives exist for defining MRAs' semantics (e.g., process calculi and other deductive systems, some of which can be compressed into four inference rules), we carefully selected the rules in Figure 2 based on their

simplicity—not just in the rules themselves but also in the transition functions of MRAs that step according to those rules.

Several observations about the operational semantics:

- MRAs can "accept" an input action a by outputting it (with *Output-Act-for-Act*), receiving a result r for a (with *Input-Result*), and then returning r to the application (with *Output-Res-for-Res*).
- MRAs can "halt" an application by outputting an action like `exit`, if the underlying system can execute such an action, or by entering an infinite loop in its transition function to block additional inputs and outputs from being made.
- MRAs are indeed obligated to return results to applications before inputting new actions. No transitions allow an MRA to input another action until it has discharged the last by returning a result for it.
- MRAs can avoid or postpone executing dangerous actions while allowing the target to continue executing (with *Output-Res-for-Act*). For example, an MRA could avoid executing a dangerous port-open input action by outputting an error-code or exception result in response. Alternatively, the MRA could quietly postpone executing the port-open action by immediately outputting a `void` result and then observing how the target uses the port; if the target uses the port securely then the MRA could output the original port-open action followed by the secure port-use action(s) (with *Output-Act-for-Act* and *Output-Act-for-Res*). By postponing (i.e., buffering) dangerous actions until they are known to be secure, MRAs can operate as edit automata; however, such buffering is only possible when the valid results of buffered actions are predictable (such as `void` results of some port-open actions).
- We make no assumptions about whether and how the executing system generates results for actions; the executing system may produce results nondeterministically or through uncomputable means (e.g., by reading a weather sensor or spontaneous keyboard input). This design captures the reality that monitors can only determine the results of many actions (e.g., `readFile`, or `getUserInput`) by having the system actually execute those actions. Hence, the *Input-Result* transition, and the single-step relation for MRAs in general, may be nondeterministic. Similarly, MRAs have no knowledge of whether and how the target generates actions, so the *Input-Action* transition may be nondeterministic as well.

These observations, and the semantics of MRAs in general, match our understanding of how real program monitors behave. For example, in the Polymer enforcement system [3], policies can make *insertion suggestions* to output arbitrary actions in response to an input action, can make *exception* or *replacement suggestions* to output an arbitrary result for the most recent input action, can monitor results of actions, and must return a result for the most recent input action before inputting another. PSLang and LoPSiL policies, and aspects in many languages (e.g., AspectJ), behave similarly [7,14,10].

Limitations. Nonetheless, because MRAs are models, some gaps do exist between the possible behaviors of MRAs and what real monitors can do in practice. MRAs share two standard limitations with other general runtime-enforcement models: (1) MRAs can interpose on and make decisions about *all* security-relevant actions and results, but in practice some events may be imperceptible to the monitor (e.g., monitoring every "clock-tick" action is possible in our model but impractical); this is a problem of *complete mediation* [15], and (2) by executing transition functions, MRAs may delay the processing of time-sensitive events, which prevents MRAs from enforcing some time-sensitive policies (this issue is inherent in runtime monitoring). Besides these standard limitations, MRAs have another: for simplicity in this paper, MRAs treat all actions as synchronous (i.e., they finish processing, and return a result for, one input action before inputting another). This limitation prevents MRAs from effectively monitoring applications whose correctness depends on some security-relevant action(s) being asynchronous. However, as mentioned in Section 1.2, the edit-automata model already provides a semantics for monitoring asynchronous actions.

3.2 Example MRAs

We next consider a couple example MRAs exhibiting simple, everyday sorts of behaviors found in practical monitors. The behaviors are so simple that they may seem trivial; nonetheless, the behaviors are outside existing runtime-enforcement models because they involve monitors acting on unpredictable results of actions (something neither truncation nor edit automata can do).

Example 1: Spam-Filtering MRA. This MRA M sanitizes the results of `getMessages` actions to filter out spam emails. M's state consists of a boolean flag indicating whether M is in the process of obtaining email messages; M begins in state 0. M's transition function δ is:

$$\delta(q, e) = \begin{cases} (0, e) & \text{if } q = 0 \text{ and } e \neq \texttt{getMessages} \\ (1, e) & \text{if } q = 0 \text{ and } e = \texttt{getMessages} \\ (0, filter(e)) & \text{if } q = 1 \end{cases}$$

That is, M outputs its inputs verbatim and does not change its state as long as it does not input a `getMessages` action. When M does input `getMessages`, it sets its boolean flag and allows `getMessages` to execute. If M then inputs a result r for `getMessages`, it outputs the spam-filtered version of r and returns to its initial state. With similar techniques, M could sanitize results in other ways (e.g., to remove system files from directory listings).

Example 2: Dangerous-Action-Confirming MRA. Our second example MRA pops up a window to confirm a dangerous action d with the user before allowing d to execute. We assume d has a default return value r, which must be returned when the user decides not to allow d to execute (r would typically be a

`null` pointer or a value indicating an exception). We also assume a `popupConfirm` action that works like a `JOptionPane.showConfirmDialog` method in Java, returning either an `OK` or `cancel` result. M uses a boolean flag, again initially set to 0, for its state, and the following transition function.

$$\delta(q,e) = \begin{cases} (0,e) & \text{if } q = 0 \text{ and } e \neq d \\ (1,\texttt{popupConfirm}) & \text{if } q = 0 \text{ and } e = d \\ (0,r) & \text{if } q = 1 \text{ and } e = \texttt{cancel} \\ (0,d) & \text{if } q = 1 \text{ and } e = \texttt{OK} \end{cases}$$

This function works as expected: M outputs non-d input events verbatim. Once M inputs a d action, it outputs a `popupConfirm` action and waits for a result. If the user cancels the execution of d, M outputs result r; otherwise it outputs action d.

Summary: Because of the simplicity in MRAs' operational semantics, and in concrete MRA transition functions, plus the fact that MRA behaviors match our understanding of the essential behaviors of real runtime monitors, we believe MRAs serve as a good basis for developing a theory of runtime enforcement.

3.3 Generalizing the Operational Semantics

Before we can formally define what it means for an MRA to enforce a policy, we need to generalize the single-step semantics to account for multiple steps. First, we define the (finite) multi-step relation, with judgment form $C \xrightarrow{x}{}^{*} C'$, in the standard way as the reflexive, transitive closure of the single-step relation. The trace above the arrow in the multi-step judgment gets built by concatenating, in order, every event labeled in the single-step judgments. Hence, $C \xrightarrow{x}{}^{*} C'$ means that the MRA builds execution x while transitioning, using any finite number of single steps, from configuration C to configuration C'.

We also define a judgment $M \Downarrow x$ to mean that MRA M, when its input events match the sequence of input events in x, in total *produces* the possibly infinite-length trace x. To define $M \Downarrow x$ formally, there are two cases to consider: First, when $x \in E^{\omega}$, $M \Downarrow x$ iff for all prefixes x' of x, there exists an M-configuration C such that $C_0 \xrightarrow{x'}{}^{*} C$ (where C_0 is M's initial configuration). Second, when $x \in E^{*}$, $M \Downarrow x$ iff there exists an M-configuration C such that (1) $C_0 \xrightarrow{x}{}^{*} C$ and (2) if x ends with an input event then M never transitions from C (otherwise, x would not be the *entire* trace produced on x's input events).

4 MRA-Based Enforcement

This section defines what it means for an MRA to enforce a policy.

4.1 Policies and Properties

A *policy* is a predicate on sets of executions [16]; a set of executions $\mathcal{X} \subseteq E^{\infty}$ satisfies policy P iff $P(\mathcal{X})$. Some policies are also *properties*. Policy P

is a property iff there exists a predicate $\hat{P}$ over E^∞ such that $\forall \mathcal{X} \subseteq E^\infty : \Big(P(\mathcal{X}) \iff \forall x \in \mathcal{X} : \hat{P}(x)\Big)$. There is a one-to-one correspondence between a property P and its predicate $\hat{P}$, so the rest of the paper uses $\hat{P}$ unambiguously to refer to both.

Intuitively, policies can determine whether a set of target executions is valid based on the executions' relationships with one another, but properties cannot take such inter-execution relationships into account. It is sometimes possible for runtime mechanisms to enforce nonproperty policies: a monitor could refer to earlier traces (e.g., saved in files) when deciding how to transform the current execution, or it could monitor multiple executions of a program concurrently [6]. For simplicity, though, this paper analyzes only the properties MRAs can enforce; we assume monitors make decisions about a single execution at a time.

There are two important differences between this paper's definition of policies and the definitions in previous models. The differences arise from the way executions are modeled here: instead of modeling executions as just the actions a monitor outputs, the MRA model also includes (1) output results, and (2) all input events, in executions. Because policies here may take output results into account, they can specify constraints on which results may be returned to targets; policies here may require results to be sanitized. For example, the spam-filtering MRA from Section 3.2 enforces a policy requiring all results of `getMessages` actions to be filtered (this policy is a property because it is satisfied iff *every* execution in a set $\mathcal{X}$ has exactly zero spam-containing results of `getMessages` actions).

Moreover, because policies in the MRA model can take input events into account, **policies here can require *arbitrary* relationships to hold between input and output events**. For example, a property $\hat{P}$ could be dissatisfied by execution $\texttt{shutdown}_i$ (i.e., $\neg\hat{P}(\texttt{shutdown}_i)$) but be satisfied by $\texttt{shutdown}_i\,;\texttt{popupConfirm}_o$. To enforce this $\hat{P}$, an MRA may have no choice but to output `popupConfirm` upon inputting a `shutdown` action. Policies in previous models (e.g., truncation and edit automata) could not specify such relationships between input and output events because the policies were predicates over output executions only. The only relationship allowed between input and output events in previous models was *transparency*, which was hardcoded into the definition of enforcement [9,12] and required monitors to output valid inputs unchanged. Transparency can be encoded in policies in the MRA model (by defining policies to be satisfied only by executions in which valid inputs get output unchanged), but **policies here are strictly more expressive than transparency** because they can specify arbitrary input-output relationships. For example, the popup-confirmation policy above specifies a relationship that is outside the scope of transparency (because there is no requirement for `shutdown` to be output unchanged).

4.2 Enforcement

We define enforcement in terms of standard principles of soundness and completeness. MRA M is *sound* with respect to property $\hat{P}$ when M only produces traces satisfying $\hat{P}$; M is *complete* with respect to $\hat{P}$ when it produces all traces satisfying $\hat{P}$; and M is *precise* with respect to $\hat{P}$ when it is sound and complete with respect to $\hat{P}$.

Definition 1. *On a system with event set E, MRA M:*

- **soundly enforces** $\hat{P}$ *iff* $\forall x \in E^\infty : ((M{\Downarrow}x) \implies \hat{P}(x))$,
- **completely enforces** $\hat{P}$ *iff* $\forall x \in E^\infty : (\hat{P}(x) \implies (M{\Downarrow}x))$, *and*
- **precisely enforces** $\hat{P}$ *iff* $\forall x \in E^\infty : ((M{\Downarrow}x) \iff \hat{P}(x))$.

Definition 1 is significantly simpler and more flexible than definitions of enforcement in related work, because it (1) does not hardcode transparency requirements, and (2) defines complete and precise, in addition to sound, enforcement.

For an example of MRA enforcement, we reconsider the dangerous-action-confirming MRA M of Section 3.2 (recall that M pops up a window to get user confirmation before executing action d; if the user cancels execution of d, a result r gets returned to the target in place of executing d). Let us use $e_i; e_o$ as shorthand for any two-event sequence in which a non-d event is input and then immediately output. Then M precisely enforces a property $\hat{P}$ satisfied by exactly those well-formed executions matching the pattern:

$$(e_i; e_o \mid d_i; \texttt{popupConfirm}_o(\texttt{cancel}_i; r_o \mid \texttt{OK}_i; d_o))^\infty \ (d_i; \texttt{popupConfirm}_o)?$$

This pattern exactly characterizes the executions M builds. M outputs its input events verbatim until no additional inputs arrive or a d action is input. Once M inputs a d action, it immediately outputs the `popupConfirm` action. Execution stops at this point if the user never selects whether d should be allowed; the pattern above therefore allows executions to optionally end with $d_i; \texttt{popupConfirm}_o$. However, if M inputs a `cancel` or `OK` result for the `popupConfirm` action, it must output the appropriate event in response (either r or d) and continue executing. Note that this policy disallows executions from just ending with, for example, an `OK` result being input to confirm a d action; the policy requires execution to continue after the `OK` input by allowing d to execute. The policy therefore specifies a non-transparency relationship between input and output events (unrelated to outputting inputs unchanged), which cannot be expressed in previous enforcement models.

4.3 Wanted: Auxiliary Predicates, Dead and Alive

Given a property $\hat{P}$ and a finite execution x, we often find it useful to know whether x can be extended into a valid execution. We introduce two predicates for this purpose: when x can be made valid by extending it with some sequence of events, we say x is *alive*; otherwise, x is *dead*. Formally, $alive_{\hat{P}}(x) \iff (\exists x' \in E^\infty : \hat{P}(x; x'))$ and $dead_{\hat{P}}(x) \iff \neg alive_{\hat{P}}(x)$.

Because $\hat{P}$ will always be clear from context, we omit it as a subscript in future $alive(x)$ and $dead(x)$ judgments. Also, because properties in practice generally have predicates $\hat{P}$ and $alive$ that are decidable over finite-length inputs, and because only considering such properties simplifies the theorems in Section 5, this paper limits its scope to properties with predicates $\hat{P}(x)$ and $alive(x)$ that are decidable over finite x.

5 Analysis of MRA-Enforceable Policies

Theorems 1-3 characterize the properties MRAs can soundly, completely, and precisely enforce. Although space limitations prevent us from getting into any details, interested readers may consult our technical report [13] for discussions and proofs of the theorems.

Notation. The theorems use the notation $\exists(x'; e_i) \preceq x : F$ to mean that there exists a prefix of x having the form $x'; e_i$ such that F holds. Similarly, the notation $\forall(x; e_i) \in E^* : F$ means that F is true for all well-formed finite executions $x; e_i$. We also use uniqueness quantification of the form $\exists_1 e \in E : F$ to mean that there exists exactly one event e in E such that F is true. Finally, we assume conjunction ($\wedge$) binds more tightly than disjunction ($\vee$).

Theorem 1. *Property $\hat{P}$ on a system with event set E can be* **soundly** *enforced by some MRA M iff there exists recursively enumerable predicate R over E^* such that all the following are true.*

1. $R(\cdot)$
2. $\forall(x; e_i) \in E^* : \left(\neg R(x) \ \vee \ \hat{P}(x; e_i) \ \vee \ \exists e' \in E : \begin{pmatrix} R(x; e_i; e'_o) \\ \wedge \ \hat{P}(x; e_i; e'_o) \end{pmatrix}\right)$
3. $\forall x \in E^\omega : \left(\neg\hat{P}(x) \implies \exists(x'; e_i) \preceq x : \neg R(x')\right)$

Theorem 2. *Property $\hat{P}$ on a system with event set E can be* **completely** *enforced by some MRA M iff:*

$$\forall(x; e_i) \in E^* : \begin{pmatrix} \forall e' \in E : dead(x; e_i; e'_o) \\ \vee \ \neg\hat{P}(x; e_i) \ \wedge \ \exists_1 e' \in E : alive(x; e_i; e'_o) \end{pmatrix}$$

To learn which policies MRAs precisely enforce, we can intersect the policies soundly enforceable with the policies completely enforceable, and then simplify the result. Doing so produces Theorem 3.

Theorem 3. *Property $\hat{P}$ on a system with event set E can be* **precisely** *enforced by some MRA M iff all the following are true.*

1. $\hat{P}(\cdot)$
2. $\forall (x; e_i) \in E^* : \left(\begin{array}{l} \neg \hat{P}(x) \\ \vee\ \hat{P}(x; e_i)\ \wedge\ \forall e' \in E : dead(x; e_i; e'_o) \\ \vee\ \neg \hat{P}(x; e_i)\ \wedge\ \exists_1 e' \in E : \hat{P}(x; e_i; e'_o) \\ \qquad\qquad\ \wedge\ \exists_1 e' \in E : alive(x; e_i; e'_o) \end{array} \right)$
3. $\forall x \in E^\omega : \left(\neg \hat{P}(x) \implies \exists (x'; e_i) \preceq x : \neg \hat{P}(x') \right)$

6 Conclusions

We have presented MRAs as general models of runtime enforcement mechanisms. MRAs do not suffer from the primary problems of previous models because they (1) allow monitors to transform actions and results and (2) do not assume that monitors can predetermine results of actions. MRAs are the first general models of runtime enforcement we know of in which result-sanitization policies can be reasoned about, and we have seen some examples of how MRAs with simple transition functions can enforce result-sanitization, and other result-dependent, policies. Also, the definitions of policies and enforcement with MRAs are significantly simpler and more expressive than existing definitions because they allow policies to require arbitrary (including non-transparency) relationships between input and output events. Finally, after defining MRAs and enforcement, we have characterized the policies they soundly, completely, and precisely enforce, so for example, security engineers should never waste effort attempting to use MRA-style mechanisms to precisely enforce policies outside the set defined by Theorem 3.

These contributions, and theories of runtime enforcement in general, are important because they:

- shape how we think about the roles and meanings of policies, mechanisms, and enforcement,
- influence our decisions about how to specify policies and mechanisms (including designs of policy-specification languages),
- enable us to reason about whether specific mechanisms enforce desired policies, and
- improve our understanding of which policies can and cannot be enforced at runtime.

We hope that with continued research, enforceability theory will benefit the security community in similar ways that computability theory has benefited the broader computer-science community.

Acknowledgments

We are grateful for feedback from Lujo Bauer, Egor Dolzhenko, Frank Piessens, and the anonymous reviewers. This research was supported by National Science Foundation grants CNS-0716343 and CNS-0742736.

References

1. Aktug, I., Dam, M., Gurov, D.: Provably correct runtime monitoring. In: Proceedings of the 15th International Symposium on Formal Methods (May 2008)
2. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters 21(4), 181–185 (1985)
3. Bauer, L., Ligatti, J., Walker, D.: Composing expressive runtime security policies. ACM Transactions on Software Engineering and Methodology 18(3), 1–43 (2009)
4. Beauquier, D., Cohen, J., Lanotte, R.: Security policies enforcement using finite edit automata. Electron. Notes Theor. Comput. Sci. 229(3), 19–35 (2009)
5. Dam, M., Jacobs, B., Lundblad, A., Piessens, F.: Security monitor inlining for multithreaded java. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP) (July 2009)
6. Devriese, D., Piessens, F.: Non-interference through secure multi-execution. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 109–124 (May 2010)
7. Erlingsson, Ú.: The Inlined Reference Monitor Approach to Security Policy Enforcement. PhD thesis, Cornell University (January 2004)
8. Fong, P.W.L.: Access control by tracking shallow execution history. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2004)
9. Hamlen, K., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Transactions on Progamming Languages and Systems 28(1), 175–205 (2006)
10. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, p. 327. Springer, Heidelberg (2001)
11. Kim, M., Kannan, S., Lee, I., Sokolsky, O., Viswantathan, M.: Computational analysis of run-time monitoring—fundamentals of Java-MaC. Run-time Verification (June 2002)
12. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security 12(3), 1–41 (2009)
13. Ligatti, J., Reddy, S.: A theory of runtime enforcement, with results. Technical Report USF-CSE-SS-102809, University of South Florida (June 2010), http://www.cse.usf.edu/~ligatti/papers/mra-tr.pdf
14. Ligatti, J., Rickey, B., Saigal, N.: LoPSiL: A location-based policy-specification language. In: International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec) (June 2009)
15. Saltzer, J., Schroeder, M.: The protection of information in computer systems. Proceedings of the IEEE 63(9), 1278–1308 (1975)
16. Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and Systems Security 3(1), 30–50 (2000)
17. Talhi, C., Tawbi, N., Debbabi, M.: Execution monitoring enforcement under memory-limitation constraints. Information and Computation 206(2-4), 158–184 (2008)
18. Viswanathan, M.: Foundations for the Run-time Analysis of Software Systems. PhD thesis, University of Pennsylvania (2000)
19. Yu, D., Chander, A., Islam, N., Serikov, I.: Javascript instrumentation for browser security. In: Proceedings of the Symposium on Principles of Programming Languages, pp. 237–249 (2007)

Enforcing Secure Object Initialization in Java

Laurent Hubert[1], Thomas Jensen[2], Vincent Monfort[2], and David Pichardie[2]

[1] CNRS/IRISA, France
[2] INRIA Rennes - Bretagne Atlantique/IRISA, France

Abstract. Sun and the CERT recommend for secure Java development to *not allow partially initialized objects to be accessed*. The CERT considers the severity of the risks taken by not following this recommendation as *high*. The solution currently used to enforce object initialization is to implement a coding pattern proposed by Sun, which is not formally checked. We propose a modular type system to formally specify the initialization policy of libraries or programs and a type checker to statically check at load time that all loaded classes respect the policy. This allows to prove the absence of bugs which have allowed some famous privilege escalations in Java. Our experimental results show that our safe default policy allows to prove 91% of classes of `java.lang`, `java.security` and `javax.security` safe without any annotation and by adding 57 simple annotations we proved all classes but four safe. The type system and its soundness theorem have been formalized and machine checked using Coq.

1 Introduction

The initialization of an information system is usually a critical phase where essential defense mechanisms are being installed and a coherent state is being set up. In object-oriented software, granting access to partially initialized objects is consequently a delicate operation that should be avoided or at least closely monitored. Indeed, the CERT recommendation for secure Java development [2] clearly requires to *not allow partially initialized objects to be accessed* (guideline OBJ04-J). The CERT has assessed the risk if this recommendation is not followed and has considered the severity as *high* and the likelihood as *probable*. They consider this recommendation as a first priority on a scale of three levels.

The Java language and the Java Byte Code Verifier (BCV) enforce some properties on object initialization, *e.g.* about the order in which constructors of an object may be executed, but they do not directly enforce the CERT recommendation. Instead, Sun provides a guideline that enforces the recommendation. Conversely, failing to apply this guidelines may silently lead to security breaches. In fact, a famous attack [4] used a partially initialized class loader for privilege elevation.

We propose a twofold solution: (i) a modular type system which allows to express the initialization policy of a library or program, *i.e.* which methods may access partially initialized objects and which may not; and (ii) a type checker,

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 101–115, 2010.

which can be integrated into the BCV, to statically check the program at load time. To validate our approach, we have *formalized* our type system, *machine checked* its soundness proof using the Coq proof assistant, and *experimentally validated* our solution on a large number of classes from Sun's Java Runtime Environment (JRE).

Section 3 overviews object initialization in Java and its impacts on security. Section 4 then informally presents our type system, which is then formally described in Section 5. Section 6 finally presents the experimental results we obtained on Sun's JRE.

2 Related Work

Object initialization has been studied from different points of view. Freund and Mitchell [7] have proposed a type system that formalizes and enforces the initialization properties ensured by the BCV, which are not sufficient to ensure that no partially initialized object is accessed. Unlike local variables, instance fields have a default value (**`null`**, **`false`** or `0`) which may be then replaced by the program. The challenge is then to check that the default value has been replaced before the first access to the field (*e.g.* to ensure that all field reads return a non-null value). This is has been studied in its general form by Fähndrich and Xia [6], and Qi and Myers [9]. Those works are focused on enforcing invariants on fields and finely tracks the different fields of an object. They also try to follow the objects after their construction to have more information on initialized fields. This is an overkill in our context. Unkel and Lam studied another property of object initialization: stationary fields [12]. A field may be stationary if all its reads return the same value. There analysis also track fields of objects and not the different initialization of an object. In contrast to our analysis, they stop to track any object stored into the heap.

Other work have targeted the order in which methods are called. It has been studied in the context of rare events (*e.g.* to detect anomaly, including intrusions). We refer the interested reader to the survey of Chandola *et al.* [3]. They are mainly interested in the order in which methods are called but not about the initialization status of arguments. While we guarantee that a method taking a fully initialized receiver is called after its constructor, this policy cannot be locally expressed with an order on method calls as the methods (constructors) which needs to be called on a object to initialize it depends on the dynamic type of the object.

3 Context Overview

Fig. 1 is an extract of class `ClassLoader` of SUN's JRE as it was before 1997. The security policy which needs to be ensured is that `resolveClass`, a security sensitive method, may be called only if the security check l. 5 has succeeded. To ensure this security property, this code relies on the properties enforced on object initialization by the BCV.

```
1 public abstract class ClassLoader {
2   private ClassLoader parent;
3   protected ClassLoader() {
4     SecurityManager sm = System.getSecurityManager();
5     if (sm != null) {sm.checkCreateClassLoader();}
6     this.parent = ClassLoader.getSystemClassLoader();
7   }
8   protected final native void resolveClass(Class c);
9 }
```

Fig. 1. Extract of the ClassLoader of Sun's JRE

Standard Java Object Construction. In Java, objects are initialized by calling a class-specific constructor which is supposed to establish an invariant on the newly created object. The BCV enforces two properties related to these constructors. These two properties are necessary but, as we shall see, not completely sufficient to avoid security problems due to object initialization.

Property 1. Before accessing an object, (i) a constructor of its dynamic type has been called and (ii) each constructor either calls another constructor of the same class or a constructor of the super-class on the object under construction, except for `java.lang.Object` *which has no super-class.*

This implies that at least one constructor of C and of each super-class of C is called: it is not possible to bypass a level of constructor. To deal with exceptional behaviour during object construction, the BCV enforces another property — concisely described in *The Java Language Specification* [8], Section 12.5, or implied by the type system described in the JSR202 [1]).

Property 2. If one constructor finishes abruptly, then the whole construction of the object finishes abruptly.

Thus, if the construction of an object finishes normally, then all constructors called on this object have finished normally. Failure to implement this verification properly led to a famous attack [4] in which it was exploited that if code such as `try {super();} catch(Throwable e){}` in a constructor is not rejected by the BCV, then malicious classes can create security-critical classes such as class loaders.

Attack on the Class Loader and the Patch from Sun. However, even with these two properties enforced, it is not guaranteed that uninitialized objects cannot be used. In Fig. 1, if the check fails, the method `checkCreateClassLoader` throws an exception and therefore terminates the construction of the object, but the garbage collector then call a `finalize()` method, which is an instance method and has the object to be collected as receiver (cf. Section 12.6 of [8]).

An attacker could code another class that extends `ClassLoader` and has a `finalize()` method. If run in a right-restricted context, *e.g.* an applet, the constructor of `ClassLoader` fails and the garbage collector then call the attacker's

```
1  public abstract class ClassLoader {
2    private volatile boolean initialized;
3    private ClassLoader parent;
4    protected ClassLoader() {
5      SecurityManager sm = System.getSecurityManager();
6      if (sm != null) {sm.checkCreateClassLoader();}
7      this.parent = ClassLoader.getSystemClassLoader();
8      this.initialized = true;}
9    private void check() {
10     if (!initialized) {
11       throw new SecurityException(
12         "ClassLoader␣object␣not␣initialized");}}
13   protected final void resolveClass(Class c){
14     this.check();
15     this.resolveClass0(c);}
16   private native void resolveClass0(Class c);
17 }
```

Fig. 2. Extract of the ClassLoader of Sun's JRE

`finalize` method. The attacker can therefore call the `resolveClass` method on it, bypassing the security check in the constructor and breaking the security of Java.

The initialization policy enforced the BCV is in fact too weak: when a method is called on an object, there is no guarantee that the construction of an object has been successfully run. An ad-hoc solution to this problem is proposed by SUN [11] in its Guideline 4-3 *Defend against partially initialized instances of non-final classes*: adding a special Boolean field to each class for which the developer wants to ensure it has been sufficiently initialized. This field, set to **`false`** by default, should be private and should be set to **`true`** at the end of the constructor. Then, every method that relies on the invariant established by the constructor must test whether this field is set to **`true`** and fail otherwise. If `initialized` is **`true`**, the construction of the object up to the initialization of `initialized` has succeeded. Checking if `initialized` is **`true`** allows to ensure that sensitive code is only executed on classes that have been initialized up to the constructor of the current class. Fig. 2 shows the same extract as in Fig. 1 but with the needed instrumentation (this is the current implementation as of JRE 1.6.0_16).

Although there are some exceptions and some methods are designed to access partially initialized objects (for example to initialize the object), most methods should not access partially initialized objects. Following the remediation solution proposed in the CERT's recommendation or Sun's guideline 4-3, a field should be added to almost every class and most methods should start by checking this field. This is resource consuming and error prone because it relies on the programmer to keep track of what is the semantic invariant, without providing the adequate automated software development tools. It may therefore lead not to

functional bugs but to security breaches, which are harder to detect. In spite of being known since 1997, this pattern is not always correctly applied to all places where it should be. This has lead to security breaches, see *e.g.*, the Secunia Advisory SA10056 [10].

4 The Right Way: A Type System

We propose a twofold solution: first, a way to specify the security policy which is simple and modular, yet more expressive than a single Boolean field; second, a modular type checker, which could be integrated into the BCV, to check that the whole program respects the policy.

4.1 Specifying an Initialization Policy with Annotations

We rely on Java annotations and on one instruction to specify our initialization policy. We herein give the grammar of the annotations we use.

```
V_ANNOT ::= @Init | @Raw | @Raw(CLASS)
R_ANNOT ::= @Pre(V_ANNOT) | @Post(V_ANNOT)
```

We introduce two main annotations: `@Init`, which specifies that a reference can only point to a fully initialized object or the **`null`** constant, and `@Raw`, which specifies that a reference may point to a partially initialized object. A third annotation, `@Raw(CLASS)`, allows to precise that the object may be partially initialized but that all constructors up to and including the constructor of `CLASS` must have been fully executed. *E.g.*, when one checks that `initialized` contains **`true`** in `ClassLoader.resolveClass`, one checks that the receiver has the type `@Raw(ClassLoader)`. The annotations produced by the `V_ANNOT` rule are used for fields, method arguments and return values. In the Java language, instance methods implicitly take another argument: a receiver — reachable through variable **`this`**. We introduce a `@Pre` annotation to specify the type of the receiver at the beginning of the method. Some methods, usually called from constructors, are meant to initialize their receiver. We have therefore added the possibility to express this by adding a `@Post` annotation for the type of the receiver at the end of the method. These annotations take as argument an initialization level produced by the rule `V_ANNOT`.

Fig. 3 shows an example of `@Raw` annotations. Class `Ex1A` has an instance field `f`, a constructor and a getter `getF`. This getter requires the object to be initialized at least up to `Ex1A` as it accesses a field initialized in its constructor. The constructor of `Ex1B` uses this getter, but the object is not yet completely initialized: it has the type `Raw(Ex1A)` as it has finished the constructor of `Ex1A` but not yet the constructor `Ex1B`. If the getter had been annotated with `@Init` it would not have been possible to use it in the constructor of `Ex1B`.

Another part of the security policy is the `SetInit` instruction, which mimics the instruction **`this`**`.initialized = `**`true`** in Sun's guideline. It is implicitly put at the end of every constructor but it can be explicitly placed before. It

```
1 class Ex1A {
2     private Object f;
3     Ex1A(Object o){
4         securityManagerCheck()
5         this.f = o;}
6     @Pre(@Raw(Ex1A))
7     getF(){return this.f;}
8 }
```

```
9 class Ex1B extends Ex1A{
10     Ex1B(){
11         super();
12         ... = this.getF();
13     }
14 }
```

Fig. 3. Motivations for `Raw(CLASS)` annotations

```
1 public C() {
2   ...
3   securityManagerCheck(); // perform dynamic security checks
4   SetInit;                // declare the object initialized up C
5   Global.register(this);  // the object is used with a method
6 }                         // that only accept Raw(C) parameters
```

Fig. 4. An Example with `SetInit`

declares that the current object has completed its initialization up to the current class. Note that the object is not yet considered fully initialized as it might be called as a parent constructor in a subclass. The instruction can be used, as in Fig.4, in a constructor after checking some properties and before calling some other method.

Fig. 5 shows class `ClassLoader` with its policy specification. The policy ensured by the current implementation of Sun is slightly weaker: it does not ensure that the receiver is fully initialized when invoking `resolveClass` but simply checks that the constructor of `ClassLoader` has been fully run. On this example, we can see that the constructor has the annotations `@Pre(@Raw)`, meaning that the receiver may be completely uninitialized at the beginning, and `@Post(@Raw(ClassLoader))`, meaning that, on normal return of the method, at least one constructor for each parent class of `ClassLoader` and a constructor of `ClassLoader` have been fully executed.

We define as default values the most precise type that may be use in each context. This gives a *safe by default* policy and lowers the burden of annotating a program.

- Fields, method parameters and return values are fully initialized objects (written `@Init`).
- Constructors take a receivers uninitialized at the beginning (`@Pre(@Raw)`) and initialized up-to the current class at the end (written `@Post(@Raw(C))` if in the class `C`).
- Other methods take a receiver fully initialized (`@Pre(@Init)`).
- Except for constructors, method receivers have the same type at the end as at beginning of the method (written `@Post(A)` if the method has the annotation `@Pre(A)`).

```
1  public abstract class ClassLoader {
2    @Init private ClassLoader parent;
3    @Pre(@Raw) @Post(@Raw(ClassLoader))
4    protected ClassLoader() {
5      SecurityManager sm = System.getSecurityManager();
6      if (sm != null) {sm.checkCreateClassLoader();}
7      this.parent = ClassLoader.getSystemClassLoader();
8    }
9    @Pre(@Init) @Post(@Init)
10   protected final native void resolveClass(@Init Class c);
11 }
```

Fig. 5. Extract of the ClassLoader of Sun's JRE

If we remove from Fig. 5 the default annotations, we obtain the original code in Fig. 1. It shows that despite choosing the strictest (and safest) initialization policy as default, the annotation burden can be kept low.

4.2 Checking the Initialization Policy

We have chosen static type checking for at least two reasons. Static type checking allows for more performances (except for some rare cases), as the complexity of static type checking is linear in the *code size*, whereas the complexity of dynamic type checking is linear in the *execution time*. Static type checking also improves reliability of the code: if a code passes the type checking, then the code is correct with respect to its policy, whereas the dynamic type checking only ensures the correction of a particular execution.

Reflection in Java allows to retrieve code from the network or to dynamically generates code. Thus, the whole code may not be available before actually executing the program. Instead, code is made available class by class, and checked by the BCV at linking time, before the first execution of each method. As the whole program is not available, the type checking must be modular: there must be enough information in a method to decide if this method is correct and, if an incorrect method is found, there must exist a safe procedure to end the program (usually throwing an exception), *i.e.* it must not be too late.

To a have a modular type checker while keeping our security policy simple, method parameters, respectively return values, need to be contra-variant, respectively co-variant, *i.e.* the policy of the overriding methods needs to be at least as general as the policy of the overridden method. Note that this is not surprising: the same applies in the Java language (although Java imposes the invariance of method parameters instead of the more general contra-variance), and when a method call is found in a method, it allows to rely on the policy of the resolved method (as all the method which may actually be called cannot be known before the whole program is loaded).

$$
\begin{array}{rl}
 & x, y, r \in \mathit{Var} \quad f \in \mathit{Field} \quad e \in \mathit{Exc} \quad i \in \mathcal{L} = \mathbb{N} \\
p \in \mathit{Prog} ::= & \{\, \mathsf{classes} \in \mathcal{P}(\mathit{Class}),\ \mathsf{main} \in \mathit{Class}, \\
 & \quad \mathsf{fields} \in \mathit{Field} \to \mathit{Type},\ \mathsf{lookup} \in \mathit{Class} \to \mathit{Meth} \rightharpoonup \mathit{Meth} \} \\
c \in \mathit{Class} ::= & \{ \mathsf{super} \in \mathit{Class}_\perp,\ \mathsf{methods} \in \mathcal{P}(\mathit{Meth}),\ \mathsf{init} \in \mathit{Meth} \} \\
m \in \mathit{Meth} ::= & \{\, \mathsf{instrs} \in \mathit{Instr}\ \mathit{array},\ \mathsf{handler} \in \mathcal{L} \to \mathit{Exc} \to \mathcal{L}_\perp, \\
 & \quad \mathsf{pre} \in \mathit{Type},\ \mathsf{post} \in \mathit{Type},\ \mathsf{argtype} \in \mathit{Type},\ \mathsf{rettype} \in \mathit{Type} \} \\
\tau \in \mathit{Type} ::= & \mathit{Init} \mid \mathit{Raw}(c) \mid \mathit{Raw}^\perp \\
e \in \mathit{Expr} ::= & \mathit{null} \mid x \mid e.f \\
\mathit{ins} \in \mathit{Instr} ::= & x \leftarrow e \mid x.f \leftarrow y \mid x \leftarrow \mathit{new}\ c(y) \mid \mathit{if}\ (\star)\ \mathit{jmp} \mid \\
 & \mathit{super}(y) \mid x \leftarrow r.m(y) \mid \mathit{return}\ x \mid \mathit{SetInit}
\end{array}
$$

Fig. 6. Language Syntax

5 Formal Study of the Type System

The purpose of this work is to provide a type system that enforces at load time an important security property. The semantic soundness of such mechanism is hence crucial for the global security of the Java platform. In this section, we formally define the type system and prove its soundness with respect to an operational semantics. All the results of this section have been machine-checked with the Coq proof assistant[1].

Syntax. Our language is a simple language in-between Java source and Java bytecode. Our goal was to have a language close enough to the bytecode in order to easily obtain, from the specification, a naive implementation at the bytecode level while keeping a language easy to reason with. It is based on the decompiled language from Demange *et al.* [5] that provides a stack-less representation of Java bytecode programs. Fig. 6 shows the syntax of the language. A program is a record that handles a set of classes, a main class, a type annotation for each field and a lookup operator. This operator is used do determine during a virtual call the method ($p.\mathsf{lookup}\ c\ m$) (if any) that is the first overriding version of a method m in the ancestor classes of the class c. A class is composed of a super class (if any), a set of method and a special constructor method init. A method handles an array of instructions, a handler function such that ($m.\mathsf{handler}\ i\ e$) is the program point (if any) in the method m where the control flows after an exception e has been thrown at point i. Each method handles also four initialization types for the initial value of the variable **this** ($m.\mathsf{pre}$), its final value ($m.\mathsf{post}$), the type of its formal parameter[2] ($m.\mathsf{argtype}$) and the type of its return value ($m.\mathsf{rettype}$). The only expressions are the *null* constant, local variables and field reads. For this analysis, arithmetic needs not to be taken into account. We only manipulate objects. The instructions are the assignment to a local variable or to a field,

[1] The development can be downloaded at
http://www.irisa.fr/celtique/ext/rawtypes/

[2] For the sake of simplicity, each method has a unique formal parameter *arg*.

$$
\begin{array}{rll}
\overline{Exc} \ni \overline{e} & ::= e \mid \bot & \text{(exception flag)} \\
\mathbb{L} \ni l & & \text{(location)} \\
\mathbb{V} \ni v & ::= l \mid null & \text{(value)} \\
\mathbb{M} = Var \to \mathbb{V} \ni \rho & & \text{(local variables)} \\
\mathbb{O} = Class \times Class_{\bot} \times (Field \to \mathbb{V}) \ni o & ::= [c, c_{init}, o] & \text{(object)} \\
\mathbb{H} = \mathbb{L} \to \mathbb{O}_{\bot} \ni \sigma & & \text{(heap)} \\
CS \ni cs & ::= (m, i, l, \rho, r) :: cs \mid \varepsilon & \text{(call stack)} \\
\mathbb{S} = Meth \times \mathcal{L} \times \mathbb{M} \times \mathbb{H} \times CS \times \overline{Exc} \ni st & ::= \langle m, i, \rho, \sigma, cs \rangle_{\overline{e}} & \text{(state)}
\end{array}
$$

Fig. 7. Semantic Domains

object creation (*new*)[3], (non-deterministic) conditional jump, super constructor call, virtual method call, return, and a special instruction that we introduce for explicit object initialization: *SetInit*.

Semantic Domains. Fig. 7 shows the concrete domain used to model the program states. The state is composed of the current method m, the current program point i in m (the index of the next instruction to be executed in $m.\mathsf{instrs}$), a function for local variables, a heap, a call stack and an exception flag. The heap is a partial function which associates to a location an object $[c, c_{init}, o]$ with c its type, c_{init} its current initialization level and o a map from field to value (in the sequel o is sometimes confused with the object itself). An initialization $c_{init} \in Class$ means that each constructors of c_{init} and its super-classes have been called on the object and have returned without abrupt termination. The exception flag is used to handle exceptions: a state $\langle \cdots \rangle_e$ with $e \in Exc$ is reached after an exception e has been thrown. The execution then looks for a handler in the current method and if necessary in the methods of the current call stack. When equal to $\bot$, the flag is omitted (normal state). The call stack records the program points of the pending calls together with their local environments and the variable that will be assigned with the result of the call.

Initialization Types. We can distinguish three different kinds of initialization types. Given a heap σ we define a value type judgment $h \vdash v : \tau$ between values and types with the following rules.

$$
\frac{}{\sigma \vdash null : \tau} \qquad \frac{}{\sigma \vdash l : Raw^{\bot}} \qquad \frac{\sigma(l) = [c_{dyn}, c_{init}, o] \quad \forall c', c_{dyn} \preceq c' \wedge c \preceq c' \Rightarrow c_{init} \preceq c'}{\sigma \vdash l : Raw(c)} \qquad \frac{\sigma(l) = [c, c, o]}{\sigma \vdash l : Init}
$$

The relation $\preceq$ here denotes the reflexive transitive closure of the relation induced by the *super* element of each class. $Raw^{\bot}$ denotes a reference to an object which

[3] Here, the same instruction allocates the object and calls the constructor. At bytecode level this gives raise to two separated instructions in the program (allocation and later constructor invocation) but the intermediate representation generator [5] on which we rely is able to recover such construct.

$$\frac{m.\mathsf{instrs}[i] = x \leftarrow new\ c(y) \quad x \neq this \quad Alloc(\sigma, c, l, \sigma') \quad \sigma' \vdash \rho(y) : c.\mathsf{init.argtype}}{\langle m, i, \rho, \sigma, cs\rangle \Rightarrow \langle c.\mathsf{init}, 0, [\cdot \mapsto null][this \mapsto l][arg \mapsto \rho(y)], \sigma', (m, i, \rho, x) :: cs\rangle}$$

$$\frac{m.\mathsf{instrs}[i] = SetInit \quad m = c.\mathsf{init} \quad \rho(this) = l \quad SetInit(\sigma, c, l, \sigma')}{\langle m, i, \rho, \sigma, cs\rangle \Rightarrow \langle m, i{+}1, \rho, \sigma', cs\rangle}$$

$$\frac{m.\mathsf{instrs}[i] = return\ x \quad \rho(this) = l \quad ((\forall c,\ m \neq c.\mathsf{init}) \Rightarrow \sigma = \sigma') \quad (\forall c,\ m = c.\mathsf{init} \Rightarrow SetInit(\sigma, c, l, \sigma') \wedge x = this)}{\langle m, i, \rho, \sigma, (m', i', \rho', r) :: cs\rangle \Rightarrow \langle m', i'{+}1, \rho'[r \mapsto \rho(x)], \sigma', cs\rangle}$$

Fig. 8. Operational Semantics (excerpt)

may be completely uninitialized (at the very beginning of each constructor). *Init* denotes a reference to an object which has been completely initialized. Between those two "extreme" types, a value may be typed as $Raw(c)$ if at least one constructor of c and of each parent of c has been executed on all objects that may be reference from this value. We can derive from this definition the sub-typing relation $Init \sqsubseteq Raw(c) \sqsubseteq Raw(c') \sqsubseteq Raw^{\perp}$ if $c \preceq c'$. It satisfies the important monotony property

$$\forall \sigma \in \mathbb{H}, \forall v \in \mathbb{V}, \forall \tau_1, \tau_2 \in Type,\ \tau_1 \sqsubseteq \tau_2 \wedge \sigma \vdash v : \tau_1 \Rightarrow \sigma \vdash v : \tau_2$$

Note that the sub-typing judgment is disconnected from the static type of object. In a first approach, we could expect to manipulate a pair (c, τ) with c the static type of an object and τ its initialization type and consider equivalent both types $(c, Raw(c))$ and $(c, Init)$. Such a choice would however impact deeply on the standard dynamic mechanism of a JVM: each dynamic cast from A to B (or a virtual call on a receiver) would requires to check that an object has not only an initialization level set up to A but also set up to B.

Operational Semantics. We define the operational semantics of our language as a small-step transition relation over program states. A fixed program p is

Expression typing

$$\frac{}{L \vdash e.f : (p.\mathsf{fields}\ f)} \qquad \frac{}{L \vdash x : L(x)} \qquad \frac{}{L \vdash null : Init}$$

Instruction typing

$$\frac{L \vdash e : \tau \quad x \neq this}{m \vdash x \leftarrow e : L \rightarrow L[x \mapsto \tau]} \qquad \frac{L(y) \sqsubseteq (p.\mathsf{fields}\ f)}{m \vdash x.f \leftarrow y : L \rightarrow L} \qquad \frac{}{\Gamma, m \vdash if \star\ jmp : L \rightarrow L}$$

$$\frac{L(this) \sqsubseteq m.\mathsf{post} \quad L(x) \sqsubseteq m.\mathsf{rettype} \quad (\forall c,\ m = c.\mathsf{init} \Rightarrow L(this) \sqsubseteq Raw(c.\mathsf{super}))}{m \vdash return\ x : L \rightarrow L}$$

$$\frac{L(y) \sqsubseteq c.\mathsf{init.argtype}}{m \vdash x \leftarrow new\ c(y) : L \rightarrow L[x \mapsto Init]} \qquad \frac{c' = c.super \quad L(y) \sqsubseteq c'.\mathsf{init.argtype}}{c.\mathsf{init} \vdash super(y) : L \rightarrow L[this \mapsto Raw(c')]}$$

$$\frac{L(r) \sqsubseteq m.\mathsf{pre} \quad L(y) \sqsubseteq m.\mathsf{argtype}}{m \vdash x \leftarrow r.m'(y) : L \rightarrow L[r \mapsto m.\mathsf{post}][x \mapsto m.\mathsf{rettype}]} \qquad \frac{L(this) \sqsubseteq Raw(c.\mathsf{super})}{c.\mathsf{init} \vdash SetInit : L \rightarrow L}$$

Fig. 9. Flow sensitive type system

implicit in the rest of this section. Fig. 8 presents some selected rules for this relation. The rule for the *new* instruction includes both the allocation and the call to the constructor. We use the auxiliary predicate $Alloc(\sigma, c, l, \sigma')$ which allocate a fresh location l in heap σ with type c, initialization type equals to $\perp$ and all fields set equal to *null*. The constraint $\sigma' \vdash \rho(y) : c.\mathsf{init.argtype}$ explicitly asks the caller of the constructor to give a correct argument with respect to the policy of the constructor. Each call rules of the semantics have similar constraints. The execution is hence stuck when an attempt is made to call a method with badly typed parameters. The *SetInit* instruction updates the initialization level of the object in *this*. It relies on the predicate $SetInit(\sigma, c, l, \sigma')$ which specifies that σ' is a copy of σ where the object at location l has now the initialization tag set to c if the previous initialization was $c.super$. It forces the current object (**this**) to be considered as initialized up to the current class (*i.e.* as if the constructor of the current class had returned, but not necessarily the constructors of the subsequent classes). This may be used in the constructor, once all fields that need to be initialized have been initialized and if some method requiring a non-raw object needs to be called. Note that this instruction is really sensitive: using this instruction too early in a constructor may break the security of the application. The *return* instruction uses the same predicate when invoked in a constructor. For convenience we requires each constructor to end with a *return this* instruction.

Typing Judgment. Each instruction *ins* of a method m is attached a typing rule (given in Fig. 5) $m \vdash ins : L \to L'$ that constraint the type of variable before (L) and after (L') the execution of *ins*.

Definition 1 (Well-typed Method). *A method m is* well-typed *if there exists flow sensitive variable types $L \in \mathcal{L} \to Var \to Type$ such that*

- $m.\mathsf{pre} \sqsubseteq L(0, this)$ *and* $m.\mathsf{argtype} \sqsubseteq L(0, arg)$,
- *for all instruction ins at point i in m and every successor j of i, there exists a map of variable types $L' \in Var \to Type$ such that $L' \sqsubseteq L(j)$ and the typing judgment $m \vdash ins : L(i) \to L'$ holds. If i is in the handler j of an exception e (*i.e $(m.\mathsf{handler}\ i\ e = j)$*) then $L(i) \sqsubseteq L(j)$.*

The typability of a method can be decided by turning the set of typing rules into a standard dataflow problem. The approach is standard [7] and not formalized here.

Definition 2 (Well-typed Program). *A program p is* well-typed *if all its methods are well-typed and the following constraints holds:*

1. *for every method m that is overridden by a method m' (*i.e *there exists c, such that $(p.\mathsf{lookup}\ c\ m = m')$),*
$$m.\mathsf{pre} \sqsubseteq m'.\mathsf{pre} \ \wedge m.\mathsf{argtype} \sqsubseteq m'.\mathsf{argtype} \wedge$$
$$m.\mathsf{post} \sqsupseteq m'.\mathsf{post} \wedge\ m.\mathsf{rettype} \sqsupseteq m'.\mathsf{rettype}$$
2. *in each method, every first point, jump target and handler point contain an instruction and every instruction (except return) has a next instruction,*

3. the default constructor $c.\mathsf{init}$ *of each class* c *is unique.*

In this definition only point 1 is really specific to the current type system. The other points are necessary to established the progress theorem of the next section.

Type Soundness. We rely on an auxiliary notion of well-formed states that capture the semantics constraints enforce by the type system. A state $\langle m, i, \rho, \sigma, cs\rangle$ is *well-formed* (wf) if there exists a type annotation $L_p \in (Meth \times \mathcal{L}) \rightarrow (Var \rightarrow Type)$ such that

$$\begin{array}{ll} \forall l \in \mathcal{L}, \forall o \in \mathbb{O}, \sigma(l) = o \Rightarrow \sigma \vdash o(f) : (p.\mathsf{fields}\ f) & \text{(wf. heap)} \\ \forall x \in Var, \sigma \vdash \rho(x) : L_p[m, i](x) & \text{(wf. local variables)} \\ \forall (m', i', \rho', r) \in cs,\ \forall x, \sigma \vdash \rho'(x) : L_p[m', i'](x) & \text{(wf. call stack)} \end{array}$$

Given a well-typed program p we then establish two key theorems. First, any valid transition from a well-formed state leads to another well-formed state (*preservation*) and then, from every well-formed state there exists at least a transition (*progress*). As a consequence we can establish that starting from an initial state (which is always well-formed) the execution is never stuck, except on final configuration. This ensures that all initialization constraints given in the operational semantics are satisfied without requiring any dynamic verification.

Limitations. The proposed language has some limitations compared to the Java (bytecode) language. Static fields and arithmetic have not been introduced but are handled by our implementation and do not add particular difficulties. Arrays have not been introduced in the language neither. Our implementation conservatively handles arrays by allowing only writes of *Init* references in arrays. Although this approach seems correct it has not been proved and it is not flexible enough (cf. Section 6). Multi-threading as also been left out of the current formalization but we conjecture the soundness result still holds with respect to the Java Memory Model because of the flow insensitive abstraction made on the heap. As for the BCV, native methods may brake the type system. It is their responsibility to respect the policy expressed in the program.

6 A Case Study: Sun's JRE

In order to show that our type system allows to verify legacy code with only a few annotations, we implemented a standalone prototype, handling the full Java bytecode, and we tested all classes of packages `java.lang`, `java.security` and `javax.security` of the JRE1.6.0_20.

348 classes out of 381 were proven safe *w.r.t.* the default policy without any modification. By either specifying the actual policy when the default policy was too strict, or by adding cast instructions (see below) when the type system was not precise enough, we were able to verify 377 classes, that is to say 99% of classes. We discuss below the 4 remaining classes that are not yet proven correct

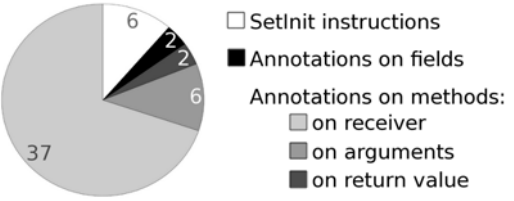


Fig. 10. Distribution of the 47 annotations and 6 instructions added to successfully type the three packages of the JRE

by our analysis. The modifications represent only 55 source lines of code out of 131,486 for the three packages studied. Moreover most code modifications are to express the actual initialization policy, which means existing code can be proven safe. Only 45 methods out of 3,859 (1.1%) and 2 fields out of 1,524 were annotated. Last but not least, the execution of the type checker takes less than 20 seconds for the packages studied.

Adapting the security policy. Fig. 10 details the annotations and the `SetInit` added to specify the security policy. In the runtime library, a usual pattern consists in calling methods that initialize fields during construction of the object. In that case, a simple annotation `@Pre(@Raw(super(C)))` on methods of class `C` is necessary. These cases represent the majority of the 37 annotations on method receivers. 6 annotations on method arguments are used, notably for some methods of `java.lang.SecurityManager` which check permissions on an object during its initialization. The instruction `SetInit` is used when a constructor initializes all the fields of the receiver and then call methods on the receiver that are not part of the initialization. In that case the method called need at least a `Raw(C)` level of initialization and the `SetInit` instruction allows to express that the constructor finished the minimum initialization of the receiver. Only 6 `SetInit` intructions are necessary.

Cast instructions. Such a static and modular type checking introduces some necessary loss of precision — which cannot be completely avoided because of computability issues. To be able to use our type system on legacy code without deep modifications, we introduce two dynamic cast operators: `(Init)` and `(Raw)`. The instruction `y = (Init)x;` allows to dynamically check that `x` points to a fully initialized object: if the object is fully initialized, then this is a simple assignation to `y`, otherwise it throws an exception. As explained in Section 3, the invariant needed is often weaker and the correctness of a method may only need a $Raw(c)$ reference. `y = (Raw(C))x` dynamically checks that `x` points to an object which is initialized up to the constructor of class `C`.

Only 4 cast instructions are necessary. There are needed in two particular cases. First, when a field must be annotated, but annotation on fields were only necessary on two fields — they imply the use of 3 `(Init)` cast instructions. The second case is on a receiver in a `finalize()` method that checks that some

fields are initialized, thereby checking that the object was `Raw(C)` but the type system could not infer this information. The later case implies to use the unique `(Raw(C))` instruction added.

Remaining classes. Finally, only 4 classes are not well-typed after the previous modifications. Indeed the compiler generates some code to compile inner classes and part of this code needs annotations in 3 classes. These cases could be handled by doing significant changes on the code, by adding new annotations dedicated to inner classes or by annotating directly the bytecode. The one class remaining is not typable because of the limited precision of our analysis on arrays: one can only store `@Init` values in arrays. To check this later class, our type system needs to be extended to handle arrays more precisely but this is left for future work.

Special case of finalize methods. As previously exposed, `finalize()` methods may be invoked on a completely uninitialized receiver. Therefore, we study the case of `finalize()` methods in the packages `java.*` and `javax.*`. In the classes of those packages there are 28 `finalize()` methods and only 12 succeed to be well-typed with our default annotation values. These are either empty or do not use their receiver at all. For the last 16 classes, the necessary modifications are either the use of cast instructions when the code's logic guarantees the success of cast, or the addition of `@Pre(@Raw)` annotations on methods called on the receiver. In that case, it is important to verify that the code of any called method is defensive enough. Therefore, the type system forced us to pay attention to the cases that could lead to security breaches or crashes at run time for `finalize()` methods. After a meticulous checking of the code we added the necessary annotations and cast instructions that allowed to verify the 28 classes.

7 Conclusion and Future Work

We have proposed herein a solution to enforce a secure initialization of objects in Java. The solution is composed of a modular type system which allows to manage uninitialized objects safely when necessary, and of a modular type checker which can be integrated into the BCV to statically check a program at load time. The type system has been formalized and proved sound, and the type-checker prototype has been experimentally validated on more than 300 classes of the Java runtime library.

The experimental results point out that our default annotations minimize the user intervention needed to type a program and allows to focus on the few classes where the security policy needs to be stated explicitly. The possible adaptation of the security policy on critical cases allows to easily prevent security breaches and can, in addition, ensure some finer initialization properties whose violation could lead the program to crash. On one hand, results show that such a static and modular type checking allows to prove in an efficient way the absence of bugs. On the other hand, rare cases necessitate the introduction of dynamic

features and analysis to be extended to analyze more precisely arrays. With such an extension, the checker would be able to prove more classes correct, but this is left for future work.

On the formalization side, an obvious extension is to establish the soundness of the approach in presence of multi-threading. We conjecture the soundness result still holds with respect to the Java Memory Model because of the flow insensitive abstraction made on the heap.

The prototype and the Coq formalization and proofs can be downloaded from http://www.irisa.fr/celtique/ext/rawtypes/.

Acknowledgment. This work was partly supported by the Région Bretagne and by the ANSSI (JavaSec project, see http://www.ssi.gouv.fr/site_article226.html).

References

[1] Buckley, A.: JSR 202: Java™ class file specification update (December 2006), http://jcp.org/en/jsr/detail?id=202
[2] The CERT Sun Microsystems secure coding standard for Java (February 2010), https://www.securecoding.cert.org/confluence/display/java/
[3] Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Survey 41(3) (2009)
[4] Dean, D., Felten, E.W., Wallach, D.S.: Java security: From HotJava to Netscape and beyond. IEEE Symposium on Security and Privacy, 190–200 (1996)
[5] Demange, D., Jensen, T., Pichardie, D.: A provably correct stackless intermediate representation for java bytecode. Research Report RR-7021, INRIA (2009), http://hal.inria.fr/inria-00414099/en/
[6] Fähndrich, M., Xia, S.: Establishing object invariants with delayed types. In: Proc. of OOPSLA 2007, pp. 337–350. ACM, New York (2007)
[7] Freund, S.N., Mitchell, J.C.: A type system for the Java bytecode language and verifier. J. Autom. Reasoning 30(3-4), 271–321 (2003)
[8] Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java™ Language Specification, 3rd edn. Addison Wesley, Reading (2005)
[9] Qi, X., Myers, A.C.: Masked types for sound object initialization. In: POPL, pp. 53–65. ACM, New York (2009)
[10] Secunia advisory sa10056: Sun jre and sdk untrusted applet privilege escalation vulnerability. Web (October 2003), http://secunia.com/advisories/10056/
[11] Sun. Secure coding guidelines for the Java programming language, version 3.0. Technical report, Oracle (2010), http://java.sun.com/security/seccodeguide.html
[12] Unkel, C., Lam, M.S.: Automatic inference of stationary fields: a generalization of Java's final fields. In: Proc. of POPL, pp. 183–195. ACM, New York (2008)

Flexible Scheduler-Independent Security

Heiko Mantel and Henning Sudbrock

Computer Science, TU Darmstadt, Germany
{mantel,sudbrock}@cs.tu-darmstadt.de

Abstract. We propose an approach to certify the information flow security of multi-threaded programs independently from the scheduling algorithm. A scheduler-independent verification is desirable because the scheduler is part of the runtime environment and, hence, usually not known when a program is analyzed. Unlike for other system properties, it is not straightforward to achieve scheduler independence when verifying information flow security, and the existing independence results are very restrictive. In this article, we show how some of these restrictions can be overcome. The key insight in our development of a novel scheduler-independent information flow property was the identification of a suitable class of schedulers that covers the most relevant schedulers. The contributions of this article include a novel security property, a scheduler independence result, and a provably sound program analysis.

1 Introduction

Whether a program can be entrusted secrets depends on the flow of information caused by running this program. *Noninterference* is a security property that characterizes secure information flow by the requirement that a program's output to untrusted sinks does not depend on secrets [1]. This requirement ensures that an attacker cannot conclude any information about secrets from the output that he can possibly observe, even if he has access to the full code of the program.

In order to obtain reliable analysis results, a noninterference analysis needs to properly respect the semantics of the given language. This raises the question of how to deal with aspects that influence a program's behavior, but that are outside the definition of the programming language's semantics. Examples are elements of the language whose behavior is not specified in the language's definition (e.g. native methods in Java) or elements of the runtime environment.

In this article, we focus on how to deal with a particular element of the runtime environment, namely the scheduler. Unlike for other properties, it is not sufficient to assume a possibilistic scheduler in a noninterference analysis, i.e. the scheduler that admits all possible scheduling choices. Secure information flow under a possibilistic scheduler need not imply that a program is secure for other schedulers because refining some part of a secure system's specification (such as the scheduler) may result in a system that violates security [2].

Many information flow analyses are scheduler dependent in the sense that they assume a particular scheduler, such that the analysis results are only valid if the

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 116–133, 2010.

program is executed under this scheduler. For instance, a uniform scheduler is assumed in [3], a Round-Robin scheduler in [4], and a possibilistic scheduler in [5].

There are also a few approaches that support a scheduler-independent analysis. So far, there are two main approaches to defining information flow properties that are scheduler independent, firstly, requiring that a program's public output is deterministically determined by the program's public input and, secondly, requiring that a program's possible behaviors for any two inputs, which comprise identical public inputs, are stepwise indistinguishable to an observer of the program's public outputs. The first approach was introduced by Zdancewic and Myers, adapting the idea of defining secure information flow based on observational determinism to language-based security [6]. The second approach was used by Sabelfeld and Sands to define the so-called strong security property [7]. While both approaches provide a semantic basis for program analyses that are sound independently of the scheduling algorithm, they are far from satisfactory. The resulting security properties are very restrictive because they are violated by many intuitively secure programs. The main deficiency of security properties based on observational determinism is that they forbid nondeterminism in the publicly observable behavior of a program, albeit intuitively secure programs can have nondeterministic public output. Strong security suffers from a different problem. It requires a restrictive lock-step indistinguishability, which implies, for instance, that a program's execution time must not depend on secrets, even if such differences in the timing do not cause differences in the public output.

In this article, we propose a scheduler-independent security property that permits nondeterminism in a program's publicly observable behavior without requiring a restrictive lock-step indistinguishability. Our solution does not require non-standard modifications to the interface of schedulers (as in other approaches, e.g., [8,9]). In fact our approach is the first that is suitable for programs with nondeterministic publicly observable behavior whose runtime depends on secrets, while providing scheduler independence for common schedulers like Round-Robin and uniform schedulers (see Section 6 for a more detailed comparison). The existence of a scheduler-independent security property with these features is somewhat surprising given that Sabelfeld proved in [10] that strong security is the weakest property that implies information flow security for a natural class of schedulers. The key step in our development was the identification of a different class of schedulers, the *robust schedulers*, that also contains the most relevant schedulers.

In summary, our contributions include (1) the definition of a novel security property for multi-threaded programs, (2) the novel notion of robust schedulers, (3) a theorem showing that our security property is scheduler independent for robust schedulers, and (4) a provably sound, scheduler-independent program analysis for enforcing our security property. We illustrate the progress made by the security analysis of a small, but realistic example program. The proofs of all theorems in this article are made available on the authors' website. We expect that our improvements constitute a significant step towards more widely applicable information flow analyses for concurrent programs.

2 Preliminaries

In this article, we consider multi-threaded programs that communicate via shared memory. In this section, we leave the set *Com* of commands unspecified. It will be instantiated by a multi-threaded imperative programming language in Section 5.

2.1 Execution Model

A multi-threaded program executes threads from a pool, that we represent by a finite list of threads. The thread pool's size has no upper bound and varies during program execution as threads are removed upon termination and new threads may be spawned. Active threads are implicitly numbered consecutively by their position in the thread pool. The program memory is shared between all threads, and thereby provides a means for inter-thread communication.

A *thread configuration* is a pair $\langle com, mem\rangle \in (Com \cup \{\mathsf{stop}\}) \times (Var \to Val)$ that models a snapshot during the execution of a single thread. If $com = \mathsf{stop}$ then the thread has terminated, while if $com \in Com$ then this is the command that remains to be executed by the thread. The second element of a thread configuration, *mem*, models the current state of the program memory by assigning a value to each program variable, where *Var* is the set of variables and *Val* is a not further specified set of values. We denote the set $(Var \to Val)$ with *Mem*.

A *program configuration* is a pair $\langle thr, mem\rangle$, consisting of a *thread pool* $thr : \mathbb{N}_0 \to (Com \cup \{\mathsf{stop}\})$ and a *shared memory* $mem \in Mem$, that models a snapshot during the execution of a multi-threaded program. If $thr(k) = \mathsf{stop}$ then there is no thread at position k, while if $thr(k) \in Com$ then this is the command that remains to be executed by the kth thread. We define the size of a thread pool *thr* by $\sharp(thr) = |\{k \in \mathbb{N}_0 \mid thr(k) \neq \mathsf{stop}\}|$. We furthermore require that $thr(k) \neq \mathsf{stop}$ implies $thr(l) \neq \mathsf{stop}$ for all $l < k$, i.e. the thread pool has no gaps. We denote the set of all thread pools satisfying these requirements with *Progs*. Note that $\sharp(thr) = \min\{k \in \mathbb{N}_0 \mid thr(k) = \mathsf{stop}\}$ holds for all $thr \in Progs$.

To make scheduling explicit, we introduce system configurations. Formally, a *system configuration* is a triple $\langle thr, mem, sst\rangle$ such that $\langle thr, mem\rangle$ is a program configuration and $sst \in sSt$ is a *scheduler state*. Scheduler states and other aspects of scheduling will be introduced in Section 3.1. We use *Conf* to denote the set of all system configurations and introduce selector functions *getT*, *getM*, and *getS* to retrieve the elements from a system configuration, i.e., $getT(\langle thr, mem, sst\rangle) = thr$, $getM(\langle thr, mem, sst\rangle) = mem$, and $getS(\langle thr, mem, sst\rangle) = sst$.

To model execution steps, we introduce the judgment

$$conf \Rightarrow_{k,p} conf'$$

where $conf, conf' \in Conf$, $k \in \mathbb{N}_0$, and $0 < p \leq 1$. Intuitively, this judgment models that a transition from the system configuration *conf* to the system configuration *conf'* is possible. The index k identifies the thread performing a computation step by its position in the thread pool $getT(conf)$. The probability of the transition is specified by the index p. Note that purely deterministic behavior can be modeled by restricting p to the singleton set $\{1\}$.

Derivability of the judgment for system configurations is defined based on two further judgments. The judgment $(sst, sin) \overset{k}{\rightsquigarrow}_p sst'$ models that the scheduler selects the kth thread with probability p. This decision may be based on sst, the state of the scheduler, and further input sin. The resulting scheduler state is sst'. The judgment $\langle com, mem\rangle \overset{\alpha}{\rightarrowtriangle} \langle com', mem'\rangle$ models that executing the command com in the memory mem results in the thread configuration $\langle com', mem'\rangle$. The label $\alpha \in Lab$ is an event from the set $Lab = \{new(coms) \mid coms \in Com^*\}$ that captures information about the creation of threads in the computation step. An event $new(coms)$ models that new threads are spawned to execute the commands in the list $coms$. We omit the label if no new threads were spawned.

Based on the above two judgments we can now model the stepwise execution of a multi-threaded program under a given scheduler by the following rule:

$$\frac{\begin{array}{cc}(sst, sin) \overset{k}{\rightsquigarrow}_p sst' & \langle thr(k), mem\rangle \overset{\alpha}{\rightarrowtriangle} \langle com', mem'\rangle \\ sin = obs(thr, mem) & thr' = update_k(thr, com', \alpha)\end{array}}{\langle\, thr, mem, sst\,\rangle \Rightarrow_{k,p} \langle\, thr', mem', sst'\,\rangle} \tag{1}$$

The third premise of the rule indicates that inputs to schedulers result from an observation of the program configuration (Section 3.1 will refine scheduler inputs). The function $update_k$ in the fourth premise updates the thread pool thr. In this article, we assume that spawned threads are inserted in the list of threads after the thread that executed the spawn operation. Moreover, if a thread terminates then it is removed from the list. For $\alpha = new(coms)$ and $k < \sharp(thr)$, the thread pool $update_k(thr, com, \alpha)$ is defined by $replace_k(thr, coms)$ if $com = \mathsf{stop}$ and by $replace_k(thr, [com]{::}coms)$ otherwise (where $::$ denotes list concatenation).[1]

2.2 Traces

A *trace* models a possible run of a program under some scheduler by a pair (str, dtr). The *system trace* $str : \mathbb{N}_0 \to Conf$ models the run of the program, where $str(0)$ is a snapshot of the system before starting its execution, and $str(k)$ is a snapshot of the system after k execution steps. The *decision trace* $dtr : \mathbb{N}_0 \to \mathbb{N}_0$ models the scheduler's decisions during the run, where $dtr(k)$ is the position of the thread selected for execution in the kth execution step.

Definition 1. *A* trace *is a pair* $tr = (str, dtr)$ *consisting of a* system trace $str : \mathbb{N}_0 \to Conf$ *and a* decision trace $dtr : \mathbb{N}_0 \to \mathbb{N}_0$.

We model the termination of a run by designated final configurations, namely those configurations in which the thread pool is empty.

Definition 2. *We call a trace* $tr = (str, dtr)$ terminating *if it reaches a final configuration, i.e. if* $\exists j \in \mathbb{N}_0 : \sharp(getT(str(j))) = 0$ *holds. The* length of a terminating trace *is* $\sharp(tr) = \min\{j \in \mathbb{N}_0 \mid \sharp(getT(str(j))) = 0\}$.

[1] For $k \in \mathbb{N}_0$, the partial function $replace_k : (Progs \times Com^*) \rightharpoonup Progs$ is defined by $replace_k(thr, [com_0, \ldots, com_{n-1}])(j) = com$ for $k < \sharp(thr)$ where $com = thr(j)$ if $j < k$, $com = com_{j-k}$ if $k \leq j < k + n$, and $com = thr(j - n)$ if $k + n \leq j$.

Given a set of traces Tr, we define the subset of terminating traces *by*
$$Tr{\Downarrow} = \{(str, dtr) \in Tr \mid \exists j \in \mathbb{N}_0 : \sharp(getT(str(j))) = 0\}.$$
The subset of traces terminating with memory *mem is defined by*
$$Tr{\Downarrow}_{mem} = \{tr \in Tr{\Downarrow} \mid getM(str(\sharp(tr))) = mem\}.$$

3 Scheduling and Scheduler-Specific Security

We capture scheduler-dependent information flow security by a noninterference-like information flow property that is parametric in the choice of the scheduler.

3.1 An Explicit Scheduler Model

During the execution of a multi-threaded program, the scheduler repeatedly decides which thread shall next proceed with the computation. Scheduling algorithms differ not only in how they make this decision, but also in the information on which they base their decision. For instance, a uniform scheduler needs to know the current number of threads in order to randomly choose among the available threads with equal probability. A Round-Robin scheduler iterates over the list of available threads in a cyclic fashion. Beyond knowing the number of threads, this requires that the scheduler remembers its scheduling choice from the previous step. A scheduler might even need to know part of the program's memory, for instance, in priority-based scheduling if priorities are first-class values that may be read and modified by the program itself. To cover these and various other possibilities, we assume that schedulers base their decision on their current internal state and their observation of the program configuration.

We leave the set of scheduler states sSt and the set of scheduler inputs sIn unspecified. We only assume that it is at least possible to retrieve the current number of threads from any given input $sin \in sIn$ and denote this number by $\sharp(sin)$. The scheduler's output is modeled by a natural number, indicating the position of the next thread to be run. The behaviour of a scheduler is modeled by a labeled transition relation, where a label $(sin, k, p) \in sIn \times \mathbb{N}_0 \times]0, 1]$ indicates that sin is the input to the scheduler, k is the position of the chosen thread, and p is the probability of this choice. We use probabilistic transitions in order to account for schedulers that are not deterministic like, e.g., uniform schedulers.

Definition 3. *A* scheduler *is a labeled transition system $(sSt, sst_0, sLab, \rightarrow)$ with initial state $sst_0 \in sSt$, label set $sLab = sIn \times \mathbb{N}_0 \times]0, 1]$, and transition relation $\rightarrow \subseteq sSt \times sLab \times sSt$ that satisfies the following properties:*

1. *if $(sst, (sin, k, p), sst') \in \rightarrow$, then $k < \sharp(sin)$;*
2. *for each triple (sst, sin, k) there is at most one probability $p \in]0, 1]$ and one scheduler state $sst' \in sSt$ such that $(sst, (sin, k, p), sst') \in \rightarrow$; and*
3. *the equality $\sum \{\!|p \mid \exists k, sst' : (sst, (sin, k, p), sst') \in \rightarrow|\!\} = 1$ holds for each pair (sst, sin) (where $\{\!|\ldots|\!\}$ denotes a multiset).*

The first property in Definition 3 ensures that a scheduler selects among the available threads, the second property ensures that a scheduler cannot choose

a single thread position with multiple probabilities or modify its internal state nondeterministically, and the third property ensures that in each state the probabilities of the transitions for a given input form a probability distribution.

Definition 4. *The judgment* $(sst, sin) \overset{k}{\rightsquigarrow}_p sst'$ *(from Section 2.1) is* derivable *for a scheduler* $(sSt, sst_0, sLab, \rightarrow)$ *if and only if* $(sst, (sin, k, p), sst') \in \rightarrow$.

Our model for schedulers is sufficiently expressive for common scheduling algorithms such as uniform, Round-Robin, and priority-based scheduling.

Example 1. A uniform scheduler can be modeled by the labeled transition system $UNI = (\{s\}, s, sLab, \rightarrow_{UNI})$, where $\rightarrow_{UNI}$ is defined by $(sst, (sin, k, p), sst') \in \rightarrow_{UNI}$ if and only if $k < \sharp(sin)$, $p = 1/\sharp(sin)$, and $sst' = sst = s$.

Example 2. A Round-Robin scheduler can be modeled by the labeled transition system $RR = (sSt_{RR}, sst_{RR,0}, sLab, \rightarrow_{RR})$, with $sSt_{RR} : \{choice, size\} \rightarrow \mathbb{N}_0$, $sst_{RR,0}(choice) = 0$, and $sst_{RR,0}(size) = 1$. The scheduler variables *choice* and *size* store from the previous step which thread position was selected and what size the thread pool had. The transition relation is defined by $(sst, (sin, k, p), sst') \in \rightarrow_{RR}$ if and only if $p = 1$, $k = (sst(choice) + 1 + (\sharp(sin) - sst(size))) \mod \sharp(sin)$, $sst'(choice) = k$, and $sst'(size) = \sharp(sin)$.[2]

We specify the traces modeling possible program runs under a given scheduler.

Definition 5. *For a scheduler* $\mathcal{S} = (sSt, sst_0, sLab, \rightarrow)$*, the* set of possible traces starting in a system configuration *conf is defined by* $(str, dtr) \in Tr_{\mathcal{S}}(conf)$ *if and only if*

$$str(0) = conf \wedge \forall j \in \mathbb{N}_0 : \left(\begin{array}{l} (\exists p \in]0,1] : \ str(j) \Rightarrow_{dtr(j),p} str(j+1)) \\ \vee \left(\begin{array}{l} \sharp(getT(str(j))) = 0 \\ \wedge\, str(j+1) = str(j) \wedge dtr(j) = 0 \end{array} \right) \end{array} \right)$$

Note that if a final configuration is reached, the program performs no further computation steps. We model this by requiring that from that point on the system trace infinitely often repeats the final configuration and the decision trace infinitely often repeats the value 0. We say that a *system configuration conf is terminating under a scheduler* $\mathcal{S}$ if $Tr_{\mathcal{S}}(conf) = Tr_{\mathcal{S}}(conf)\Downarrow$. *A thread pool thr is terminating* if for all $mem \in Mem$ the system configuration $\langle thr, mem, sst_0 \rangle$ is terminating under all schedulers $\mathcal{S}$ with initial scheduler state sst_0.

We use the probabilities of single execution steps to compute the probability that a terminating program run with a given sequence of scheduler decisions occurs when executing a program in a given configuration under a given scheduler:

Definition 6. *For a trace* $tr = (str, dtr) \in Tr_{\mathcal{S}}(conf)\Downarrow$ *we define the* probability of tr under the scheduler $\mathcal{S}$ *by* $\rho_{\mathcal{S}}(tr) = p_0 * \ldots * p_{\sharp(tr)-1}$*, where the* p_j *are the unique probabilities with* $str(j) \Rightarrow_{dtr(j),p_j} str(j+1)$ *for* $0 \leq j < \sharp(tr)$.

[2] Note that our condition on k ensures, firstly, that no thread is skipped if the current thread terminates (in this case $\sharp(sin) - sst(size)$ equals -1) and, secondly, that newly created threads obtain their term only after all other threads have been scheduled (in this case $\sharp(sin) - sst(size)$ equals the number of newly created threads).

3.2 Scheduler-Specific Security Property

We consider a security lattice with two security domains, *low* and *high*, where the requirement is that no information flows from *high* to *low*. This is the simplest policy capturing information flow security. A *domain assignment* is a function $dom : Var \to \{low, high\}$ that associates a security domain with each program variable. The resulting security requirement is that no information may flow from variables with domain *high* to variables with domain *low*.

We assume that attackers cannot directly access the values of high variables (i.e., access control works correctly). The following indistinguishability relation captures this upper bound on the observational capabilities of attackers.

Definition 7. *Two memories* $mem, mem' \in Mem$ *are* low-equal*, denoted by* $mem =_L mem'$*, if and only if* $mem(var) = mem'(var)$ *for all* $var \in Var$ *with* $dom(var) = low$*. We use* $[mem]_{=_L}$ *to denote the equivalence class* $\{mem' \in Mem \mid mem =_L mem'\}$*.*

Definition 8. A thread pool *thr* is $\mathcal{S}$-secure *for* $\mathcal{S} = (sSt, sst_0, sLab, \to)$ *if*

$$\sum_{mem' \in [mem]_{=_L}} \rho_{\mathcal{S}}(\langle thr, mem_1, sst_0 \rangle, mem') = \sum_{mem' \in [mem]_{=_L}} \rho_{\mathcal{S}}(\langle thr, mem_2, sst_0 \rangle, mem')$$

holds for all mem_1*,* mem_2*,* $mem \in Mem$ *with* $mem_1 =_L mem_2$*, where the value* $\rho_{\mathcal{S}}(conf, mem)$ *is the probability that a program run under the scheduler* $\mathcal{S}$ *that starts in the system configuration conf terminates with memory mem. It is defined by* $\rho_{\mathcal{S}}(conf, mem) = \sum_{tr \in Tr_{\mathcal{S}}(conf) \Downarrow_{mem}} \rho_{\mathcal{S}}(tr)$*.*

A command *com* is $\mathcal{S}$-secure *if the thread pool* thr_{com} *containing the single thread com is* $\mathcal{S}$*-secure (i.e.* $thr_{com}(0) = com$ *and* $thr_{com}(j) = \mathsf{stop}$ *for all* $j > 0$*).*

Our notion of $\mathcal{S}$-security guarantees that the probability that an $\mathcal{S}$-secure program terminates with given values of low variables is independent from the initial values of secrets. This means, $\mathcal{S}$-security implies that an attacker who can observe the initial and final values of low variables cannot conclude anything about high inputs. This implication still holds if the attacker knows the code of the program and the scheduling algorithm. Moreover, it also holds if the attacker can observe multiple runs of the program. Note that the number of execution steps of an $\mathcal{S}$-secure program may depend on the values of low variables (in contrast to, e.g., the bisimulation-based scheduler-specific security condition in [7]).

Remark 1. While we prefer to define $\mathcal{S}$-security using the probabilities of traces, an equivalent property could be defined using Markov chains, as, e.g., in [11].

4 Scheduler-Independent Information Flow Security

In this section, we present the novel information flow property that is the main contribution of this article. Our security property is scheduler independent in the sense that it implies $\mathcal{S}$-security for a large class of schedulers. We characterize this

class by the novel notion of robust schedulers and show that this class covers the most common scheduling algorithms. As we will illustrate, our security property is suitable for programs with nondeterministic behavior and whose runtime may depend on secrets. That is, it overcomes restrictions of the existing scheduler-independent security properties.

4.1 A Novel Security Property

We partition the set *Com* into *high commands* that definitely do not modify low variables and into *low commands* that potentially modify values of low variables.

Definition 9. *The* set of high commands $HCom \subset Com$ *is the largest set of commands such that if* $com \in HCom$ *then the following holds:*

$$\forall com' \in Com \cup \{\mathsf{stop}\} : \forall mem, mem' \in Mem : \forall com_0, \ldots, com_{n-1} \in Com :$$
$$\langle com, mem\rangle \xrightarrow{new([com_0,\ldots,com_{n-1}])} \langle com', mem'\rangle \implies$$
$$\big(mem =_L mem' \wedge com' \in HCom \cup \{\mathsf{stop}\} \wedge \forall j \in \{0,\ldots,n-1\}.\ com_j \in HCom\big)$$

The set of low commands $LCom \subset Com$ *is defined as* $Com \setminus HCom$.

We refer to threads executing high commands as *high threads* and to threads executing low commands as *low threads*. Note that a low thread becomes high after an execution step if the command that remains to be executed is high. High threads, by definition, cannot become low during a program's execution.

Low matches link the positions of corresponding low threads in thread pools.

Definition 10. *A* low match *of two thread pools* thr_1 *and* thr_2 *with the same number of low threads (i.e.* $|\{k_1 \in \mathbb{N}_0 \mid thr_1(k_1) \in LCom\}| = |\{k_2 \in \mathbb{N}_0 \mid thr_2(k_2) \in LCom\}|$*) is an order-preserving bijection with the domain* $\{k_1 \in \mathbb{N}_0 \mid thr_1(k_1) \in LCom\}$ *and the range* $\{k_2 \in \mathbb{N}_0 \mid thr_2(k_2) \in LCom\}$.

That is, a low match maps the position of the nth low thread in one thread pool to the position of the nth low thread in the other thread pool:

Theorem 1. *The low match of* thr_1 *and* thr_2 *is unique and given by the function*

$$l\text{-}match_{thr_1,thr_2}(k_1) =$$
$$\min\Big\{k_2 \in \mathbb{N}_0 \mid \big|\{l_1 \le k_1 \mid thr_1(l_1) \in LCom\}\big| = \big|\{l_2 \le k_2 \mid thr_2(l_2) \in LCom\}\big|\Big\}.$$

Due to space restrictions, the proof of the above theorem as well as the proofs of all other theorems in this article are provided on the authors' website.

We use the PER-approach [12] to define the novel security property, i.e., we define an indistinguishability relation on thread pools that is not reflexive, as it only relates thread pools to themselves that have secure information flow.

Definition 11. *A symmetric relation* R *on thread pools with an equal number of low threads is a* low bisimulation modulo low matching, *if whenever* $thr_1\ R\ thr_2$, $mem_1 =_L mem_2$, *and* $\langle thr_1(k_1), mem_1\rangle \xrightarrow{\alpha_1} \langle com_1, mem_1'\rangle$, *then*

1. *if* $thr_1(k_1) \in LCom$*, then there exist* com_2*,* mem_2'*, and* α_2 *with*
   (a) $\langle thr_2(k_2), mem_2\rangle \xrightarrow{\alpha_2} \langle com_2, mem_2'\rangle$*,*
   (b) $mem_1' =_L mem_2'$*, and*
   (c) $update_{k_1}(thr_1, com_1, \alpha_1) \;\; R \;\; update_{k_2}(thr_2, com_2, \alpha_2)$
   where $k_2 = l\text{-}match_{thr_1, thr_2}(k_1)$*; and*
2. *if* $thr_1(k_1) \in HCom$*, then* $update_{k_1}(thr_1, com_1, \alpha_1) \;\; R \;\; thr_2$*.*

The relation $\sim$ *is the union of all low bisimulations modulo low matching.*

Definition 12. A thread pool *thr* is FSI-secure *if* $thr \sim thr$. A command *com* is FSI-secure *if the thread pool* thr_{com} *(see Definition 8) is FSI-secure.*

We will show in Section 4.3 that all terminating FSI-secure programs are also $\mathcal{S}$-secure for any robust scheduler $\mathcal{S}$. This scheduler independence result motivates the expansion of the acronym FSI-security, which is *flexible scheduler-independent security*.

The following theorem shows that FSI-security is compositional.

Theorem 2. *Let* thr_1 *and* thr_2 *be FSI-secure thread pools. Then their* parallel composition $par(thr_1, thr_2)$ *is also FSI-secure, where*

$$par(thr_1, thr_2)(k) = \begin{cases} thr_1(k) & \text{, if } k < \sharp(thr_1) \\ thr_2(k - \sharp(thr_1)) & \text{, otherwise.} \end{cases}$$

The compositionality result is not only crucial for a modular analysis, but also illustrates that FSI-security is suitable for multi-threaded programs containing races: As it suffices that each individual thread of a program is FSI-secure, FSI-security imposes no restrictions on the relationships between variables occurring in concurrent threads. This constitutes a significant improvement over security properties based on observational determinism.

Moreover, FSI-security is suitable for programs whose runtime depends on confidential information. While FSI-security requires stepwise indistinguishability for low threads (Item *1* in Definition 11), no such requirement is imposed on a thread once it is high (Item *2* in Definition 11). This constitutes a major improvement over the strong security condition. In particular, unlike strong security, FSI-security is satisfied by every high command.

Theorem 3. *Let* $com \in HCom$*. Then com is FSI-secure.*

In summary, FSI-security overcomes serious deficiencies of the two main approaches to defining scheduler-independent security mentioned in Section 1.

4.2 The Class of Robust Schedulers

The essential idea of robust schedulers is that the scheduling order of low threads does not depend on the high threads in a thread pool. We formalize the class of robust schedulers in Definition 15 based on the auxiliary notions of thread purge functions (Definition 13) and of $\mathcal{S}$-simulations (Definition 14).

Definition 13. *The thread pool th-purge(thr) is defined by*

$$\textit{th-purge}(thr)(k_1) = thr\Big(\min\big\{k_2 \in \mathbb{N}_0 \mid k_1 = \big|\{l < k_2 \mid thr(l) \in LCom \cup \{\mathsf{stop}\}\}\big|\big\}\Big)$$

for all $k_1 \in \mathbb{N}_0$. We denote with th-purge(conf) the system configuration obtained from conf by replacing $getT(conf)$ with th-purge($getT(conf)$).

Intuitively, *th-purge*(*thr*) is obtained from *thr* by removing all high threads and leaving the order of low threads unchanged:

Theorem 4. *For a thread pool thr, th-purge(thr) contains no high threads and as many low threads as thr. Moreover, if $k \in \mathbb{N}_0$ with $thr(k) \in LCom$ then*

$$thr(k) = \textit{th-purge}(thr)(\textit{l-match}_{thr,\textit{th-purge}(thr)}(k)).$$

Definition 14. *Let $\mathcal{S} = (sSt, sst_0, sLab, \rightarrow)$ be a scheduler. An* $\mathcal{S}$-simulation *is a relation $<$ that relates arbitrary configurations with configurations that do not contain high threads, such that $conf_1 < conf_2$ and $conf_1 \Rightarrow_{k_1,p_1} conf_1'$ imply*

1. *if $getT(conf_1)(k_1) \in LCom$, then there exists $conf_2'$ with*
   (a) *$conf_2 \Rightarrow_{k_2,p_2} conf_2'$, where $k_2 = \textit{l-match}_{getT(conf_1),getT(conf_2)}(k_1)$ and $p_2 = p_1/\textit{l-prob}_{\mathcal{S}}(conf_1)$, and $\textit{l-prob}_{\mathcal{S}}(conf)$ denotes the probability that a low thread is selected by the scheduler $\mathcal{S}$ in the system configuration conf that is defined by $\textit{l-prob}_{\mathcal{S}}(\langle thr, mem, sst\rangle) = \sum\{\!|p \mid \exists k, sst' : thr(k) \in LCom \wedge (sst, obs(thr, mem)) \overset{k}{\rightsquigarrow}_p sst'|\!\}$, as well as*
   (b) *$conf_1' < \textit{th-purge}(conf_2')$; and*
2. *if $getT(conf_1)(k_1) \in HCom$, then $conf_1' < conf_2$.*

The relation $<_{\mathcal{S}}$ is the union of all $\mathcal{S}$-simulations.

Definition 15. *The scheduler $\mathcal{S} = (sSt, sst_0, sLab, \rightarrow)$ is* robust *if*

$$\langle thr, mem, sst_0\rangle <_{\mathcal{S}} \textit{th-purge}(\langle thr, mem, sst_0\rangle)$$

holds for each FSI-secure thread pool thr and each memory mem.

Intuitively, a scheduler is robust if the scheduling of low threads during a run of an FSI-secure thread pool remains unchanged when one removes all high threads from the thread pool. That is, the probability that the scheduler selects a low thread among all low threads in a configuration equals the probability to select the matching low thread if all high threads were removed (i.e., $p_2 = p_1/\textit{l-prob}_{\mathcal{S}}(conf_1)$). This is, in particular, the case for uniform and Round-Robin schedulers:

Theorem 5. *The uniform scheduler (see Example 1) is robust.*

Theorem 6. *The Round-Robin scheduler (see Example 2) is robust.*

Robust schedulers will only be employed in combination with observation functions that properly confine the interface between programs and schedulers:

Definition 16. *We call the observation function obs (introduced in Section 2.1)* confined, *if it satisfies the following property for all thread pools thr_1, thr_2 and for all memories mem_1, mem_2:*

$$(\sharp(thr_1) = \sharp(thr_2) \wedge mem_1 =_L mem_2) \implies obs(thr_1, mem_1) = obs(thr_2, mem_2)$$

Confined observation functions only provide information about the current number of threads and the values of public variables. This is sufficient for common schedulers like Round-Robin or priority-based schedulers.

Note that Definition 15 quantifies over all FSI-secure thread pools. This is essential. Quantifying over all thread pools (i.e., including ones that are not FSI-secure) would result in a significantly smaller class of robust schedulers, that, for instance, does not include Round-Robin and uniform schedulers.

4.3 Scheduler Independence Result

We are now ready to present the scheduler independence result:

Theorem 7. *Let thr be a terminating thread pool that is FSI-secure and let $\mathcal{S}$ be a robust scheduler under a confined observation function. Then the thread pool thr is $\mathcal{S}$-secure.*

Theorems 5 and 6 show that we obtain scheduler-independent information flow security for a practically relevant class of schedulers. As FSI-security overcomes restrictions of the two main approaches to scheduler-independent security (see Section 4.1), we expect that our results will contribute to more widely applicable information flow analyses for concurrent programs.

5 Security Analysis for a Multi-threaded Language

We use a simple multi-threaded imperative programming language supporting the dynamic creation of new threads for illustrating how to analyze concrete programs with respect to FSI-security. We define the set *Com* by the following grammar (using a set *Exp* of expressions that we do not specify further):

$$\begin{aligned} com ::= {} & \mathsf{skip} \mid var{:=}exp \mid com; com \mid \mathsf{if}\ (exp)\ \mathsf{then}\ com\ \mathsf{else}\ com\ \mathsf{fi} \\ & \mid \mathsf{while}\ (exp)\ \mathsf{do}\ com\ \mathsf{od} \mid \mathsf{spawn}(com, \ldots, com), \end{aligned}$$

where $var \in Var$ and $exp \in Exp$. The operational semantics for commands is formalized by a calculus for the judgment $\langle com, mem\rangle \xrightarrow{\alpha} \langle com', mem'\rangle$ introduced in Section 2.1. The derivation rules are as usual, we refrain from stating their definition due to space restrictions.

5.1 Security Type System

We present a security type system for our example language. This type system provides the basis for an automated scheduler-independent security analysis.

We type commands with types of the form (ass, stp), where $ass, stp \in \{low, high\}$. The intuition of the typing judgment $\vdash com : (ass, stp)$ is as follows: If $ass = high$, then neither the thread executing *com* nor the threads that are created due to spawn-commands within *com* assign to low variables, i.e., *com* is a high command. If $stp = low$, then the number of execution steps made by a thread executing *com* cannot depend on the values of high variables. However, the execution time of threads spawned by this thread may depend on high values.

The typing rules are displayed in Figure 1. We denote with $dom(exp)$ the security domain of an expression, where $dom(exp) = low$ if all variables occurring

$$[\mathsf{SKIP}]\frac{}{\vdash \mathsf{skip} : (high, low)} \qquad [\mathsf{ASS}]\frac{dom(exp) \sqsubseteq dom(var)}{\vdash var{:=}exp : (dom(var), low)}$$

$$[\mathsf{IF}]\frac{\vdash com_1 : (ass, stp) \quad \vdash com_2 : (ass, stp) \quad dom(exp) \sqsubseteq ass}{\vdash \mathsf{if}\ (exp)\ \mathsf{then}\ com_1\ \mathsf{else}\ com_2\ \mathsf{fi} : (ass, stp \sqcup dom(exp))}$$

$$[\mathsf{WHILE}]\frac{\vdash com : (ass, stp) \quad stp \sqcup dom(exp) \sqsubseteq ass}{\vdash \mathsf{while}\ (exp)\ \mathsf{do}\ com\ \mathsf{od} : (ass, stp \sqcup dom(exp))}$$

$$[\mathsf{SPAWN}]\frac{\forall i \in \{0, \ldots, k-1\}.\ \vdash com_i : (ass, stp_i)}{\vdash \mathsf{spawn}(com_0, \ldots, com_{k-1}) : (ass, low)}$$

$$[\mathsf{SEQ}]\frac{\vdash com_1 : (ass_1, stp_1) \quad \vdash com_2 : (ass_2, stp_2) \quad stp_1 \sqsubseteq ass_2}{\vdash com_1; com_2 : (ass_1 \sqcap ass_2, stp_1 \sqcup stp_2)}$$

$$[\mathsf{SUB}]\frac{\vdash com : (ass', stp') \quad ass \sqsubseteq ass' \quad stp' \sqsubseteq stp}{\vdash com : (ass, stp)}$$

Fig. 1. Security type system

in *exp* have domain *low*, and $dom(exp) = high$ otherwise. As usual for a two-level policy, we assume $low \sqsubseteq high$ and denote the greatest lower bound respectively least upper bound operator on security domains with $\sqcap$ and $\sqcup$, respectively. Note that subtyping is covariant in the first component of a type and contravariant in its second component (compare rule [SUB] in Figure 1).

Rule [ASS] forbids assignments from high to low variables, and rules [IF] and [WHILE] forbid assignments to low variables under high guards of conditionals and loops (compare, e.g., [13]). Furthermore, rules [IF] and [WHILE] ensure that commands containing high guards (and whose runtime might hence depend on the values of high variables) can only be typed if $stp = high$. Rule [SPAWN] allows to type programs that dynamically spawn threads: A spawn-command is typable with $stp = low$, as it is executed in a single execution step. Moreover, the command $\mathsf{spawn}(com_0, \ldots, com_{k-1})$ is only typable with $ass = high$ if each com_i is typable with $ass = high$. Rules [SEQ] and [WHILE] ensure that if a typable command assigns to low variables, then its runtime before such an assignment only depends on the values of low variables. This is essential for the soundness of the type system, as it ensures that lock-step execution is possible for low threads that correspond to each other under the low matching.[3]

Theorem 8. *If the judgment* $\vdash com : (ass, stp)$ *is derivable in the type system for some* $com \in Com$ *and* $ass, stp \in \{low, high\}$ *then com is FSI-secure.*

[3] Note that our security type system does not contain a rule for typing conditionals with high guards whose branches may contain assignments to low variables if the branches are related by an indistinguishability relation. Such a rule is, e.g., provided in [14], and could be soundly integrated into our type system. We refrain from such a rule here to ensure that there are no choice points when generating a type derivation.

Initial thread :
networkOut$_l$:="getStockPrices";
stockPrices$_l$:=networkIn$_l$;
spawn(*writeStockPricesToDatabase*);
networkOut$_l$:="getFundsPrices";
fundsPrices$_l$:=networkIn$_l$;
spawn(*writeFundsPricesToDatabase*);
spawn(*computeAccountOverview*)

writeStockPricesToDatabase:
i$_l$:=0;
while (i$_l$ < getSize(stockPrices$_l$)) do
 database$_l$:= database$_l$
 + getTitleAt(stockPrices$_l$, i$_l$)
 + getPriceAt(stockPrices$_l$, i$_l$);
 i$_l$:= i$_l$+1 od

writeFundsPricesToDatabase:
j$_l$:=0;
while (j$_l$ < getSize(fundsPrices$_l$)) do
 database$_l$:= database$_l$
 + getTitleAt(fundsPrices$_l$, j$_l$)
 + getPriceAt(fundsPrices$_l$, j$_l$);
 j$_l$:= j$_l$+1 od

computeAccountOverview:
k$_h$:= 0; overview$_h$:= "";
while (k$_h$ < getSize(userPortfolio$_h$)) do
 title$_h$:= getTitleAt(userPortfolio$_h$, k$_h$);
 if (isStock(title$_h$)) then
 price$_h$:= getPriceFor(stockPrices$_l$, title$_h$)
 ∗ getQuantityAt(userPortfolio$_h$, k$_h$)
 else
 price$_h$:= getPriceFor(fundsPrices$_l$, title$_h$)
 ∗ getQuantityAt(userPortfolio$_h$, k$_h$)
 fi;
 oldPrice$_h$:=getLastPrice(database$_l$, title$_h$);
 if (oldPrice$_h$ ≤ price$_h$)
 then tendency$_h$:= "up"
 else tendency$_h$:= "down"
 fi;
 overview$_h$:= overview$_h$ + title$_h$
 + price$_h$ + tendency$_h$;
 k$_h$:= k$_h$+1
od

Fig. 2. Exemplary security analysis: implementation

Thus, due to the compositionality of FSI-security (Theorem 2), a thread pool *thr* is FSI-secure if $thr(k)$ is typable for each $k < \sharp(thr)$.

Note that the type systems proposed in [15,8,11,16] are similar to our type system as they restrict the assignments a program may perform after executing a conditional or a loop with a high guard. However, note that [15,11,16] only guarantee soundness for one scheduler-specific security property. Note also that [8] targets a language that allows dynamic thread creation (a typical feature of multi-threaded programming languages) only in a very limited form (no threads may be created inside loops), and the article assumes that threads idle after their termination instead of being removed from the thread pool. Our type system and its soundness result do not share these limitations. The scheduler independence result from [8] will be further compared to the result in this article in Section 6.

5.2 Exemplary Security Analysis

Consider the code fragment in Figure 2, which is part of an application managing personal finances. The program contacts two network-based services that provide up-to-date pricing information for stocks respectively funds (by writing to respectively reading from the variables networkOut$_l$ and networkIn$_l$). The program appends the retrieved information to the information in the variable database$_l$ that contains historical pricing information for future reference. Moreover, using the novel data and historical data already present in database$_l$, the program

generates an overview of the user's custody account. Those three activities are spawned in new threads (*writeStockPricesToDatabase*, *writeFundsPricesToDatabase*, and *computeAccountOverview*) to improve the interactivity of the overall program and not block computations following this code fragment. In our example, we model the network requests, the data retrieved from the network, the data stored in the database, and the generated overview with string values. The data encoded in those string values is accessed by the program using selector expressions like, for instance, getLastPrice.

The subscripts of variables indicate whether a variable is classified as low (l) or as high (h). Information in the database and information retrieved from the network services is public and classified as *low*, while the user's portfolio and the report created based on the portfolio are confidential and classified as *high*.

Applying the type system to the program proves that the program is FSI-secure: The initial thread as well as the threads *writeStockPricesToDatabase* and *writeFundsPricesToDatabase* are typable with the type (low, low), while the thread *computeAccountOverview* is typable with the type $(high, high)$.

Note that the program is rejected by existing analyses that guarantee security for common schedulers. Observational determinism [6] is violated, as the order in which entries are written to the database depends on the order in which the threads *writeStockPricesToDatabase* and *writeFundsPricesToDatabase* are scheduled. Strong security [7] is violated as the runtime of the loop in the thread *computeAccountOverview* depends on confidential information. The soundness of the type system together with the scheduler independence result guarantee that the order of the database entries never depends on confidential information when using a robust scheduler.

6 Related Work

An overview on information flow security in a multi-threaded setting is provided in [17]. Here, we focus on approaches that cover the problem of scheduling.

Most approaches assume a particular scheduling algorithm. In consequence, their results do not necessarily generalize to other schedulers. Several approaches consider a scheduler that selects threads purely nondeterministically (for instance, [5,18,19,20,21]). Uniform schedulers are assumed in [3,11], and a Round-Robin scheduler is assumed in [4,22].

There are only a few approaches to scheduler-independent information flow security. In the following, we discuss those approaches in more detail.

The idea of *observational determinism* goes back to McLean [23] and Roscoe et al. [24,25], who proposed security properties not at the level of a programming language, but more abstractly for specifications. The idea was adapted to a language-based setting in [6,26]. Observational determinism requires that public observations of program executions are deterministic regardless of the interleaving of threads and the values of secret variables. If this requirement is satisfied, restricting the possible interleavings by assuming a concrete scheduler cannot result in a dependency of public observations on secrets. Observational determinism has the drawback that it forbids useful nondeterminism which

occurs, for instance, when multiple threads append data to the same public variable (as in the example program from Section 5.2).

Sabelfeld and Sands [7] introduce *strong security*, which is scheduler-independent for a natural class of schedulers. Strong security is quite restrictive, as it requires that the runtime of a program must not depend on secret data. This drawback appeared unavoidable because [10] proved that the strong security condition is the weakest compositional property that implies information flow security for the natural class of schedulers. Hence, the strong security condition was used, despite its restrictiveness, as the foundation of many later developments (e.g., [27,28,29]) and has been generalized in various ways, e.g., for distributed systems [27] or to control declassification [30]. While [7] proposes a type system that can transform some insecure programs into strongly secure programs, only a subset of the intuitively secure programs is amenable to this approach (for instance, the type system does not transform the example program from Section 5.2 into a strongly secure program).

The combining calculus [20] is a first step towards combining approaches based on observational determinism and strong security as it allows the combination of different analysis techniques in a security analysis. However, a scheduler-independence result has not yet been established for the combining calculus.

Boudol and Castellani [8] propose a security type system for *controlled thread systems* that consist of a thread pool and a scheduler. If a controlled thread system is typable, then the thread pool is secure under the scheduler. In contrast to this article, the approach requires the size of a thread pool to remain fixed during a program run: dynamic thread creation is not supported, and threads remain in the thread pool upon termination (and may still be selected by the scheduler). Boudol and Castellani argue that if the termination of certain threads would be signaled to the scheduler, then controlled thread systems writing public variables cannot be typed. This is a non-standard restriction, as schedulers typically use the number of live threads when choosing the next thread.

As a different approach to relax the security property while remaining scheduler-independent, [9,31,32] propose to use non-standard schedulers that provide a customized interface to the scheduled threads. Via two special commands, programs can *hide* (and at a later point *unhide*) a thread; the scheduler guarantees that during the execution of hidden threads no other thread is scheduled. The approach allows to securely execute programs containing threads that assign to low variables after performing computations whose runtime depends on high data (hiding the thread during those computations), but at the cost that a scheduler with a non-standard interface must be used. Such threads are rejected by our security property, as they may cause information leakage when being executed under currently available schedulers.

Another approach that prevents scheduling during computations whose runtime depends on secrets is followed by [22]. It provides a program transformation that introduces yield-statements into a program instructing the scheduler to select another thread, such that no yield-statements occur during computations depending on secrets, and rescheduling only occurs after a yield-statement. The

approach is implemented for a Round-Robin scheduler, but the article argues that it is applicable for a wide class of schedulers. The transformation entails that computations on secrets block all remaining threads. This is particularly critical when these computations are time-consuming. In contrast, our approach allows any computations to be interleaved with the executions of other threads.

In the following we discuss two approaches that investigate scheduler-independence on the level of system specifications. Van der Meyden and Zhang [33] adapt security conditions for asynchronous systems to scheduled synchronous systems. They consider schedulers whose decisions do not depend on secret actions and show that the security properties are *scheduler-implementation independent* in the sense that a system satisfies a property under one implementation of a scheduler if and only if it satisfies the property under all of the scheduler's possible implementations. Note that this differs from requiring that security holds under different schedulers. Moreover, [33] prove that if the security definitions are satisfied for all deterministic schedulers, then they are also satisfied for all non-deterministic schedulers. Probabilistic schedulers are not considered.

Also when considering protocols the scheduling might impact security. In particular, the hidden random choice of a secret value in a security protocol could be revealed if the protocol's schedule depends on the choice's outcome. As a solution, [34] proposes to make random choices invisible to the scheduler by annotating protocol actions with labels and requiring that (a) the possible actions after a secret random choice obtain the same label and (b) the only input to the scheduler are the labels of the schedulable actions. The development is based on the probabilistic process algebra CCS_p. It differs from our approach as it requires program annotations that guide the possible choices of the scheduler.

7 Conclusion

Scheduler-independent information flow security is an important problem for concurrent programs, but previously existing solutions are far from being satisfactory: They are either very restrictive in the sense that they reject many intuitively secure programs, or in the sense that they require non-standard modifications of schedulers and their interfaces. Both restrictions limit the applicability of information flow security analyses for concurrent programs.

Aiming at more widely applicable information flow analyses, we developed the novel security condition FSI-security. FSI-security overcomes deficiencies of the existing approaches to scheduler-independent security while still achieving scheduler independence for common schedulers. Our scheduler independence result is rather surprising in the light of the impossibility result from [10], which states that for a natural class of schedulers a compositional scheduler-independent security condition must be at least as restrictive as the strong security condition. The key insight for obtaining a security property that is less restrictive yet compositional and scheduler-independent for relevant schedulers was the identification of a different class of schedulers, the *robust schedulers*, which is also natural but smaller than the class in [10].

Acknowledgments. The authors thank Dave Sands for helpful comments in the early phase of this research project and the anonymous reviewers for their suggestions. This work was funded by the DFG (German Research Foundation) in the Computer Science Action Program. This article reflects only the authors' views, and the DFG and the authors are not liable for any use that may be made of the information contained therein.

References

1. Goguen, J.A., Meseguer, J.: Security Policies and Security Models. In: 3rd IEEE Symposium on Security and Privacy, pp. 11–20. IEEE, Los Alamitos (1982)
2. Jacob, J.: On the Derivation of Secure Components. In: 10th IEEE Symposium on Security and Privacy, pp. 242–247. IEEE, Los Alamitos (1989)
3. Volpano, D., Smith, G.: Probabilistic Noninterference in a Concurrent Language. Journal of Computer Security 7(2,3), 231–253 (1999)
4. Russo, A., Hughes, J., Naumann, D.A., Sabelfeld, A.: Closing Internal Timing Channels by Transformation. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 120–135. Springer, Heidelberg (2008)
5. Smith, G., Volpano, D.: Secure Information Flow in a Multi-threaded Imperative Language. In: 25th ACM Symposium on Principles of Programming Languages, pp. 355–364. ACM, New York (1998)
6. Zdancewic, S., Myers, A.C.: Observational Determinism for Concurrent Program Security. In: 16th IEEE Computer Security Foundations Workshop, pp. 29–43. IEEE, Los Alamitos (2003)
7. Sabelfeld, A., Sands, D.: Probabilistic Noninterference for Multi-threaded Programs. In: 13th IEEE Computer Security Foundations Workshop, pp. 200–214. IEEE, Los Alamitos (2000)
8. Boudol, G., Castellani, I.: Noninterference for Concurrent Programs and Thread Systems. Theoretical Computer Science 281(1-2), 109–130 (2002)
9. Russo, A., Sabelfeld, A.: Securing Interaction between Threads and the Scheduler. In: 19th IEEE Computer Security Foundations Workshop, pp. 177–189. IEEE, Los Alamitos (2006)
10. Sabelfeld, A.: Confidentiality for Multithreaded Programs via Bisimulation. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 260–274. Springer, Heidelberg (2004)
11. Smith, G.: Probabilistic Noninterference through Weak Probabilistic Bisimulation. In: 16th IEEE Computer Security Foundations Workshop, pp. 3–13. IEEE, Los Alamitos (2003)
12. Sabelfeld, A., Sands, D.: A Per Model of Secure Information Flow in Sequential Programs. In: Swierstra, S.D. (ed.) ESOP 1999. LNCS, vol. 1576, pp. 50–59. Springer, Heidelberg (1999)
13. Volpano, D., Smith, G., Irvine, C.: A Sound Type System for Secure Flow Analysis. Journal of Computer Security 4(2,3), 167–188 (1996)
14. Mantel, H., Sands, D.: Controlled Declassification Based on Intransitive Noninterference. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 129–145. Springer, Heidelberg (2004)
15. Smith, G.: A New Type System for Secure Information Flow. In: 14th IEEE Computer Security Foundations Workshop, pp. 115–125. IEEE, Los Alamitos (2001)
16. Matos, A.A., Boudol, G., Castellani, I.: Typing Noninterference for Reactive Programs. Journal of Logic and Algebraic Programming 72(2), 124–156 (2007)

17. Sabelfeld, A., Myers, A.C.: Language-based Information-Flow Security. IEEE Journal on Selected Areas in Communication 21(1), 5–19 (2003)
18. Sabelfeld, A.: The Impact of Synchronisation on Secure Information Flow in Concurrent Programs. In: Bjørner, D., Broy, M., Zamulin, A.V. (eds.) PSI 2001. LNCS, vol. 2244, pp. 225–239. Springer, Heidelberg (2001)
19. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure Information Flow by Self-Composition. In: 17th IEEE Computer Security Foundations Workshop, pp. 100–114. IEEE, Los Alamitos (2004)
20. Mantel, H., Sudbrock, H., Kraußer, T.: Combining Different Proof Techniques for Verifying Information Flow Security. In: Puebla, G. (ed.) LOPSTR 2006. LNCS, vol. 4407, pp. 94–110. Springer, Heidelberg (2007)
21. Mantel, H., Reinhard, A.: Controlling the What and Where of Declassification in Language-Based Security. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 141–156. Springer, Heidelberg (2007)
22. Russo, A., Sabelfeld, A.: Security for Multithreaded Programs under Cooperative Scheduling. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 474–480. Springer, Heidelberg (2007)
23. McLean, J.D.: Proving Noninterference and Functional Correctness using Traces. Journal of Computer Security 1(1), 37–57 (1992)
24. Roscoe, A.W., Woodcock, J.C.P., Wulf, L.: Non-interference through Determinism. In: Gollmann, D. (ed.) ESORICS 1994. LNCS, vol. 875, pp. 33–53. Springer, Heidelberg (1994)
25. Roscoe, A.W.: CSP and Determinism in Security Modelling. In: 16th IEEE Symposium on Security and Privacy, pp. 114–127. IEEE, Los Alamitos (1995)
26. Huisman, M., Worah, P., Sunesen, K.: A Temporal Logic Characterisation of Observational Determinism. In: 19th IEEE Computer Security Foundations Workshop, pp. 3–15. IEEE, Los Alamitos (2006)
27. Mantel, H., Sabelfeld, A.: A Unifying Approach to the Security of Distributed and Multi-threaded Programs. Journal of Computer Security 11(4), 615–676 (2003)
28. Focardi, R., Rossi, S., Sabelfeld, A.: Bridging Language-Based and Process Calculi Security. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 299–315. Springer, Heidelberg (2005)
29. Köpf, B., Mantel, H.: Transformational Typing and Unification for Automatically Correcting Insecure Programs. International Journal of Information Security 6(2-3), 107–131 (2007)
30. Lux, A., Mantel, H.: Declassification with Explicit Reference Points. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 69–85. Springer, Heidelberg (2009)
31. Barthe, G., Rezk, T., Russo, A., Sabelfeld, A.: Security of Multithreaded Programs by Compilation. In: Biskup, J., Lopez, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 2–18. Springer, Heidelberg (2007)
32. Russo, A., Sabelfeld, A.: Securing Interaction between Threads and the Scheduler in the Presence of Synchronization. Journal of Logic and Algebraic Programming 78(7), 593–618 (2009)
33. van der Meyden, R., Zhang, C.: Information Flow in Systems with Schedulers. In: 21st IEEE Computer Security Foundations Symposium, pp. 301–312. IEEE, Los Alamitos (2008)
34. Chatzikokolakis, K., Palamidessi, C.: Making Random Choices Invisible to the Scheduler. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 42–58. Springer, Heidelberg (2007)

Secure Multiparty Linear Programming Using Fixed-Point Arithmetic

Octavian Catrina[1] and Sebastiaan de Hoogh[2]

[1] Dept. of Computer Science, University of Mannheim, Germany
[2] Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract. Collaborative optimization problems can often be modeled as a linear program whose objective function and constraints combine data from several parties. However, important applications of this model (e.g., supply chain planning) involve private data that the parties cannot reveal to each other. Traditional linear programming methods cannot be used in this case. The problem can be solved using cryptographic protocols that compute with private data and preserve data privacy. We present a practical solution using multiparty computation based on secret sharing. The linear programming protocols use a variant of the simplex algorithm and secure computation with fixed-point rational numbers, optimized for this type of application. We present the main protocols as well as performance measurements for an implementation of our solution.

Keywords: Secure multiparty computation, linear programming, secure fixed-point arithmetic, secret sharing.

1 Introduction

The optimization of processes involving multiple parties can often be formulated as a collaborative linear programming problem: minimize (or maximize) a linear objective function subject to a set of linear constraints, where the function and the constraints are defined by combining data from all parties. This linear program may include confidential data that the parties cannot reveal to each other. For example, a linear program for supply chain planning uses business data whose disclosure has negative effects on the participant's negotiation position and competition with other suppliers (e.g., production costs and available capacity) [11]. The supply chain partners cannot use traditional methods to solve the linear program, since this would reveal their confidential data.

Secure computation preserves input privacy using cryptographic protocols. Roughly speaking, the protocols ensure that the output is correct and the computation is carried out without revealing anything else besides the agreed upon output. However, the high communication and computation overhead of cryptographic protocols makes secure computation slower than usual computation with public data. Moreover, finding efficient protocols for complex applications like linear programming is a particularly challenging task. The solutions proposed so far rely on variants of the simplex algorithm that use integer arithmetic [13,19].

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 134–150, 2010.

For non-trivial linear programs, these algorithms require computation with very large integers (thousands of bits) and the protocols become impractical. Our goal is to obtain more efficient protocols, suitable for practical applications.

Our contributions. We take a different approach to secure multiparty simplex, by using computation with rational numbers in fixed-point representation, and we provide a complete solution (all building blocks) as well as performance measurements with a prototype implementation. The protocols are structured into three main layers. The core layer consists of protocols for secure arithmetic in a field and generation of secret random values. This layer could be instantiated using different secure computation methods (e.g., secret sharing or homomorphic encryption). We use multiparty computation based on secret sharing (semi-honest model), which offers the most efficient protocols. The arithmetic layer offers protocols for computation with boolean, integer, and rational (fixed-point) data types. Finally, the protocols in the application layer carry out an oblivious computation of the simplex algorithm with secret-shared input and output (a linear program and its solution). The simplex protocol leaks only the number of iterations and the termination condition (optimal solution or unbounded problem).

The complexity of a secure simplex iteration is dominated by several steps that consist of many secure comparisons or multiplications executed in parallel. Our approach to improving the performance of the protocol focuses on reducing the communication complexity of these steps. The protocol is based on a simplex variant that uses fixed-point arithmetic and needs a minimum number of comparisons and fixed-point multiplications. The design of the lower layer protocols, the data encoding, the use of secure fixed-point arithmetic, and the design of the simplex protocol contribute to achieving this goal.

Related Work. Secure linear programming protocols were proposed by Li and Atallah [13] for the two-party case and by Toft [19] for the multiparty case. Heuristics that apply simplex to "disguised" linear programs have both correctness flaws and security problems [1], so we do not discuss them in the following. We review the relevant features of the solutions in [13,19].

Both protocols use secure integer computation and the simplex algorithm. Let ℓ denote the bit-length of the integers in the initial tableau of the linear program. The first protocol [13] is based on a simplified variant of simplex, without divisions. An iteration of this algorithm can double the bit-length of the values in the tableau ($k = 2^\theta \ell$ bits after θ iterations, worst case). Therefore, the protocol can solve only small linear programs that terminate in few iterations.

Toft's protocol [19] uses a simplex variant [15] with the property that the divisions computed in every iteration yield integer results. This algorithm has important advantages: the computation can be carried out using secure integer arithmetic; the values in the tableau are exact (no rounding errors) and do not grow as fast as in the previous variant; secure division can be efficiently computed in this particular case. However, the values still grow during the initial iterations (up to $k = \theta\ell$ bits after θ iterations, worst case) and the growth levels off at large bit-lengths, reaching thousands of bits for practical problems. This

severely degrades the performance and limits the practical applications, since the protocol has to use a data encoding that avoids overflow and hides the bit-length variation throughout the computation. In particular, secure comparison becomes impractical for inputs of thousands of bits.

We avoid this drawback by using a simplex variant with small tableau and fixed-point arithmetic. The goal is to reduce the bit-length of the secret shares by a factor of 10 and the input bit-length of all comparisons to 100 bits. Due to the structure of the simplex algorithm the gains exceed by far the effects of more complex fixed-point arithmetic. We use our general framework for secure fixed-point computation introduced in [4], extending and adapting the protocols for this type of application. In particular we use a different division protocol that allows to efficiently compute many division operations with common divisor, so that a simplex iteration computes a single reciprocal. Rounding errors for an iteration are close to the resolution of the fixed-point representation.

Oblivious computation of simplex iterations is achieved in [13,19] using two different methods. Li and Atallah use secret permutations of the rows and columns of the tableau in each iteration. Toft introduces a secret indexing method that allows to read or write entries in the tableau without revealing the index. Our protocol uses secret indexing, which is simpler and more versatile. We give more efficient solutions for secret reading and pivot selection.

We use standard techniques for multiparty computation based on secret sharing, similar to [6,7,14,18]. However, the protocols in [7,18] aim at providing integer computation with perfect privacy and constant round complexity, while our goal is fixed-point computation and lower communication complexity, for more efficient parallel computation. We obtain important performance gains using a combination of techniques that includes additive hiding with statistical privacy (instead of perfect privacy), protocols with logarithmic round complexity (instead of constant round complexity), optimized data encoding (especially for binary values), and non-interactive generation of shared random values [5].

2 Preliminaries

2.1 Linear Programming and the Simplex Algorithm

The simplex algorithm is the most popular method for solving linear programs [2]. Its simple structure and the possibility to parallelize a large part of the computation also makes it the best choice for secure linear programming.

We consider the task of solving the linear program shown in Eq. 1, for $b_1, \ldots, b_m \geq 0$. We start by adding the slack variables $x_{n+1}, \ldots, x_{n+m}$, to transform Eq. 1 to the standard form with equality constraints shown in Eq. 2. A feasible solution is a vector $x_1, \ldots, x_{n+m} \geq 0$ that satisfies the constraints. The goal is to find an optimal solution that also maximizes the objective function. A basis is a set of m indexes corresponding to variables whose coefficients in the constraints are linearly independent vectors. A solution with null values for non-basis variables is called basic solution. Observe that $x_j = 0$ for $j = 1, \ldots, n$ and $x_{n+i} = b_i$ for $i = 1, \ldots, m$ is a basic feasible solution of Eq. 2.

$$\begin{array}{lll} \max & \sum_{j=1}^{n} f_j x_j & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & i = 1, \ldots, m \\ & x_j \geq 0 & j = 1, \ldots, n \end{array} \tag{1}$$

$$\begin{array}{lll} \max & \sum_{j=1}^{n} f_j x_j & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i & i = 1, \ldots, m \\ & x_j \geq 0 & j = 1, \ldots, n+m \end{array} \tag{2}$$

Simplex starts from an initial basic feasible solution and improves it by performing a sequence of iterations until it finds the optimal solution or detects that the linear program is unbounded. In each iteration, a basis variable is replaced by another variable and the linear program is re-written accordingly, using a procedure called pivoting. Simplex uses a tableau representation of the linear program. Two variants of tableau are shown in Fig. 1. The vectors S and U contain the indexes of the current basis and non-basis variables, respectively.

	x_1	$\ldots$	x_n	x_{n+1}	$\ldots$	x_{n+m}	
$x_{S(1)}$	a_{11}	$\ldots$	a_{1n}	1	$\ldots$	0	b_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_{S(m)}$	a_{m1}	$\ldots$	a_{mn}	0	$\ldots$	1	b_m
F	$-f_1$	$\ldots$	$-f_n$	0	$\ldots$	0	0

	$x_{U(1)}$	$\ldots$	$x_{U(n)}$	
$x_{S(1)}$	a_{11}	$\ldots$	a_{1n}	b_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$
$x_{S(m)}$	a_{m1}	$\ldots$	a_{mn}	b_m
F	$-f_1$	$\ldots$	$-f_n$	0

Fig. 1. Initial simplex tableau (left) and condensed tableau variant (right)

The computation can be carried out in many different ways. For secure simplex, the choice of the algorithm depends on the complexity of the building blocks and has a strong impact on performance. We considered different variants of plain and revised simplex. Variants for integer arithmetic work with very large numbers, making secure comparison impractical and increasing the communication overhead. The protocol presented in this paper uses an algorithm for fixed point arithmetic and the condensed tableau in Fig. 1. This variant needs a minimum number of secure comparisons and fixed-point multiplications. The algorithm is described below. We denote $V(i)$ the element of vector V at index i and $M(i,j)$ the element of matrix M at row index i and column index j.

1. *Initialization:* For $i \in [1..m]$, $j \in [1..n]$, set $A(i,j) \leftarrow a_{ij}$, $F(j) \leftarrow f_j$, $B(i) \leftarrow b_i$, $U(j) \leftarrow j$, $S(i) \leftarrow n+i$. Set $T \leftarrow \begin{pmatrix} A & B \\ -F & 0 \end{pmatrix}$.
2. *Iterations:*

a) *Get Pivot Column:* Select $c \in [1..n]$ such that $T(m+1, c) < 0$. If no such c, report "Optimal Solution" and exit. If more options, choose at random or using Bland's rule (minimum $U(c)$).

b) *Get Pivot Row:* Select $r \in [1..m]$, such that $T(r,c) > 0$ and $T(n+1, r)/T(r,c)$ is minimal. If no such r, report "Unbounded Problem" and exit. If more options, choose at random or using Bland's rule (minimum $S(r)$).

c) *Update the tableau (pivoting):*

$$
\begin{array}{lll}
T(i,j) \leftarrow T(i,j) - T(i,c)T(r,j)/T(r,c) & i \in [1..m+1]\backslash\{r\}, j \in [1..n+1]\backslash\{c\} \\
T(i,c) \leftarrow -T(i,c)/T(r,c) & i \in [1..m+1]\backslash\{r\} \\
T(r,j) \leftarrow T(r,j)/T(r,c) & j \in [1..n+1]\backslash\{c\} \\
T(r,c) \leftarrow 1/T(r,c) & \\
U(c) \quad \leftrightarrow S(r) & \text{(swap)}
\end{array}
$$

3. *Final solution:* For $i \in [1..m]$ set $x_{S(i)} \leftarrow T(i, n+1)$. All other variables take the value 0. The objective function takes the value $T(m+1, n+1)$.

2.2 Core Protocols

Basic framework. Assume a group of $n > 2$ parties, $P_1, \ldots, P_n$, that communicate on secure channels. Party P_i has private input x_i and output y_i, function of all inputs. Multiparty computation using secret sharing proceeds as follows. The parties use a linear secret sharing scheme to deliver shares of their private inputs to the group. Thus, they create a distributed state of the computation where each party has a share of each secret variable. Certain subsets of parties can reconstruct a secret by pooling together their shares, while any other subset cannot learn anything about it. The secret sharing scheme allows to compute with shared variables. The protocols used for this purpose take on input shared data and return shared data, and thus enable secure protocol composition.

The protocols offer perfect or statistical privacy, in the sense that the views of protocol execution (all values learned by an adversary) can be simulated such that the distributions of real and simulated views are perfectly or statistically indistinguishable, respectively. Let X and Y be distributions with finite sample spaces V and W and $\Delta(X,Y) = \frac{1}{2}\sum_{v \in V \bigcup W} |Pr(X = v) - Pr(Y = v)|$ the statistical distance between them. We say that the distributions are perfectly indistinguishable if $\Delta(X,Y) = 0$ and statistically indistinguishable if $\Delta(X,Y)$ is negligible in some security parameter.

The basic framework uses Shamir secret sharing over a finite field $\mathbb{F}$ and allows secure arithmetic in $\mathbb{F}$ with perfect privacy against a passive threshold adversary able to corrupt t out of n parties. Essentially, in this model, the parties do not deviate from the specified protocol and any $t+1$ parties can reconstruct a secret, while t or less parties cannot distinguish it from random uniform values in $\mathbb{F}$. We assume $|\mathbb{F}| > n$, to enable Shamir sharing, and $n > 2t$, for multiplication of secret-shared values. We denote $[x]$ a Shamir sharing of x and $[x]^{\mathbb{F}}$ a sharing in a particular field $\mathbb{F}$. We refer the reader to [6] for a more formal and general presentation of this approach to secure computation.

Complexity metrics. The running time of the protocols is (usually) dominated by the communication between parties. We evaluate the complexity of the protocols using two metrics that reflect different aspects of the interaction. Communication complexity measures the amount of data sent by each party. For our protocols, a suitable abstract metric is the number of invocations of a primitive during which every party sends a share (field element) to the others. Round complexity

Table 1. Secure arithmetic in a finite field $\mathbb{F}$

Operation	Purpose	Rounds	Invocations
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} + [b]^{\mathbb{F}}$	Add secrets	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} + b$	Add secret and public	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} b$	Multiply secret and public	0	0
$[c]^{\mathbb{F}} \leftarrow [a]^{\mathbb{F}} [b]^{\mathbb{F}}$	Multiply secrets	1	1
$a \leftarrow \mathsf{Output}([a]^{\mathbb{F}})$	Reveal a secret	1	1
$[z] \leftarrow \mathsf{Inner}([X]^{\mathbb{F}}, [Y]^{\mathbb{F}})$	$[\sum_{i=1}^{m} X(i)Y(i)]^{\mathbb{F}}$	1	1

measures the number of sequential invocations and is relevant for the inherent network delay, independent of the amount of data. Invocations that can be executed in parallel count as a single round.

Shared random values. Secure computation often combines secret sharing with additive or multiplicative hiding. For example, given a shared variable $[x]$ the parties jointly generate a shared random value $[r]$, compute $[y] = [x] + [r]$, and reveal $y = x + r$. This is similar to one-time pad encryption of x with key r.

For a secret $x \in \mathbb{Z}_q$ and random uniform $r \in \mathbb{Z}_q$ we obtain $\Delta(x+r \bmod q, r) = 0$, hence perfect privacy. Alternatively, for $x \in [0..2^k - 1]$, random uniform $r \in [0..2^{k+\kappa}-1]$, and $q > 2^{k+\kappa+1}$ we obtain $\Delta(x+r \bmod q, r) < 2^{-\kappa}$, hence statistical privacy with security parameter κ. This property holds for other distributions of r that can be generated more efficiently. The variant with statistical privacy can substantially simplify the protocols by avoiding wraparound modulo q, although it requires larger q (hence larger shares) for a given data range.

We use Pseudo-random Replicated Secret Sharing (PRSS) [5] to generate without interaction shared random values in $\mathbb{F}$ with uniform distribution and random sharings of 0. Also, we use the integer variant RISS [8] to generate shared random integers in a given interval and the ideas in [9] for share conversions. To enable these techniques, we assume that numbers are encoded in $\mathbb{Z}_q$ and $q > 2^{k+\kappa+\nu+1}$, where k is the required integer bit-length, κ is the security parameter, $\nu = \lceil \log(\binom{n}{t}) \rceil$, n is the number of parties, and t is the corruption threshold.

Efficient inner product. Consider the following common task: given two shared vectors $[X] = ([X(1)], \ldots, [X(m)])$ and $[Y] = ([Y(1)], \ldots, [Y(m)])$, $X, Y \in \mathbb{F}^m$, compute their inner product $[z] = [\sum_{i=1}^{m} X(i)Y(i)]$. A naive solution is to use the multiplication protocol and compute $[z] = \sum_{i=1}^{m} [X(i)][Y(i)]$, with complexity 1 round and m invocations. We present an efficient protocol with perfect privacy and complexity reduced to 1 invocation. Assume Shamir sharing for n parties with threshold $t < n/2$. Denote $[X(i)]_j$, $[Y(i)]_j$, $i \in [1..m]$, the input shares and $[z]_j$ the output share of party P_j. The protocol, called Inner, proceeds as follows:

1. Party P_j, $j \in [1..n]$, computes $d_j = \sum_{i=1}^{m} ([X(i)]_j [Y(i)]_j)$ and then shares d_j sending $[d_j]_k$ to party P_k, $k \in [1..n]$.
2. Party P_k, $k \in [1..n]$, computes the share $[z]_k = \sum_{j \in J} \lambda_j [d_j]_k$, where $J \subseteq [1..n]$, $|J| = 2t + 1$, and $\{\lambda_j\}_{j \in J}$ is the reconstruction vector for J.

Protocol Inner is a generalization of the classical protocol for secure multiplication in a field [12]. The proofs of correctness and security are similar.

Secret indexing. The purpose of secret indexing is to read/write a value from/to an array without revealing the value and its index. Efficient secret indexing can be achieved by encoding an index $x \in [1..m]$ as a secret bitmask $[V] = ([V(1)], \ldots, [V(m)])$ such that $V(i) = 1$ for $i = x$ and $V(i) = 0$ for $i \neq x$ [19]. We use this technique to obtain oblivious computation of simplex iterations.

Protocols 2.1 and 2.2 allow secret reading and writing from/to a vector. The secure multiplications re-randomize the shares, providing perfect privacy. Extension to a matrix is obvious. We call the protocols that read/write a column/row SecReadCol, SecReadRow, SecWriteCol, and SecWriteRow. Protocol Inner reduces the complexity of secret reading to 1 invocation (instead of m) for a vector of length m, and to m (or n) invocations (instead of mn) for an $m \times n$ matrix. This has a significant impact on the complexity of the simplex protocol.

Protocol 2.1: $[s] \leftarrow \mathsf{SecRead}([A], [V])$

```
1 [s] ← Inner([A], [V]) ;                          // 1 rnd, 1 inv
2 return [s];
```

Protocol 2.2: $[A] \leftarrow \mathsf{SecWrite}([A], [V], [s])$

```
1 foreach i ∈ [1..m] do parallel
2     [A(i)] ← [A(i)] + [V(i)] ([s] − [A(i)]);     // 1 rnd, m inv
3 return [A];
```

3 Arithmetic Protocols

Fixed-point representation. Fixed-point numbers are rational numbers represented as a sequence of digits split into integer and fractional parts by a virtual radix point: $\tilde{x} = s \cdot (d_{e-2} \ldots d_0 . d_{-1} \ldots d_{-f})$. For binary digits the value is $\tilde{x} = s \cdot \sum_{i=-f}^{e-2} d_i 2^i$, where $s \in \{-1, 1\}$, e is the length of the integer part (including the sign bit), and f is the length of the fractional part. Denote $\bar{x} = s \cdot \sum_{i=0}^{e+f-2} d_i 2^i$ and observe that $\tilde{x} = \bar{x} \cdot 2^{-f}$, hence $\tilde{x}$ is encoded as an integer $\bar{x}$ scaled by the factor 2^{-f}.

We define a fixed-point data type as follows. Let k, e, and f be integers such that $k > 0$, $f \geq 0$, and $e = k - f \geq 0$. Denote $\mathbb{Z}_{\langle k \rangle} = \{x \in \mathbb{Z} \mid -2^{k-1} + 1 \leq x \leq 2^{k-1} - 1\}$. The fixed-point data type with resolution 2^{-f} and range 2^e is the set $\mathbb{Q}_{\langle k,f \rangle} = \{\tilde{x} \in \mathbb{Q} \mid \tilde{x} = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle k \rangle}\}$. Intuitively, $\mathbb{Q}_{\langle k,f \rangle}$ is obtained by sampling the range of real values $[-2^{e-1} + 2^{-f}, 2^{e-1} - 2^{-f}]$ at 2^{-f} intervals.

Data encoding in a field. Any secret value in a secure computation has a data type which is public information. Data types are encoded in a field $\mathbb{F}$ as follows.

Logical values $false, true$ and bit values $0, 1$ are encoded as 0_F and 1_F, respectively. $\mathbb{F}$ can be a small binary field $\mathbb{F}_{2^m}$ or prime field $\mathbb{Z}_q$. This encoding

Table 2. Secure fixed-point arithmetic: addition and multiplication

	Fixed-point Op.	Integer Op./ Secure Op.	Abs. Error
Add (Subtract)	$\tilde{c} = \tilde{a} + \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$	$\bar{c} = \bar{a} + \bar{b}$ $[c] \leftarrow [a] + [b]$	$\delta = 0$
Multiply w/o scaling	$\tilde{c} = \tilde{a}\tilde{b} \in \mathbb{Q}_{\langle k+f,2f \rangle}$ $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$	$\bar{c} = \bar{a}\bar{b}$ $[c] \leftarrow [a][b]$	$\delta = 0$
Multiply w/ scaling	$\tilde{c} = \tilde{a}\tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$	$\bar{c} = \mathsf{trunc}(\bar{a}\bar{b}, f)$ $[c] \leftarrow \mathsf{TruncPr}([a][b], k+f, f)$	$\delta = \delta_t 2^{-f}$ $\lvert\delta_t\rvert < 1$
Inner product $A = (a_1, \ldots, a_m)$ $B = (b_1, \ldots, b_m)$	$\tilde{c} = \sum_{i=1}^{m} \tilde{a}_i \tilde{b}_i$ $\tilde{a}_i, \tilde{b}_i, \tilde{c} \in \mathbb{Q}_{\langle k,f \rangle}$	$\bar{c} = \mathsf{trunc}(\sum_{i=1}^{m} \bar{a}_i \bar{b}_i, f)$ $[x] \leftarrow \mathsf{Inner}([A], [B])$ $[c] \leftarrow \mathsf{TruncPr}([x], k+f, f)$	$\delta = \delta_t 2^{-f}$ $\lvert\delta_t\rvert < 1$
Multiply double optim.	$\tilde{d} = \tilde{a}\tilde{b}\tilde{c} \in \mathbb{Q}_{\langle k,f \rangle}$ $\tilde{a}, \tilde{b}, \tilde{c}, \tilde{d} \in \mathbb{Q}_{\langle k,f \rangle}$	$\bar{d} = \mathsf{trunc}(\bar{a}\bar{b}\bar{c}, 2f)$ $[c] \leftarrow \mathsf{TruncPr}([a][b][c], k+2f, 2f)$	$\delta = \delta_t 2^{-f}$ $\lvert\delta_t\rvert < 1$

allows secure evaluation of boolean functions using secure arithmetic in $\mathbb{F}$. Efficient encoding of binary values is essential for reducing the complexity of shared random bit generation, comparison, and other secure simplex building blocks.

Signed integers are encoded in $\mathbb{Z}_q$ using $\mathsf{fld} : \mathbb{Z}_{\langle k \rangle} \mapsto \mathbb{Z}_q$, $\mathsf{fld}(\bar{x}) = \bar{x} \bmod q$, $q > 2^k$. For any $\bar{a}, \bar{b} \in \mathbb{Z}_{\langle k \rangle}$ and $\odot \in \{+, -, \cdot\}$ we have $\bar{a} \odot \bar{b} = \mathsf{fld}^{-1}(\mathsf{fld}(\bar{a}) \odot \mathsf{fld}(\bar{b}))$. Moreover, if $\bar{b} | \bar{a}$ then $\bar{a}/\bar{b} = \mathsf{fld}^{-1}(\mathsf{fld}(\bar{a}) \cdot \mathsf{fld}(\bar{b})^{-1})$. Secure arithmetic with signed integers can thus be computed using secure arithmetic in $\mathbb{Z}_q$.

A fixed-point number $\tilde{x} \in \mathbb{Q}_{\langle k,f \rangle}$ is represented as a secret integer $\bar{x} = \tilde{x} 2^f$ encoded in $\mathbb{Z}_q$ and public parameters that specify the resolution and the range, f and e (or $k = e + f$). We define the map $\mathsf{int}_f : \mathbb{Q}_{\langle k,f \rangle} \mapsto \mathbb{Z}_{\langle k \rangle}$, $\mathsf{int}_f(\tilde{x}) = \tilde{x} 2^f$.

We distinguish different representations of a number using the following simplified notation: we denote $\tilde{x}$ a rational number of some fixed-point type $\mathbb{Q}_{\langle k,f \rangle}$ and $\bar{x} = \tilde{x} 2^f \in \mathbb{Z}_{\langle k \rangle}$ the integer value of its fixed-point representation; for secure computation using secret-sharing we denote $x = \bar{x} \bmod q \in \mathbb{Z}_q$ the field element that encodes $\bar{x}$ (and hence $\tilde{x}$) and $[x]$ a sharing of x. The notation $x = (\texttt{condition})?\ a : b$ means that the variable x is assigned the value a when `condition`=*true* and b otherwise.

Fixed-point arithmetic. Tables 2 and 3 contain a summary of the main arithmetic protocols used in simplex. Secure addition, subtraction, and comparison of fixed-point numbers are immediate extensions of the integer operations. We need additional protocols for multiplication and division.

Multiplication. Table 2 shows several cases of secure fixed-point multiplication used in simplex. Let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k,f \rangle}$ and $\tilde{c} = \tilde{a}\tilde{b}$. We obtain the representation of $\tilde{c}$ with resolution 2^{-2f} by integer multiplication, $\bar{c} = \bar{a}\bar{b} = \tilde{a}\tilde{b}2^{2f}$, and we can scale down $\tilde{c}$ to resolution 2^{-f} by truncation (when necessary). The truncation protocol $\mathsf{TruncPr}$ computes $\bar{c}/2^f$ and rounds to the nearest integer with probability $1 - \alpha$, where α is the distance to that integer [4]. The absolute error is with high

Table 3. Complexity of arithmetic protocols ($\log(q_1) \approx \kappa$)

Operation	Protocol	Rounds	Invocations	Field
$\lfloor \bar{a}/2^f \rfloor + u$	$[d] \leftarrow \mathsf{TruncPr}([a], k, f)$	1	1	$\mathbb{Z}_q$
$\bar{a} \in \mathbb{Z}_{\langle k \rangle}$, $u \in_R \{0,1\}$		2	$2f$	$\mathbb{Z}_{q_1}$
	TruncPr after precomp.	1	1	$\mathbb{Z}_q$
	$[d] \leftarrow \mathsf{TruncPrN}([a], k, f)$	1	1	$\mathbb{Z}_q$
$(\bar{a} < 0)?1 : 0$, $\bar{a} \in \mathbb{Z}_{\langle k \rangle}$	$[s] \leftarrow \mathsf{LTZ}([a], k)$	1	1	$\mathbb{Z}_q$
$(\bar{a} > 0)?1 : 0$, $\bar{a} \in \mathbb{Z}_{\langle k \rangle}$	$\mathsf{GTZ}([a], k) = \mathsf{LTZ}(-[a], k)$	2	$2k$	$\mathbb{Z}_{q_1}$
		$\log(k) + 1$	$2k - 3$	$\mathbb{F}_{2^8}$
	LTZ after precomp.	1	1	$\mathbb{Z}_q$
		$\log(k) + 1$	$2k - 3$	$\mathbb{F}_{2^8}$
$\tilde{x} \approx 1/\tilde{c} \in \mathbb{Q}_{\langle p+1,p \rangle}$	$[x] \leftarrow \mathsf{RecItNR}([c], p)$	3θ	3θ	$\mathbb{Z}_q$
$\tilde{c} \in \mathbb{Q}_{\langle p+1,p \rangle} \bigcap (0.5, 1)$	(using TruncPr)	2	$2p\theta$	$\mathbb{Z}_{q_1}$
	RecItNR after precomp.	3θ	3θ	$\mathbb{Z}_q$
Normalization: $\bar{c} = \bar{v}\bar{x}$,	$([c], [v]) \leftarrow \mathsf{Norm}([x], k, f, p)$	3	3	$\mathbb{Z}_q$
$\tilde{x} \in \mathbb{Q}_{\langle k+1,f \rangle}$,	(using TruncPr)	$2^\dagger$	$6k - 2p$	$\mathbb{Z}_{q_1}$
$\tilde{c} \in \mathbb{Q}_{\langle p+1,p \rangle} \bigcap (0.5, 1)$		$2\log(k) + 1$	$k + 1.5k\log(k)$	$\mathbb{F}_{2^8}$
	Norm after precomp.	3	3	$\mathbb{Z}_q$
		$2\log(k) + 1$	$k + 1.5k\log(k)$	$\mathbb{F}_{2^8}$

probability $|\delta_t| \leq 0.5$, and always $|\delta_t| < 1$. TruncPr provides statistical privacy, while TruncPrN performs the same operation more efficiently but with weaker protection of the discarded part (additive hiding with non-uniform random). TruncPrN is sufficient for multiplications in simplex, since values less than 2^{-f} are negligible and the computation is carried out with extended precision (large f). Note that the optimizations for inner product and double multiplication shown in Table 2 are also important for improving the accuracy.

Reciprocal and division. Simplex needs an accurate and efficient protocol for multiple division operations with the same positive divisor, $\tilde{a}_1/\tilde{b}, \ldots, \tilde{a}_m/\tilde{b}$. This can be achieved by computing $\tilde{y} = 1/\tilde{b}$ followed by m parallel multiplications $\tilde{z}_i = \tilde{a}_i\tilde{y}$. Protocols 3.1, RecItNR, and 3.2, DivNR, follow this approach (the division protocol in [4] is not suitable for this type of application).

Let $\tilde{c} \in \mathbb{Q}_{\langle p+1,p \rangle} \bigcap (0.5, 1)$. RecItNR computes $\tilde{x} \approx 1/\tilde{c}$, $\tilde{x} \in \mathbb{Q}_{\langle p+1,p \rangle}$, for secret-shared input and output. The protocol uses the Newton-Raphson method and starts by computing the initial approximation $\tilde{x}_0 \approx 1/\tilde{c}$, $\tilde{x}_0 = 2.9142 - 2\tilde{c}$, with relative error $\epsilon_0 < 0.08578$ (at least $\log_2(\epsilon_0) = 3.5$ exact bits) [10]. Each iteration computes an improved approximation $\tilde{x}_{i+1} = \tilde{x}_i(2 - \tilde{x}_i\tilde{c})$. For exact arithmetic (without truncation) the relative error after iteration i is $\epsilon_i = \epsilon_{i-1}^2 = \epsilon_0^{2^i}$. Intuitively, the number of exact bits doubles at each iteration, so $p + 1$ bits ($\delta < 2^{-p}$) are obtained after $\theta = \lceil \log \frac{p+1}{3.5} \rceil$ iterations. The error due to computation of an iteration with limited precision is $|\delta_T| < 2^{-p}$. Since the error introduced by an iteration decreases quadratically during next iterations, we conclude that the output error of RecItNR is approximately bounded by 2^{-p}.

Let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\langle k+1,f\rangle}$ and $\tilde{b} > 0$. Protocol DivNR computes the quotient $\tilde{z} \approx \tilde{a}/\tilde{b}$, $\tilde{z} \in \mathbb{Q}_{\langle k+1,f\rangle}$, with secret-shared inputs and output. The protocol consists of the following main steps: compute the normalized divisor $\tilde{c} \in \mathbb{Q}_{\langle p+1,p\rangle} \bigcap (0.5, 1)$ using the protocol Norm; compute the reciprocal $\tilde{x} \approx 1/\tilde{c}$ using the protocol RecItNR; then compute the quotient $\tilde{z} \approx \tilde{a}/\tilde{b}$ and scale it to obtain $\tilde{z} \in \mathbb{Q}_{\langle k+1,f\rangle}$. A variant for multiple divisors repeats steps 4 and 5 for each divisor (parallel computation).

Protocol 3.1: $[x] \leftarrow \mathsf{RecItNR}([c], p)$

1 $(\theta, \alpha, \beta) \leftarrow (\lceil \log \frac{p+1}{3.5} \rceil, \mathsf{fld}(\mathsf{int}_p(2.9142)), \mathsf{fld}(\mathsf{int}_{2p}(2.0)))$;
2 $[x] \leftarrow \alpha - 2[c]$;
3 **foreach** $i \in [1..\theta]$ **do**
4 $\quad [x] \leftarrow [x](\beta - [x][c])$;
5 $\quad [x] \leftarrow \mathsf{TruncPr}([y], 3p, 2p)$;
6 **return** $[x]$;

Protocol 3.2: $[z] \leftarrow \mathsf{DivNR}([a], [b], k, f)$

1 $([c], [v]) \leftarrow \mathsf{Norm}([b], k)$;
2 $[x] \leftarrow \mathsf{RecItNR}([c], k)$;
3 $[y] \leftarrow [v][x]$;
4 $[z] \leftarrow [a][y]$;
5 $[z] \leftarrow \mathsf{TruncPr}([z], 3k - f, 2k - f)$;
6 **return** $[z]$;

Protocol Norm computes $\bar{c}$ and $\bar{v}$ such that $2^{k-1} \leq \bar{c} < 2^k$ and $\bar{c} = \bar{b}\bar{v}$, with secret-shared input and outputs [4,17]. Let $\tilde{c} = \bar{c}2^{-k}$ and let $0 < m \leq k$ such that $2^{m-1} \leq \bar{b} < 2^m$. Observe that $\bar{v} = 2^{k-m}$, $\tilde{c} \in \mathbb{Q}_{\langle k+1,k\rangle} \bigcap (0.5, 1)$, and $\tilde{c} = \tilde{b}2^{f-m}$; $\tilde{c}$ is the normalized input for RecItNR and $\bar{v}$ the normalization factor. Steps 3-4 of DivNR compute $\tilde{z} = \tilde{a}\tilde{x}2^{f-m} \approx \tilde{a}/\tilde{b}$ without loss of accuracy and then step 4 scales this value to obtain $\tilde{z} \in \mathbb{Q}_{\langle k+1,f\rangle}$. Observe that $\tilde{a}\tilde{x}2^{f-m}2^f = \bar{a}\bar{x}\bar{v}2^{-(2k-f)}$, so the output $\bar{z} = \mathsf{trunc}(\bar{a}\bar{x}\bar{v}, 2k - f)$ is the representation of $\tilde{z}$ with resolution 2^{-f}. The output error of DivNR is upper bounded by 2^{-f}.

RecItNR and DivNR do not open any secret-shared value and their building blocks provide perfect or statistical privacy. The number of iterations depends only on public configuration parameters, hence it can be revealed. We conclude that the two protocols offer statistical privacy.

4 Secure Simplex Protocol

Protocol 4.1, Simplex, solves linear programs using the algorithm and the building blocks presented in the previous sections. The inputs are the secret-shared values of the linear program: the matrix $[A]$ and the vectors $[B]$ and $[F]$. The output consists of a public value indicating the termination state, optimal or unbounded, a secret-shared array $[X]$ containing the solution, and the optimum $[z]$ of the objective function. The protocol reveals only the number of iterations and

the termination condition. The tableau, pivot indexes, and related variables are protected throughout the computation using the techniques discussed in Section 2.2. For a passive adversary that corrupts $t < n/2$ parties, the building blocks provide perfect or statical privacy. By the composition theorem in Chapter 4 of [3] we conclude that the simplex protocol provides statistical privacy.

Complexity is shown in the protocol specifications by annotating the relevant steps; for better clarity, we assume a generic comparison protocol with complexity ρ rounds and μ invocations. For a vector V and matrix M we denote: $V(i..j) = (V(i), \ldots, V(j))$; $M(i, \cdot)$ the row i; $M(\cdot, j)$ the column j. Angle brackets are used to specify the number of elements, e.g., $V\langle m\rangle$, $M\langle m, n\rangle$.

Protocol 4.1 initializes the tableau $[T]$ and the basis and non-basis index vectors $[S]$ and $[U]$, performs the simplex iterations, and then extracts from the final tableau the optimal solution (if it exists). The iterations are computed by Protocol 4.2. The computation is structured into several sub-protocols that select the pivot's column and row (GetPivCol and GetPivRow) and then update the tableau $[T]$ (UpdTab) and the vectors $[S]$ and $[U]$ (UpdVar).

Protocol 4.1: $(result, [X], [z]) \leftarrow \mathsf{Simplex}([A], [B], [F])$

Input: $[A\langle m, n\rangle]$, $[B\langle m\rangle]$, $[F\langle m\rangle]$.
Output: $result \in \{\mathsf{Opt}, \mathsf{Unb}\}$; $[X\langle n\rangle]$ and $[z]$ if $result = \mathsf{Opt}$.

1 $[T] \leftarrow \begin{pmatrix} [A] & [B] \\ -[F] & [0] \end{pmatrix}$;
2 $([S], [U]) \leftarrow \mathsf{InitVar}(m, n)$;
3 $([T], [S], result) \leftarrow \mathsf{Iteration}([T], [S], [U])$;
4 **if** $result = \mathsf{Unb}$ **then return** Unb;
5 $[X] \leftarrow \mathsf{GetSolution}([T(\cdot, n+1)], [S])$;
6 **return** $(\mathsf{Opt}, [X], [T(m+1, n+1)])$;

Protocol 4.2: $([T], [S], result) \leftarrow \mathsf{Iteration}([T], [S], [U])$

Input: $[T\langle m+1, n+1\rangle]$, $[S\langle m\rangle]$, $[U\langle n\rangle]$;
Output: $[T\langle m+1, n+1\rangle]$, $[S\langle m\rangle]$; $result \in \{\mathsf{Opt}, \mathsf{Unb}\}$;

1 **repeat forever**
2 $([V], s) \leftarrow \mathsf{GetPivCol}([T(m+1, 1..n)])$; // protocol 4.5
3 **if** $s = 0$ **then return** $([T], [S], \mathsf{Opt})$;
4 $[C] \leftarrow \mathsf{SecReadCol}([T], [V])$; // 1 rnd, $m+1$ inv
5 $([W], s) \leftarrow \mathsf{GetPivRow}([T(1..m, n+1)], [C])$; // protocol 4.6
6 **if** $s = 0$ **then return** $([T], [S], \mathsf{Unb})$;
7 $[R] \leftarrow \mathsf{SecReadRow}([T], [W])$; // 1 rnd, $n+1$ inv
8 $[p] \leftarrow \mathsf{SecRead}([R], [V])$; // 1 rnd, 1 inv
9 $[T] \leftarrow \mathsf{UpdTab}([T], [C], [R], [V], [W], [p])$; // protocol 4.8
10 $([S], [U]) \leftarrow \mathsf{UpdVar}([S], [U], [V], [W])$; // 2 rnd, $m+n+2$ inv
11 **end**

The iterations terminate when the algorithm finds the optimal solution or determines that the linear program is unbounded. Termination is detected by the pivot selection protocols, which report that no pivot exists ($s = 0$). Protocol 4.3,

GetSolution, extracts the solution from the tableau by assigning $X(S(i)) \leftarrow B(i)$ for $i \in [1..m]$ and 0 to the other elements; it uses secret indexing and the protocol Int2BitMask [17] that converts secret integers to secret bitmasks.

Protocol 4.3: $[X] \leftarrow \mathsf{GetSolution}([B], [S])$

Input: $[B\langle m\rangle]$, $[S\langle m\rangle]$;
Output: $[X\langle n+m\rangle]$;
1 **foreach** $i \in [1..n+m]$ **do** $[X(i)] \leftarrow 0$;
2 **foreach** $i \in [1..m]$ **do parallel**
3 $[V] \leftarrow \mathsf{Int2BitMask}([S(i)], m+n)$; // 3 rnd, $3(m+n)-4$ inv
4 $\mathsf{SecWrite}([X], [V], [B(i)])$; // 1 rnd, $m+n$ inv
5 **return** $[X]$; // decision variables: $[X(1)], \ldots, [X(n)]$

The vectors $[S]$ and $[U]$ are initialized by InitVar with the indexes of the initial basis and non-basis variables. At each iteration, the basis variable with index $[S([W])]$ is replaced by the non-basis variable with index $[U([V])]$. Protocol 4.4 updates $[S]$ and $[U]$ by swapping the corresponding entries.

Protocol 4.4: $([S], [U]) \leftarrow \mathsf{UpdVar}([S], [U], [V], [W])$

Input: $[S\langle m\rangle]$, $[U\langle n\rangle]$, $[V\langle n\rangle]$, $[W\langle m\rangle]$
Output: $[S\langle m\rangle]$, $[U\langle n\rangle]$ (updated)
1 $[s] \leftarrow \mathsf{SecRead}([S], [W])$; // 1 rnd, 1 inv
2 $[u] \leftarrow \mathsf{SecRead}([U], [V])$; // + 1 inv
3 $[S] \leftarrow \mathsf{SecWrite}([S], [W], [u])$; // 1 rnd, m inv
4 $[U] \leftarrow \mathsf{SecWrite}([U], [V], [s])$; // + n inv
5 **return** $([S], [U])$;

Pivot selection. Protocol 4.5, GetPivCol, finds the index of the pivot's column by selecting a negative entry in the cost vector F; it returns a public bit s indicating if the pivot column was found or not and a secret bitmask $[V]$ that encodes the column's index. If none of the F values is negative then $s = 0$; simplex has found the optimal solution and terminates. Otherwise, $s = 1$ and $[V]$ encodes the index of the first negative entry[1].

Protocol 4.5: $([V], s) \leftarrow \mathsf{GetPivCol}([F])$

Input: $[F\langle n\rangle]$;
Output: $[V\langle n\rangle]$, $s \in \{0, 1\}$;
1 **foreach** $i \in [1..n]$ **do parallel**
2 $[D(i)] \leftarrow \mathsf{LTZ}([F(i)], k)$; // ρ rnd, $n\mu$ inv
3 $s \leftarrow 1 - \mathsf{EQZPub}(\sum_{i=0}^{n}[D(i)])$; // 1 rnd, 1 inv
4 **if** $s = 0$ **then return** $([D], s)$;
5 $[V] \leftarrow \mathsf{SelectFirst}([D])$; // $\log(n)$ rnd, $n\log(n)/2$ inv
6 **return** $([V], s)$;

[1] An implementation of the pivot selection protocols has to take into account the roundoff errors, e.g., by evaluating $\mathsf{LTZ}([a]+\delta, k)$ instead of $\mathsf{LTZ}([a], k)$ where $\delta > 0$ is an estimate of the maximum error.

c_0	c_1	$c_0 b_1'$	$c_1 b_0'$	Output	Constraints	Selection
≤ 0	≤ 0	≤ 0	≤ 0	1	None applicable	b_0/c_0
				0	None applicable	b_1/c_1
> 0	≤ 0	> 0	≤ 0	1	b_1/c_1 not applicable	b_0/c_0
≤ 0	> 0	≤ 0	> 0	0	b_0/c_0 not applicable	b_1/c_1
> 0	> 0	≥ 0	≥ 0	1	$b_0/c_0 < b_1/c_1$	b_0/c_0
				0	$b_1/c_1 \leq b_0/c_0$	b_1/c_1

Fig. 2. Constraint comparison using CompCons

EQZPub($[v]$) is a simple equality test with public output [16] and returns $(v = 0)?\ 1 : 0$. SelectFirst($[D]$) computes the secret bitmask of the minimum index i such that $D(i) = 1$ [17].

The index of the pivot's row is determined by Protocol 4.6, GetPivRow. The protocol computes $\text{argmin}_i\{\frac{B(i)}{C(i)} \mid C(i) > 0\}$. If none of the C values is strictly positive, it returns $s = 0$; the simplex protocol terminates and reports that the linear program is unbounded. Otherwise, $s = 1$ and $[W]$ is a secret bitmask that encodes the index of the pivot's row.

Protocol 4.6: $[W], s \leftarrow \mathsf{GetPivRow}([B], [C])$

Input: $[B\langle m\rangle]$, $[C\langle m\rangle]$.
Output: $[W\langle m\rangle], s \in \{0, 1\}$.

1 **foreach** $i \in [1..m]$ **do parallel**
2 $\quad [D(i)] \leftarrow \mathsf{GTZ}([C(i)])$; // ρ rnd, $m\mu$ inv
3 $s \leftarrow 1 - \mathsf{EQZPub}(\sum_{i=0}^{m}[D(i)])$; // 1 rnd, 1 inv
4 **if** $s = 0$ **then return** $([D], s)$;
5 **foreach** $i \in [1..m]$ **do parallel**
6 $\quad [B'(i)] \leftarrow [B(i)] + (1 - [D(i)])2^f$;
7 $[W] \leftarrow \mathsf{MinCons}([B'], [C], m)$; // $\lceil \log(m) \rceil(\rho + 3)$ rnd, $(m-1)(\mu+5)$ inv
8 **return** $([W], s)$;

Protocol 4.7: $[s] \leftarrow \mathsf{CompCons}([b_0'], [c_0], [b_1'], [c_1])$

1 $[x] \leftarrow [b_0'][c_1] - [b_1'][c_0]$; // 1 rnd, 2 inv
2 $[s] \leftarrow \mathsf{LTZ}([x], k + f)$; // ρ rounds, μ inv
3 **return** $[s]$;

GetPivRow uses the following method. Steps 1-4 select the relevant constraints by computing the secret bitmask $[D]$, $D(i) = (C(i) > 0)?\ 1 : 0$, $i \in [1..m]$. If D is null the protocol terminates and reports that no pivot row was found. Steps 5-7 compute the secret bitmask $[W]$ that encodes $\text{argmin}_i\{\frac{B'(i)}{C(i)}\}$, where $B'(i) = B(i)$ if $C(i) > 0$ and $B'(i) > 0$ if $C(i) \leq 0$. Replacing $B(i)$ with $B'(i)$ avoids the combination $C(i) \leq 0$ (non-applicable constraint) and $B(i) = 0$ when the constraints are compared by Protocol 4.7, CompCons. The selection done by CompCons is shown in Figure 2. The complexity of CompCons can be reduced by modifying LTZ to scale down the input to resolution 2^{-f} before comparison,

without interaction [16]. The constraint comparison in [19] uses a similar method to avoid division, but for $C(i) \leq 0$ sets $B'(i) = \infty$, i.e., greater than any value, and $C'(i) = 1$. The method used in Protocol 4.6 is more efficient (it eliminates 1 round and $m + n$ invocations) and reduces the risk of overflow.

The protocol MinCons computes $\text{argmin}_i\{\frac{B'(i)}{C(i)}\}$ by combining CompCons and a generic protocol that finds the index of the minimum value in a vector of length m in $\lceil \log(m) \rceil$ steps using $m - 1$ comparisons [19].

Update of the tableau. Protocol 4.8 updates the secret-shared tableau without revealing the position of the pivot. The computation can be carried out for different trade-offs between accuracy and efficiency. The solution shown as Protocol 4.8 achieves low complexity with minimum effects on accuracy (i.e., close to the best accuracy for a given fixed-point representation, $\delta < 2^{-f}$). Division is computed by multiplication with the reciprocal of the pivot as in Protocol 3.2, and with a single final scaling, in order to minimize rounding errors.

The complexity of the protocol is reduced by adapting the algorithm in Section 2.1 as follows. Let r and c be the indexes of the pivot's row and column, respectively, and $p = T(r, c)$. The tableau is updated by computing:

$$R'(c) \leftarrow p + 1; \;\; R'(j) \leftarrow T(r, j), \;\; j \in [1..n + 1] \backslash \{c\};$$
$$C'(r) \leftarrow (p - 1)/p; \;\; C'(i) \leftarrow T(i, c)/p, \;\; i \in [1..m + 1] \backslash \{r\};$$
$$T(i, j) \leftarrow T(i, j) - C'(i)R'(j), \;\; i \in [1..m + 1], \; j \in [1..n + 1].$$

Protocol 4.8: $[T] \leftarrow \mathsf{UpdTab}([T], [C], [R], [V], [W], [p])$

Input: $[T\langle m + 1, n + 1\rangle]$, $[C\langle m + 1\rangle]$, $[R\langle n + 1\rangle]$, $[V\langle n\rangle]$, $[W\langle m\rangle]$; $[p]$.
Output: $[T\langle m + 1, n + 1\rangle]$ (updated).

1 $[y] \leftarrow \mathsf{Rec}([p], k)$; // protocol 4.9
2 $[R'] \leftarrow \mathsf{SecWrite}([R], [V], [p] + 2^f)$; // 1 rnd, n inv
3 $[C'] \leftarrow \mathsf{SecWrite}([C], [W], [p] - 2^f)$; // + m inv
4 **foreach** $i \in [1..m + 1]$ **do parallel**
5 $\quad [C'(i)] \leftarrow [C'(i)][y]$; // 1 rnd, $m + 1$ inv
6 **foreach** $i \in [1..m + 1]$, $j \in [1..n + 1]$ **do parallel**
7 $\quad [T'(i, j)] \leftarrow [C'(i)][R'(j)]$; // 1 rnd, $(m + 1)(n + 1)$ inv
8 $\quad [T'(i, j)] \leftarrow \mathsf{TruncPrN}([T'(i, j)], 3k, 2k)$; // 1 rnd $(m + 1)(n + 1)$ inv
9 $\quad [T(i, j)] \leftarrow [T(i, j)] - [T'(i, j)]$;
10 **return** $[T]$;

Protocol 4.9: $[y] \leftarrow \mathsf{Rec}([p], k)$

1 $([c], [v]) \leftarrow \mathsf{Norm}([p], k)$; // see Table 3
2 $[x] \leftarrow \mathsf{RecItNR}([c], k)$; // see Table 3
3 $[y] \leftarrow [v][x]$; // 1 rnd, 1 inv
4 **return** $[y]$;

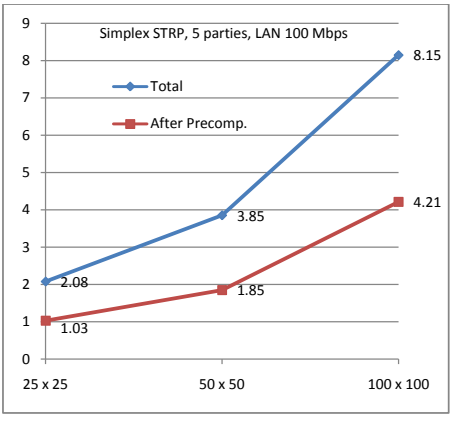
The protocol computes R' and C' in 2 rounds and $2m + n + 1$ invocations (steps 2-5), then $T'(i, j) \leftarrow C'(i)R'(j)$ in 2 rounds and $2(m+1)(n+1)$ invocations (steps 7-8, multiplication and scaling) and, finally, $T(i, j) \leftarrow T(i, j) - T'(i, j)$.

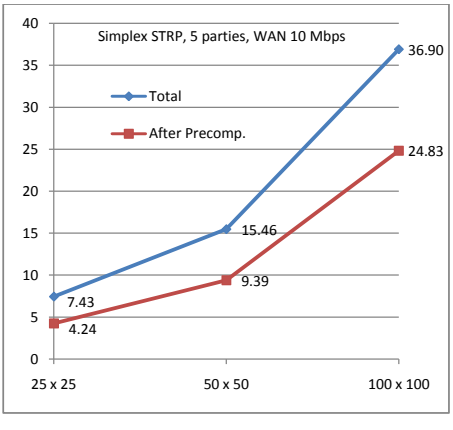
The cost of achieving best accuracy per iteration is a larger modulus, $\log(q) > 3k$. The modulus can be reduced to $\log(q) > 2k$ by scaling down C' before step 7 and/or by computing $1/p$ with precision $k' < k$.

5 Performance Evaluation and Conclusions

We implemented and tested the simplex protocol using our Java libraries for secure computation. We measured the running time of the protocol for five processes (parties) running on different PCs (Intel Core Duo, 1.8 GHz) with full mesh interconnection topology. The experiments were carried out in an isolated network for two settings: Ethernet LAN with 100 Mbps links and WAN with 10 Mbps links. The average round-trip time of the WAN was 40 ms. The LAN experiments show the protocol performance for low network delay, while the WAN experiments show the effects of higher network delay and lower bandwidth.

Figure 3 shows the running time of an iteration for $\log(q) = 288$ bits, $k = 2f = 80$ bits, and linear programs of several sizes: $m = n = 25$, $m = n = 50$, $m = n = 100$. To reduce the number of rounds, all the shared random bits needed by an iteration (for comparisons and reciprocal) are generated in parallel by an initial precomputation phase. Moreover, the running time can be reduced by executing the precomputation in parallel with the previous iteration.





	LAN			WAN		
	25 × 25	50 × 50	100 × 100	25 × 25	50 × 50	100 × 100
Precomputation	1.05	2.00	3.93	3.19	6.08	12.08
Select pivot column	0.22	0.37	0.65	0.67	1.44	2.52
Select pivot row	0.52	0.83	1.40	1.82	3.06	5.41
Update the tableau	0.29	0.65	2.17	1.75	4.88	16.90

Fig. 3. Running time (seconds) for secure simplex iterations

The simplex algorithm in Section 2.1 can be modified to carry out the computation using integer pivoting [15,19]. Correctness and accuracy of our protocol were verified using an implementation (for public data) of the algorithm with integer pivoting, which performs the same pivot operations with exact arithmetic.

A simplex protocol for this variant of the algorithm can easily be obtained by adapting the protocols in Section 4. The main difference is the update of the tableau using secure integer arithmetic instead of fixed-point arithmetic. This protocol is more efficient than the variant in [19] (e.g., $2m + n$ comparisons instead of $3m+n$ and $2mn$ multiplications for the update of the tableau instead of $3m(n + m)$). However, they are both affected by the growth of the values in the tableau, that can reach thousands of bits for linear programs with tens of variables and constraints [19]. The experiments showed a large increase of the running time for pivot selection and precomputation (comparisons) and for the update of the tableau (large shares) even for $\log(q) = 1024$ bits.

The tests show that our approach offers an important performance gain and suitable accuracy for secure linear programming. The main performance bottleneck is the secure comparison. Our comparison protocol [16] provides statistical privacy and performs most of the computation in a small field, hence with low overhead (Table 3). By encoding binary values in small fields and using efficient share conversions, the amount of data exchanged is reduced from $O(k^2)$ bits (when integer and binary values are encoded in the same field) to $O(k)$. The simplex algorithm used by the protocol needs only $2m+n$ comparisons (instead of $3m+n$ when using the large tableau to select the pivot) and fixed-point arithmetic can reduce their input bit-length by a factor of 10 with respect to integer pivoting. Nevertheless, most of the precomputation time and pivot selection time shown in Fig. 3 is due to comparisons. Further performance improvement would require secure comparison with sublinear complexity; currently, in the multiparty setting, this can be achieved only by trading off privacy for efficiency.

The solutions presented in this paper can be applied to other simplex variants (e.g., revised simplex) and to protocols for general linear programs (finding an initial basic feasible solution). The building blocks can be used in other applications with similar requirements (accurate secure computation with rational numbers or computation with many parallel operations). These are topics of ongoing research, in parallel with the improvement of secure fixed-point arithmetic.

Acknowledgements. Part of this work was funded by the European Commission through the grant FP7-213531 to the SecureSCM project. We thank Claudiu Dragulin for his contribution to the implementation of the protocols.

References

1. Bednarz, A., Bean, N., Roughan, M.: Hiccups on the road to privacy-preserving linear programming. In: WPES 2009: Proc. of the 8th ACM Workshop on Privacy in the electronic society, pp. 117–120. ACM, New York (2009)
2. Bertsimas, D., Tsitsiklis, J.: Introduction to Linear Optimization. Athena Scientific, Belmont (1997)

3. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
4. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Financial Cryptography and Data Security. LNCS, Springer, Heidelberg (2010)
5. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (2005)
6. Cramer, R., Damgård, I., Maurer, U.: General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)
7. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)
8. Damgård, I., Thorbek, R.: Non-interactive Proofs for Integer Multiplication. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 412–429. Springer, Heidelberg (2007)
9. Damgard, I., Thorbek, R.: Efficient conversion of secret-shared values between different fields. Cryptology ePrint Archive, Report 2008/221 (2008)
10. Ercegovac, M.D., Lang, T.: Digital Arithmetic. Morgan Kaufmann, San Francisco (2003)
11. Frati, F., Damiani, E., Ceravolo, P., Cimato, S., Fugazza, C., Gianini, G., Marrara, S., Scotti, O.: Hazards in full-disclosure supply chains. In: Proc. 8th Conference on Advanced Information Technologies for Management, AITM 2008 (2008)
12. Gennaro, R., Rabin, M., Rabin, T.: Simplified VSS and fast-track multi-party computations with applications to threshold cryptography. In: Proc. of ACM Symposium on Principles of Distributed Computing, PODC 1998 (1998)
13. Li, J., Atallah, M.: Secure and Private Collaborative Linear Programming. In: Proc. 2nd Int. Conference on Collaborative Computing: Networking, Applications and Worksharing (ColaborateCom 2006), Atlanta, USA, pp. 19–26 (2006)
14. Nishide, T., Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)
15. Rosenberg, G.: Enumeration of All Extreme Equilibria of Bimatrix Games with Integer Pivoting and Improved Degeneracy Check. Research Report LSE-CDAM-2005-18, London School of Economics and Political Science (2005)
16. SecureSCM. Security Analysis. Deliverable D9.2, EU FP7 Project Secure Supply Chain Management, SecureSCM (2009)
17. SecureSCM. Protocol Description V2. Deliverable D3.2, EU FP7 Project Secure Supply Chain Management, SecureSCM (2010)
18. Toft, T.: Primitives and Applications for Multi-party Computation. PhD dissertation, Univ. of Aarhus, Denmark, BRICS, Dep. of Computer Science (2007)
19. Toft, T.: Solving Linear Programs Using Multiparty Computation. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 90–107. Springer, Heidelberg (2009)

A Certifying Compiler for Zero-Knowledge Proofs of Knowledge Based on Σ-Protocols*

José Bacelar Almeida[1], Endre Bangerter[2], Manuel Barbosa[1], Stephan Krenn[3], Ahmad-Reza Sadeghi[4], and Thomas Schneider[4]

[1] Universidade do Minho, Portugal
{jba,mbb}@di.uminho.pt
[2] Bern University of Applied Sciences, Biel-Bienne, Switzerland
endre.bangerter@jdiv.org
[3] Bern University of Applied Sciences, Biel-Bienne, Switzerland, and University of Fribourg, Switzerland
stephan.krenn@bfh.ch
[4] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de

Abstract. Zero-knowledge proofs of knowledge (ZK-PoK) are important building blocks for numerous cryptographic applications. Although ZK-PoK have a high potential impact, their real world deployment is typically hindered by their significant complexity compared to other (non-interactive) crypto primitives. Moreover, their design and implementation are time-consuming and error-prone.

We contribute to overcoming these challenges as follows: We present a comprehensive specification language and a compiler for ZK-PoK protocols based on Σ-protocols. The compiler allows the fully automatic translation of an abstract description of a proof goal into an executable implementation. Moreover, the compiler overcomes various restrictions of previous approaches, e.g., it supports the important class of exponentiation homomorphisms with hidden-order co-domain, needed for privacy-preserving applications such as DAA. Finally, our compiler is certifying, in the sense that it automatically produces a formal proof of the soundness of the compiled protocol for a large class of protocols using the Isabelle/HOL theorem prover.

Keywords: Zero-Knowledge, Protocol Compiler, Formal Verification.

1 Introduction

A zero-knowledge proof of knowledge (ZK-PoK) is a two-party protocol between a prover and a verifier, which allows the prover to convince the verifier that he knows a secret value that satisfies a given relation (*proof of knowledge* or

* This work was in part funded by the European Community's Seventh Framework Programme (FP7) under grant agreement no. 216499. The compiler can be found at http://zkc.cace-project.eu, and a full version of this paper is given in [1].

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 151–167, 2010.

soundness), without the verifier being able to learn anything about the secret (*zero-knowledge*). For a formal definition we refer to [2].

Almost all practically efficient ZK-PoK are based on so-called Σ-protocols, and allow one to prove knowledge of a preimage under a homomorphism (e.g., a secret discrete logarithm). These preimage proofs can then be combined to considerably more complex protocols. In fact, many systems in applied cryptography use such proofs as building blocks. Examples include voting schemes [3], biometric authentication [4], group signatures [5], interactive verifiable computation [6], e-cash [7], and secure multiparty computation [8].

While many of these applications only exist at the specification level, directed applied research efforts have already brought innovative systems using ZK-PoKs to the real world. The probably most prominent example is *Direct Anonymous Attestation* (DAA) [9], a privacy enhancing mechanism for remote authentication of computing platforms. Another example is the *idemix* anonymous credential system [10] for user-centric identity management.

Up to now, design, implementation and verification of the formal security properties (i.e., zero-knowledge and soundness) as well as code security properties (e.g., security against side channel vulnerabilities, etc.) is done "by hand". Past experiences, for example in realizing DAA and idemix, have shown that these are time consuming and error prone tasks. This is certainly due to the fact that ZK-PoK are considerably more complex than other non-interactive cryptographic primitives such as encryption schemes.

In particular, the soundness property needs to be proved for each ZK-PoK protocol from scratch. The proofs often are not inherently complex, but still require an intricate knowledge of the techniques being used. This is a major hurdle for the real world adoption of ZK-PoK, since even experts in the field are not immune to protocol design errors. In fact, minor flaws in protocol designs can lead to serious security flaws, see for example [11] reporting a flaw in [12].

In this paper we describe a compiler and integrated tools that support and partially automate the design, implementation and formal verification of ZK-PoK based on Σ-protocols. Our goal is to overcome the aforementioned difficulties concerning ZK-PoK, and thus to bring ZK-PoK to practice by making them accessible to a broader group of crypto and security engineers.

Our Contributions. In a nutshell, we present a toolbox that takes an abstract description of the proof goal[1] of a ZK-PoK as input, and produces a C implementation of a provably sound protocol. More precisely, we extend previous directions with multiple functionalities of practical and theoretical relevance:

– We present a compiler for ZK-PoK based on Σ-protocols. The compiler (and its input language) support all relevant basic Σ-protocols and composition techniques found in the literature. The user can specify preimage proofs for arbitrary group homomorphisms, and combine them by logical `AND` and `OR` operators. Also, algebraic relations among the secrets can be proved.

[1] By proof goal, we refer to *what a prover wants to demonstrate in zero-knowledge*. For instance, the proof goal can be to prove knowledge of a discrete logarithm.

Examples of protocols that can be automatically generated by our compiler include [3,4,5,6,7,8,9,10,13,14,15,16].

- The compiler absorbs certain design-level decisions, for example by automatically choosing security parameters and intervals used in the protocols to assert the statistical ZK property for proofs in hidden order groups. It thus eliminates the potential of security vulnerabilities resulting from inconsistent parameter choices. Further, the compiler has capabilities to automatically rewrite the proof goal to reduce the complexity of the generated protocol.
- Last but not least, our compiler partially alleviates the implementor from the responsibility to establish a theoretical security guarantee for the protocol, by producing a formal proof of its soundness[2]. Technically, the compiler produces a *certificate* that the protocol generated by the compiler fulfills its specification. The validity of the certificate is then formally verified by the Isabelle/HOL formal theorem prover [17]. Our tool can therefore be seen as a *certifying* compiler. The formal verification component currently supports a subset of the protocols for which our compiler can generate code, but it already covers a considerable class of applications, including [7,13,14].

Related Work. Compiler based (semi-)automatic generation of cryptographic protocols has attracted considerable research interest recently, for instance in the field of multi-party computations [18,19,20].

A first prototype ZK-PoK compiler was developed in [21,22] and extended within the CACE project [23,24]. Yet, this compiler offers no optimization or verification functionalities and can only handle a subset of the proof goals supported by our compiler whose architecture was presented in [25].

Very recently, Meiklejohn et al. [26] presented another ZK-PoK compiler for specific applications such as e-cash. To maximize efficiency, their tool generates protocols which exploit precomputations, a feature which is not yet supported by our compiler. However, our compiler provides a substantially broader class of proofs goals such as Or-compositions, and homomorphisms such as RSA [27]. Further, formal verification is left as an "interesting area of study" in [26].

Symbolic models that are suitable for expressing and formally analyzing protocols that rely on ZK protocols as building blocks were presented in [28,29]. In [28] the first mechanized analysis framework for such protocols was proposed by extending the automatic tool ProVerif [30]. The work in [31] proposed an alternative solution to the same problem based on a type-based mechanism. Our work does not overlap with these contributions, and can be seen as complementary. The previous frameworks assume that the underlying ZK-PoK components are secure under adequate security models in order to prove the security of higher level protocols. We work at a lower level and focus on proving that specific ZK-PoK protocols generated by our compiler satisfy the standard computational

[2] The soundness property is arguably the most relevant security property for many practical applications of ZK-PoK, as it essentially establishes that it is infeasible to prove an invalid knowledge claim. However, our tool is currently being expanded to cover other relevant security properties, namely the zero-knowledge property.

security model for this primitive. Recent results in establishing the computational soundness of ZK-PoK-aware symbolic analysis can be found in [32]. Currently, we do not establish a connection between the security properties offered by the ZK-PoK protocols produced by our compiler and the level of security required to enable the application of computational soundness results.

We follow a recent alternative approach to obtaining computational security guarantees through formal methods: directly transposing provable security arguments to mechanized proof tools. This allows to deal directly with the intricacies of security proofs, but the potential for mechanization is yet to match that of symbolic analysis. In this aspect, our work shares some of its objectives with parallel work by Barthe et al. [33] describing the formalization of the theory of ZK-PoK in the Coq-based CertiCrypt tool [34]. This formalization includes proofs for the general theorems that establish the completeness, soundness and special honest-verifier ZK properties for Σ-protocols based on homomorphisms. Proving that a concrete protocol satisfies this set of properties can then be achieved by instantiating these general proofs with concrete homomorphisms. Although not completely automatic, this requires relatively small interaction with the user. In this work we provide further evidence that the construction of computational security proofs over mechanized proof tools can be fully automatic. The catch is that our verification component is highly specialized for (a specific class of) ZK-PoK and relies on in-depth knowledge on how the protocol was constructed.

Our work is also related to the formal security analysis of cryptographic protocol implementations. A tool for the analysis of cryptographic code written in C is proposed in [35]. In [36,37], approaches for extracting models from protocol implementations written in F#, and automatically verifying these models by compilation to symbolic models (resp. computational models) in ProVerif [38] (resp. CryptoVerif [39]), can be found. As above, the latter works target higher level protocols such as TLS that use cryptographic primitives as underlying components. Furthermore, the static cryptographic library that implements these primitives must be trusted by assumption. Our work can be seen as a first step towards a tool to automatically extend such a *trusted computing base* when ZK-PoK protocols for different goals are required.

Structure of this document. In §2 we recap the theoretical framework used by our compiler, which we then present in §3. Finally, the formal verification infrastructure is explained in §4.

2 Preliminaries

Before recapitulating the basics of Σ-protocols we introduce some notation. By $s \in_R \mathcal{S}$ we denote the uniform random choice of s from set $\mathcal{S}$. The order of a group $\mathcal{G}$ is denoted by $\mathrm{ord}(\mathcal{G})$, and the smallest prime dividing $a \in \mathbb{Z}$ by $\mathrm{minDiv}(a)$. We use the notation from [40] for specifying ZK-PoK. A term like

$$\mathsf{ZPK}\left[(\chi_1, \chi_2) : y_1 = \phi_1(\chi_1) \quad \wedge \quad y_2 = \phi_2(\chi_2) \quad \wedge \quad \chi_1 = a\chi_2\right]$$

means "*zero-knowledge proof of knowledge of values* χ_1, χ_2 *such that* $y_1 = \phi_1(\chi_1)$, $y_2 = \phi_2(\chi_2)$, *and* $\chi_1 = a\chi_2$" with homomorphisms ϕ_1, ϕ_2. Variables of which knowledge is proved are denoted by Greek letters, whereas publicly known quantities are denoted by Latin letters. Note that this notation only specifies a *proof goal*: it describes what has to be proved, but there may be various, differently efficient protocols to do so. We call a term like $y = \phi(\chi)$ in the proof goal an *atom*. A *predicate* is the composition of atoms and predicates using arbitrary many (potentially none) boolean operators And ($\wedge$) and Or ($\vee$).

2.1 Σ-Protocols as ZK-PoK Protocols

Most practical ZK-PoK are based on *Σ-protocols*. Given efficient algorithms $\mathsf{P}_1, \mathsf{P}_2, \mathsf{V}$, these have the following form: to prove knowledge of a secret χ satisfying a relation with some public y, the prover first sends a *commitment* $t := \mathsf{P}_1(\chi, y)$ to the verifier, who then draws a random *challenge* c from a predefined *challenge set* $\mathcal{C}$. Receiving c, the prover computes a *response* $s := \mathsf{P}_2(\chi, y, c)$. Now, if $\mathsf{V}(t, c, s, y) = \mathsf{true}$, the verifier accepts the proof, otherwise it rejects. Whenever the verifier accepts, we call (t, c, s) an *accepting transcript*.

Formally, for the protocol to be a proof of knowledge with knowledge error κ, the verifier must always accept for an honest prover. Furthermore, there must be an algorithm E' satisfying the following: whenever a (potentially malicious) prover can make the verifier accept with probability $\varepsilon > \kappa$, E' can extract χ from the prover in a number of steps proportional to $(\varepsilon - \kappa)^{-1}$ [2]. For Σ-protocols, this boils down to the existence of an efficient *knowledge extractor* E, which takes as inputs two accepting protocol transcripts $(t, c', s'), (t, c'', s'')$ with $c' \neq c''$, and y, and outputs a value χ' satisfying the relation [41,42].

A Σ-protocol satisfies the ZK property, if there is an efficient *simulator* S, taking c, y as inputs, and outputting tuples that are indistinguishable from real accepting protocol transcripts with challenge c [41,42].

2.2 Proving Atoms

We next summarize the basic techniques for proving atoms.

The Σ^ϕ-Protocol. The Σ^ϕ-protocol allows to efficiently prove knowledge of preimages under homomorphisms with a finite domain [13,43]. For instance, it can be used to prove knowledge of the content of ciphertexts under the Cramer/Shoup [44] or the RSA [27] encryption schemes. Also, it can be used for all homomorphisms mapping into a group over elliptic curves.

The protocol flow, as well as inputs and outputs of both parties, are shown in Fig. 1. Depending on the homomorphism ϕ, the knowledge error κ of the Σ^ϕ-protocol varies significantly. We thus recall a definition by Cramer [41].

A homomorphisms $\phi : \mathcal{G} \to \mathcal{H}$ is called *special*, if there is a $v \in \mathbb{Z} \setminus \{0\}$, such that for every $y \in \mathcal{H}$ a preimage u of y^v can efficiently be computed. The pair (u, v) is then called a *pseudo-preimage* of y. For instance, all homomorphisms with known-order codomain are special: if $\mathrm{ord}(\mathcal{H}) = q$, a pseudo-preimage of $y \in \mathcal{H}$ is given by $(0, q)$, as we always have $\phi(0) = 1 = y^q$.

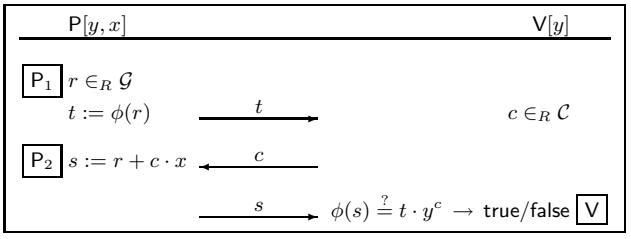


Fig. 1. The Σ^{ϕ}-protocol for performing $\mathsf{ZPK}[(\chi) : y = \phi(\chi)]$

We now state the knowledge error that can be achieved by the Σ^{ϕ}-protocol:

Theorem 1 ([41]). *Let ϕ be a homomorphism with finite domain. Then the Σ^{ϕ}-protocol using $\mathcal{C} = \{0, \ldots, c_{\max} - 1\}$ is a ZK-PoK with $\kappa = 1/c_{\max}$, if either $c_{\max} = 2$, or ϕ is special with special exponent v and $c_{\max} \leq \mathrm{minDiv}(v)$.*

The Σ^{GSP}- and the Σ^{exp}-Protocols. The practically important class of exponentiation homomorphisms with hidden-order codomain (including, for example, $\phi : \mathbb{Z} \to \mathbb{Z}_n^* : a \mapsto g^a$, where n is an RSA modulus, and g generates the quadratic residues modulo n) cannot be treated with the Σ^{ϕ}-protocol.

Two Σ-protocols for such homomorphisms can be found in the literature. The Σ^{GSP}-protocol generalizes the Σ^{ϕ}-protocol to the case of infinite domains (i.e., $\mathcal{G} = \mathbb{Z}$), and can be used very efficiently if assumptions on the homomorphism ϕ are made [45,46]. On the other hand, the so-called Σ^{exp}-protocol presented in [23,47] takes away these assumptions, by adding an auxiliary construction based on a common reference string and some computational overhead.

2.3 Operations on Σ-Protocols

Next, we briefly summarize some techniques, which allow one to use Σ-protocols in a more general way than for proving atoms only.

Reducing the knowledge error. The knowledge error of a Σ-protocol can be reduced from κ to κ^r by repeating the protocol r times in parallel. In this way, arbitrarily small knowledge errors can be achieved [2].

Composition techniques. In practice, it is often necessary to prove knowledge of multiple, or one out of a set of, secret values in one step. This can be achieved by performing so-called And- or Or-*compositions*, respectively. While the former requires the prover to know the secrets for *all* combined predicates to convince the verifier, he only needs to know *at least one* of them for the latter [48].

For a Boolean And, the only difference to running the proofs for the combined predicates independently in parallel is that the verifier only sends *one* challenge c, which is then used in all combined predicates.

Combining n predicates by a Boolean Or is slightly more involved. The prover is allowed to simulate all-but-one accepting protocol transcripts (for the predicates it does not know the secrets for) by allowing it to choose the according

c_i. The remaining challenge is then fixed such that $\sum_{i=1}^{n} c_i \equiv c \mod c_{\mathsf{max}}$. To ensure this, the response is now given by $((s_1, c_1), \ldots, (s_n, c_n))$, where s_i is the response of the i-th predicate. In addition to running all verification algorithms, the verifier also checks that the c_i add up to the challenge c.

In principle, any Boolean formula can be proved by combining these two techniques. Yet, when proving knowledge of k out of n secrets this becomes very inefficient if n is large. A much more efficient way for performing such *n-out-of-k threshold compositions* is to apply the technique from [49], instantiated with Shamir's secret sharing scheme [50].

Non-interactivity. By computing the challenge as hash of the commitment t with a random oracle, a Σ-protocol can be made non-interactive [51]. Besides reducing the round- and communication complexity of the proofs, this technique allows conversion into signature proofs of knowledge as well.

Algebraic relations among preimages. By re-adjusting the atoms of a proof goal, virtually any algebraic relations among the preimages can be proven. For examples we refer to [16,45,46,48,52,53].

3 Compiler

In this section we present a compiler, which is easy to use and that can generate code and documentation for the ZK-PoK protocols used in most systems found in the literature. Its modular design allows one to easily extend its functionality with minor effort only. Finally, an integrated tool is capable of formally verifying the soundness of the generated protocols for special homomorphisms.

3.1 Architecture

Our compiler is built from multiple components as shown in Fig. 2. These components are again designed modularly themselves. Thus, enhancing the compiler on a high level (e.g., adding backends) and on a low level (e.g., changing libraries used in the backends) can be done without affecting unrelated components.

Protocol Specification. We call the input language of our compiler *Protocol Specification Language* (or *PSL*). It is based on the standard notation for ZK-PoK introduced in [40], but takes away any ambiguity by also containing group specifications, etc. On a high level, PSL allows one to specify the protocol inputs, the algebraic setting, the types of Σ-protocols to be used, and how they should be composed (cf. §2.2 and §2.3). For a more detailed discussion of PSL see §3.2.

Protocol Compiler. The Protocol Compiler translates a PSL file into a protocol implementation formulated in our *Protocol Implementation Language* (*PIL*). This language can be thought of as a kind of pseudo-code describing the sequence of operations computed by both parties (including group operations, random choices, checks, etc.) and the messages exchanged.

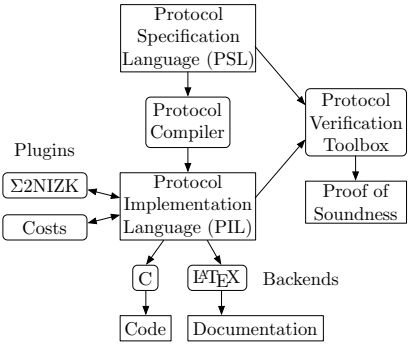


Fig. 2. Architecture of our ZK-PoK compiler suite

Backends. Backends allow to transform the protocol implementation into various output languages. The C Backend generates source code in the C programming language for prover and verifier which can be compiled and linked together with the GNU multi-precision arithmetic library [54] into executable code. The LaTeX Backend generates a human-readable documentation of the protocol.

Protocol Verification Toolbox. This component takes as inputs a PSL file together with the corresponding PIL file obtained from a compilation run. It first extracts the necessary information for constructing a formal proof of the soundness property of the generated protocol. The theorem prover Isabelle/HOL [55,56] is then used to automatically check the generated formal proof. We give more details on this toolbox in §4.

Plugins. Currently, two plugins are available in our compiler. The Σ2NIZK-plugin transforms the generated protocol in a non-interactive version thereof by applying the technique from [51], cf. §2.3. Its functionality could easily be extended to signature proofs of knowledge. The Costs plugin determines the abstract costs of the protocol by computing the communication complexity and the number of group operations needed in any of the involved groups. This enables a comparison of the efficiency of different protocols already on an abstract level.

3.2 Protocol Specification Language and Optimizations

Next, we illustrate usage of our compiler (and, in particular, PSL) using the following informal statement from a group-oriented application as example:

> *"One of two legitimate users has committed to message m without revealing m or the identity of the user who committed."*

For the example we assume that Pedersen commitments [14] were used to commit to m. Such commitments are of the form $c = g^m h^r$ for randomly chosen r. Here, $\langle g \rangle = \langle h \rangle = \mathcal{H}$ where $\mathcal{H}$ is a subgroup of prime order q of $\mathbb{Z}_p^*$ for some prime p. Further $\log_g h$ must not be known to the committing party.

```
Declarations { Prime(1024) p;
               Prime(160)  q;
               G=Zmod+(q)  m, r, sk_1, sk_2;
               H=Zmod*(p)  g@{order=q}, h@{order=q}, c@{order=q},
                           pk_1@{order=q}, pk_2@{order=q};  }
Inputs { Public        := p,q,g,h,c,pk_1,pk_2;
         ProverPrivate := m,r,sk_1,sk_2; }
Properties { KnowledgeError := 80;
             ProtocolComposition := P_0 And (P_1 Or P_2);  }
GlobalHomomorphisms { Homomorphism (phi : G -> H : (a) |-> (g^a)); }
// Predicates
SigmaPhi P_0 { Homomorphism (psi : G^2 -> H : (a,b) |-> (g^a * h^b));
               ChallengeLength := 80;  Relation ((c) = psi(m,r));     }
SigmaPhi P_1 { ChallengeLength := 80;  Relation ((pk_1) = phi(sk_1)); }
SigmaPhi P_2 { ChallengeLength := 80;  Relation ((pk_2) = phi(sk_2)); }
```

Fig. 3. PSL Example

To authenticate users we use Diffie-Hellman keys: each user randomly picks a sufficiently large secret key sk_i, computes the public key $\mathrm{pk}_i = g^{\mathrm{sk}_i}$ and publishes pk_i. For simplicity, we use the same group $\mathcal{H}$ for commitments and keys of users, but the compiler could use different groups as well.

Now, given $c, \mathrm{pk}_1, \mathrm{pk}_2$, the informal statement translates into this proof goal:

$$\mathsf{ZPK}\Big[(\mu,\rho,\sigma_1,\sigma_2) : c = g^{\mu}h^{\rho} \wedge \big(\mathrm{pk}_1 = g^{\sigma_1} \vee \mathrm{pk}_2 = g^{\sigma_2}\big)\Big].$$

With homomorphisms $\psi : (a,b) \mapsto g^a h^b$ and $\phi : (a) \mapsto g^a$ we rewrite this as

$$\mathsf{ZPK}\Big[(\mu,\rho,\sigma_1,\sigma_2) : \underbrace{c = \psi(\mu,\rho)}_{P_0} \wedge \big(\underbrace{\mathrm{pk}_1 = \phi(\sigma_1)}_{P_1} \vee \underbrace{\mathrm{pk}_2 = \phi(\sigma_2)}_{P_2}\big)\Big],$$

where the atoms are P_0, P_1, and P_2. This proof goal together with the underlying algebraic setting can be expressed in PSL as shown in Fig. 3 and described next. Each PSL file consists of the following blocks:

Declarations. All variables used in the protocol must first be declared here. PSL supports several data types with a given bit-length such as integers (`Int`) or primes (`Prime`). Also intervals (`[a,b]`) and predefined multiplicative and additive groups are supported, e.g., `Zmod*(p)` ($\mathbb{Z}_p^*, *$) and `Zmod+(q)` ($\mathbb{Z}_q, +$). Identifiers can be assigned to groups and constants can be predefined in this section. The compiler also supports abstract groups, which can later be instantiated with arbitrary groups (e.g., those over elliptic curves). The order of elements can be annotated for verification purposes, e.g., as `g@{order=q}`.

Inputs. Here, the inputs of both parties have to be specified. Only inputs that have been declared beforehand can be assigned.

Properties. This section specifies the properties of the protocol to be generated. For instance, `KnowledgeError := 80` specifies an intended knowledge error of $\kappa = 2^{-80}$. The proof goal can be specified by arbitrarily nesting atoms using Boolean And and Or operators. Furthermore, the compiler supports k-out-of-n-threshold compositions [49] based on Shamir secret sharing [50] (cf. §2.3).

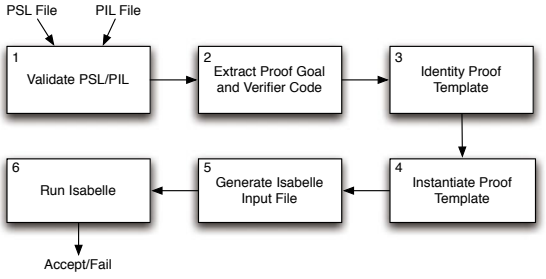


Fig. 4. Internal operation of the Protocol Verification Toolbox (PVT)

GlobalHomomorphisms. Homomorphisms that appear in multiple atoms can be defined globally in this optional section. Describing homomorphisms in PSL is a natural translation from their mathematical notation consisting of name, domain, co-domain, and the mapping function.

Predicates. Finally, the atoms used in `ProtocolComposition` are specified. Each predicate is proved with a Σ-protocol: one of `SigmaPhi`, `SigmaGSP` or `SigmaExp`. For each Σ-protocol, the relation between public and private values must be defined using local or global homomorphisms. `ChallengeLength` specifies the maximum challenge length that can be used to prove this atom with the given Σ-protocol (cf. §2.2 for details). Note that, in general, this value cannot be automatically determined by the compiler. It may depend, for example, on the size of the special exponent of the homomorphism, whose factorization might not be available. The compiler then automatically infers the required number of repetitions of each atom from this specification.

Optimizations. The compiler automatically transforms the proof goal in order to reduce its complexity. For instance, `P_1 Or P_2 Or (P_1 And P_2)` is simplified to `P_1 Or P_2`, which halves the complexity of the resulting protocol. By introspecting the predicates, further optimizations could be implemented easily.

4 Verification

The Protocol Verification Toolbox (PVT) of our compiler suite (cf. Fig. 2) automatically produces a formal proof for the soundness property of the compiled protocol. That is, it formally validates the guarantee obtained by a verifier executing the protocol: "The prover indeed knows a witness for the proof goal."

Overview. The internal operation of the PVT is sketched in Fig. 4; the phases (1) to (6) are explained in the following. As inputs, two files are given: the protocol specification (a PSL file) that was fed as input to the compiler, and the protocol implementation description that was produced by the compiler (a PIL file). The PVT first checks (1) the syntactic correctness of the files and their semantic consistency (e.g., it verifies that the PSL and PIL files operate on the

same groups, and other similar validations). Then, the information required for the construction of the soundness proof is extracted (2). This information essentially consists of the proof goal description from the PSL file and the code for the verifier in the implementation file. In particular, the former includes the definition of the concrete homomorphisms being used in the protocol, and information about the algebraic properties of group elements and homomorphisms[3].

The reason for the verification toolbox only considering the verifier code is that by definition [2] the soundness of the protocol essentially concerns providing guarantees for the verifier, regardless of whether the prover is honestly executing the protocol or not. Looking at the description of Σ-protocols in §2, one can see that the verifier code typically is very simple. The exception is the final algebraic verification that is performed on the last response from the prover, which determines whether the proof should be accepted. The theoretical soundness proof that we construct essentially establishes that this algebraic check is correct with respect to the proof goal, i.e., that it assures the verifier that the prover must know a valid witness. The soundness proof is then generated in three steps:

a) An adequate proof template is selected from those built into the tool (3). If no adequate template exists, the user is notified and the process terminates.
b) The proof template is instantiated with the concrete parameters corresponding to the input protocol (4) and translated into an output file (5) compatible with the Isabelle/HOL proof assistant: a theory file.
c) The proof assistant is executed on the theory file (6). If the proof assistant successfully finishes, then we have a formal proof of the theoretical soundness of the protocol. Isabelle/HOL also permits generating a human-readable version of the proof that can be used for product documentation.

The process is fully automatic and achieving this was a major challenge to our design. As can be seen in Fig. 4, our tool uses Isabelle/HOL [56] as a back-end (6). In order to achieve automatic validation of the generated proofs, it was necessary to construct a library of general lemmata and theorems in HOL that capture, not only the properties of the algebraic constructions that are used in ZK-PoK protocols, but also the generic provable security stepping stones required to establish the theoretical soundness property. We therefore employed and extended the Isabelle/HOL Algebra Library [57], which contains a wide range of formalizations of mathematical constructs. By relying on a set of existing libraries such as this, development time was greatly shortened, and we were able to create a proof environment in which we can express proof goals in a notation that is very close to the standard mathematical notation adopted in cryptography papers. More information about Isabelle/HOL can be found in [55,56].

Remark. No verification is carried out of the executable code generated from the PIL file. This is a program correctness problem rather than a theoretical security problem, and must be addressed using different techniques not covered here.

We next detail the most important aspects of our approach.

[3] This justifies the verification-specific annotations in the PSL file, as described in §3.

Proof strategy. Proving the soundness property of the ZK-PoK protocols produced by the compiler essentially means proving that the success probability of a malicious prover in cheating the verifier is bounded by the intended knowledge error. As described in §2.1, this involves proving the existence of (or simply to construct) an efficient knowledge extractor.

Our verification component is currently capable of dealing with the Σ^{ϕ}-protocol, which means handling proof goals involving special homomorphisms (cf. §2.2) for which it is possible to efficiently find pseudo-preimages. As all special homomorphisms used in cryptography fall into one of two easily recognizable classes, the verification toolbox is able to automatically find a pseudo-preimage for any concrete homomorphism that it encounters without human interaction.

A central stepping stone in formally proving the existence of an efficient knowledge extractor is the following lemma (which actually proves Theorem 1) that we have formalized in HOL.

Lemma 2 (Shamir's Trick [47]). *Let* (u_1, v_1) *and* (u_2, v_2) *be pseudo-preimages of* y *under homomorphism* ϕ*. If* v_1 *and* v_2 *are co-prime, then there exists a polynomial time algorithm that computes a preimage* x *of* y *under* ϕ*. This algorithm consists of using the Extended Euclidean Algorithm to obtain* $a, b \in \mathbb{Z}$ *such that* $av_1 + bv_2 = 1$*, and then calculating* $x = au_1 + bu_2$*.*

Given a special homomorphism and two accepting protocol transcripts for a ZK-PoK of an atom, we prove the existence of a knowledge extractor by ensuring that we are able instantiate Lemma 2.

The PVT also supports Boolean And and Or composition. If multiple predicates are combined by And, the verification tool defines as proof goal the existence of a knowledge extractor for each and all of them separately: one needs to show that the witness for each predicate can be extracted independently from the other predicates. In case of Or proofs (i.e., knowledge of one out of a set of preimages), the proof strategy looks as follows. First, for each atom, an Isabelle theorem proves the existence of a knowledge extractor. In a second step, it is then shown that the assumptions of at least one of these theorems are satisfied (i.e., that at least for one predicate we actually have different challenges).

Isabelle/HOL formalization. The HOL theory file produced by the Protocol Verification Toolbox is typical, in the sense that it contains a set of auxiliary lemmata that are subsequently used as simplification rules, and a final lemma with the goal to be proved. The purpose of the auxiliary lemmata is to decompose the final goal into simpler and easy to prove subgoals. They allow a systematic proof strategy that, because it is modularized, can handle proof goals of arbitrary complexity. Concretely, the proof goal for a simple preimage ZK-PoK such as those associated with Diffie-Hellman keys ($\mathrm{pk} = g^{\mathrm{sk}}$) used in the example in §3 looks like the following theorem formulation:

Theorem (Proof Goal). *Let* G *and* H *be commutative groups, where* G *represents the group of integers. Take as hypothesis the algebraic definition of the exponentiation homomorphism* $\phi : G \to H$*, quantified for all values of* G*: fix* $g \in H$ *with order* q *and assume* $\forall a \in G.\ \phi(a) = g^a$*. Furthermore, take a prime*

$q > 2$ and $c_{\mathsf{max}} \in \mathbb{Z}$ such that $0 < c_{\mathsf{max}} < q$, take $t, \mathrm{pk} \in H$ such that the order of pk *is q, take $s', s'' \in G$ and $c', c'' \in \mathbb{Z}$ such that $0 < c', c'' < c_{\mathsf{max}}$ and $c' \neq c''$, and assume $\phi(s') = t \cdot \mathrm{pk}^{c'} \ \wedge \ \phi(s'') = t \cdot \mathrm{pk}^{c''}$. Then there exist $a, b \in \mathbb{Z}$ such that $\phi(au + b\Delta s) = \mathrm{pk} \wedge av + b\Delta c = 1$, where $\Delta s := s' - s''$ and $\Delta c := c' - c''$, and $(u, v) = (0, q) \in G \times \mathbb{Z}$ is a pseudo-preimage of* pk *under ϕ.*

Instrumental in constructing the proof goal and auxiliary lemmata for the formal proof are the verifier's verification equations extracted from the PIL file. Indeed, the part of the proof goal that describes the two transcripts of the protocol (t, c', s') and (t, c'', s'') is constructed by translating this verification equation into Isabelle/HOL. For example, the following PIL-statement:

```
Verify((_t*(pk^_c)) == (g^_s));
```

will be translated into the Isabelle/HOL formalization

$$t \otimes_H (\mathrm{pk}(\wedge_H)c') = g(\wedge_H)s';\ t \otimes_H (\mathrm{pk}(\wedge_H)c'') = g(\wedge_H)s'';\ c' \neq c'';$$

where $\otimes_H$ and $(\wedge_H)$ represent the multiplicative and exponentiation operations in H, respectively. A typical proof is then structured as follows.

A first lemma with these equations as hypothesis allows the system to make a simple algebraic manipulation, (formally) proving that

$$(t \otimes_H (\mathrm{pk}(\wedge_H)c')) \otimes_H \mathrm{inv}_H(t \otimes_H (\mathrm{pk}(\wedge_H)c'')) = g(\wedge_H)s' \otimes_H \mathrm{inv}_H(g(\wedge_H)s'')$$

where inv_H represents the inversion operation for H. The subsequent lemmata continue simplifying this equation, until we obtain:

$$\mathrm{pk}(\wedge_H)(c' - c'') = g(\wedge_H)(s' - s'').$$

By introducing the homomorphism $\phi : G \to H$ we are able to show

$$\mathrm{pk}(\wedge_H)(\Delta c) = \phi(\Delta s)$$

where $\Delta c = c' - c''$ and $\Delta s = s' - s''$. We thus obtained $(\Delta s, \Delta c)$ as a first pseudo-preimage. The second one needed in Lemma 2 is found by analyzing the proof goal in the PSL file, which in our case was `Relation((pk) = phi(sk))`.

As we have embedded in our tool the domain specific knowledge to generate pseudo-preimages for the class of protocols that we formally verify, we can introduce another explicit pseudo-preimage as an hypothesis in our proof, e.g. $(0, q)$, and prove that it satisfies the pseudo-preimage definition. At this point we can instantiate the formalization of Lemma 2, and complete the proof for the above theorem, which implies the existence of a knowledge extractor.

Proof goals for more complex Σ-protocols involving And and Or compositions are formalized as described in the previous subsection and in line with the theoretic background introduced in §2. For And combinations, the proof goal simply contains the conjunction of the independent proof goals for each of the simple

preimage proofs provided as atoms. For Or combinations, the proof goal assumes the existence of two transcripts for the composed protocol

$$((t^1,\ldots,t^n),c',((s_1^1,c_1^1),\ldots,(s_1^n,c_1^n)) \quad \text{with} \quad \sum_{i=1}^{n} c_1^i \equiv c' \mod c_{\mathsf{max}}$$

and analogously for c'', such that $c' \neq c''$. It then states that for some $i \in \{1,\ldots,n\}$ we can construct a proof of existence of a knowledge extractor such as that described above. The assumptions regarding the consistency of the previous summations are, again, a direct consequence of the verifier code as stated in §2.3.

Acknowledgments. We gratefully acknowledge Wilko Henecka and Andreas Grünert for their support on the implementation of the compiler. Also, we would like to thank Stefania Barzan for implementing a preliminary version of the PVT.

References

1. Almeida, J., Bangerter, E., Barbosa, M., Krenn, S., Sadeghi, A.R., Schneider, T.: A certifying compiler for zero-knowledge proofs of knowledge based on Σ-protocols. Cryptology ePrint Archive, Report 2010/339 (2010)
2. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993)
3. Han, W., Chen, K., Zheng, D.: Receipt-freeness for Groth e-voting schemes. Journal of Information Science and Engineering 25, 517–530 (2009)
4. Kikuchi, H., Nagai, K., Ogata, W., Nishigaki, M.: Privacy-preserving similarity evaluation and application to remote biometrics authentication. Soft Computing 14, 529–536 (2010)
5. Camenisch, J.: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. PhD thesis, ETH Zurich, Konstanz (1998)
6. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999)
7. Brands, S.: Untraceable off-line cash in wallet with observers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994)
8. 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)
9. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: ACM CCS 2004, pp. 132–145. ACM Press, New York (2004)
10. Camenisch, J., Herreweghen, E.V.: Design and implementation of the idemix anonymous credential system. In: ACM CCS 2002, pp. 21–30. ACM Press, New York (2002)
11. Kunz-Jacques, S., Martinet, G., Poupard, G., Stern, J.: Cryptanalysis of an efficient proof of knowledge of discrete logarithm. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 27–43. Springer, Heidelberg (2006)
12. Bangerter, E., Camenisch, J., Maurer, U.: Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 154–171. Springer, Heidelberg (2005)

13. Schnorr, C.: Efficient signature generation by smart cards. Journal of Cryptology 4, 161–174 (1991)
14. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
15. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
16. Lipmaa, H.: On diophantine complexity and statistical zeroknowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)
17. Paulson, L.: Isabelle: a Generic Theorem Prover. Volume 828 of LNCS. Springer (1994)
18. MacKenzie, P., Oprea, A., Reiter, M.K.: Automatic generation of two-party computations. In: ACM CCS 2003, pp. 210–219. ACM, New York (2003)
19. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: USENIX Security 2004 (2004)
20. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) Public Key Cryptography – PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
21. Briner, T.: Compiler for zero-knowledge proof-of-knowledge protocols. Master's thesis, ETH Zurich (2004)
22. Camenisch, J., Rohe, M., Sadeghi, A.R.: Sokrates - a compiler framework for zero-knowledge protocols. In: WEWoRC 2005 (2005)
23. Bangerter, E., Camenisch, J., Krenn, S., Sadeghi, A.R., Schneider, T.: Automatic generation of sound zero-knowledge protocols. Cryptology ePrint Archive, Report 2008/471, Poster Session of EUROCRYPT 2009 (2008)
24. Bangerter, E., Briner, T., Heneka, W., Krenn, S., Sadeghi, A.R., Schneider, T.: Automatic generation of Σ-protocols. In: EuroPKI 2009 (to appear, 2009)
25. Bangerter, E., Krenn, S., Sadeghi, A.R., Schneider, T., Tsay, J.K.: On the design and implementation of efficient zero-knowledge proofs of knowledge. In: Software Performance Enhancements for Encryption and Decryption and Cryptographic Compilers – SPEED-CC 2009, October 12-13 (2009)
26. Meiklejohn, S., Erway, C., Küpçü, A., Hinkle, T., Lysyanskaya, A.: ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. In: USENIX 10 (to appear, 2010)
27. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)
28. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In: IEEE Symposium on Security and Privacy – SP 2008, pp. 202–215. IEEE, Los Alamitos (2008)
29. Baskar, A., Ramanujam, R., Suresh, S.P.: A dolev-yao model for zero knowledge. In: Datta, A. (ed.) ASIAN 2009. LNCS, vol. 5913, pp. 137–146. Springer, Heidelberg (2009)
30. Blanchet, B.: ProVerif: Cryptographic protocol verifier in the formal model (2010)
31. Backes, M., Hritcu, C., Maffei, M.: Type-checking zero-knowledge. In: ACM CCS 2008, pp. 357–370. ACM, New York (2008)

32. Backes, M., Unruh, D.: Computational soundness of symbolic zero-knowledge proofs against active attackers. In: IEEE Computer Security Foundations Symposium - CSF 2008, 255–269 Preprint on IACR ePrint 2008/152 (2008)
33. Barthe, G., Hedin, D., Zanella Béguelin, S., Grégoire, B., Heraud, S.: A machine-checked formalization of Σ-protocols. In: 23rd IEEE Computer Security Foundations Symposium, CSF 2010, IEEE, Los Alamitos (2010)
34. Barthe, G., Grégoire, B., Béguelin, S.: Formal certification of code-based cryptographic proofs. In: ACM SIGPLAN-SIGACT POPL 2009, pp. 90–101 (2009)
35. Goubault-Larrecq, J., Parrennes, F.: Cryptographic protocol analysis on real C code. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 363–379. Springer, Heidelberg (2005)
36. Bhargavan, K., Fournet, C., Gordon, A., Tse, S.: Verified interoperable implementations of security protocols. ACM Trans. Program. Lang. Syst. 31(1), 1–61 (2008)
37. Bhargavan, K., Fournet, C., Corin, R., Zalinescu, E.: Cryptographically verified implementations for TLS. In: ACM CCS 2008, pp. 459–468. ACM, New York (2008)
38. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: Workshop on Computer Security Foundations – CSFW 2001, p. 82. IEEE, Los Alamitos (2001)
39. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: IEEE Symposium on Security and Privacy – SP 2006, pp. 140–154. IEEE, Los Alamitos (2006)
40. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
41. Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, CWI and University of Amsterdam (1997)
42. Damgård, I.: On Σ-protocols, Lecture on Cryptologic Protocol Theory, Faculty of Science, University of Aarhus (2004)
43. Guillou, L., Quisquater, J.J.: A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
44. 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)
45. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
46. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 77–85. Springer, Heidelberg (2002)
47. Bangerter, E.: Efficient Zero-Knowledge Proofs of Knowledge for Homomorphisms. PhD thesis, Ruhr-University Bochum (2005)
48. Smart, N.P. (ed.): Final Report on Unified Theoretical Framework of Efficient Zero-Knowledge Proofs of Knowledge. CACE project deliverable (2009)
49. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
50. Shamir, A.: How to share a secret. Communications of the ACM 22, 612–613 (1979)
51. 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)

52. Brands, S.: Rapid demonstration of linear relations connected by boolean operators. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 318–333. Springer, Heidelberg (1997)
53. Bresson, E., Stern, J.: Proofs of knowledge for non-monotone discrete-log formulae and applications. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 272–288. Springer, Heidelberg (2002)
54. Granlund, T.: The GNU MP Bignum Library (2010), http://gmplib.org/
55. Nipkow, T., Paulson, L.: Isabelle (2010), http://isabelle.in.tun.de
56. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
57. Ballarin, C., Kammüller, F., Paulson, L.: The Isabelle/HOL Algebra Library (2008), http://isabelle.in.tum.de/library/HOL/HOL-Algebra/document.pdf

Short Generic Transformation to Strongly Unforgeable Signature in the Standard Model*

Joseph K. Liu[1], Man Ho Au[2], Willy Susilo[2], and Jianying Zhou[1]

[1] Cryptography and Security Department
Institute for Infocomm Research, Singapore
{ksliu,jyzhou}@i2r.a-star.edu.sg
[2] Centre for Computer and Information Security (CCISR)
School of Computer Science and Software Engineering
University of Wollongong, Australia
{aau,wsusilo}@uow.edu.au

Abstract. Standard signature schemes are usually devised to merely achieve existential unforgeability, i.e., to prevent forgeries on new messages not previously signed. Unfortunately, existential unforgeability is not suitable for several applications, since a new signature on a previously signed message may be produced. Therefore, there is a need to construct signature schemes with strong unforgeability, that is, it is hard to produce a new signature on any message, even if it has been signed before by legitimate signer. Recently, there have been several generic transformations proposed to convert weak unforgeability into strong unforgeability. For instance, various generic transforms of signatures that are existential unforgeable under adaptive chosen message attack (`uf-cma`) to strongly unforgeable under adaptive chosen message attack (`suf-cma`) have been proposed. Moreover, methods of converting signatures that are existentially unforgeable under generic chosen message attack (`uf-gma`) to `uf-cma` secure digital signatures have also been studied. Combination of these methods yields generic transform of digital signatures offering `uf-gma` security to `suf-cma` security. In this paper, we present a short universal transform that directly converts any `uf-gma` secure signatures into `suf-cma` secure. Our transform is *the shortest generic transformation*, in terms of signature size expansion, which results in `suf-cma` secure signature in the standard model. While our generic transformation can convert any `uf-gma` secure signature to `suf-cma` secure signature directly, the efficiency of ours is comparable to those which only transform signatures from `uf-gma` secure to `uf-cma` secure in the standard model.

1 Introduction

Digital signatures are amongst the most fundamental primitive of modern cryptography. The definition on the security of digital signatures was first formally

* The first and forth author of this work are funded by the A*STAR project SEDS-0721330047.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 168–181, 2010.

introduced by Goldwasser, Micali and Rivest [14]. Security of digital signatures can be classified according to the goals, as well as resources available to the adversary. As defined in [14], the goals of the adversary can be classified into four categories, listed below in descending order of difficulty.

- *Total Break*: The adversary is able to output the secret key of the signer.
- *Universal Forgery*: The adversary is able produce forgery on *any* message.
- *Selective Forgery*: The adversary is able to forge a signature on a message chosen before he learns the public key of the signer.
- *Existential Forgery*: The adversary is able to forge a signature on a new message[1] of its choice.

Security notions of digital signatures are also classified according to the resources available to the adversary. Two kinds of attacks are defined in [14]. In a *Key-Only Attack*, the adversary is only given the public key of the signer. *Message Attack* allows the enemy to examine some signatures corresponding to either known or chosen-messages prior to his attempt to break the scheme. *Message Attack* are further sub-divided into four categories, listed below in ascending order of adversary's capabilities.

- *Known Message*: The adversary is given access to signatures for a set of messages. These messages are known to the adversary but are *not* chosen by him.
- *Generic Chosen Message*: Also known as *Weak Chosen Message Attack* in [6] - the adversary is given access to signatures for a set of messages chosen by him. However, the adversary must choose the set of messages *before* knowing the public key of the signer.
- *Directed Chosen Message*: Similar to *Generic Chosen Message*, the adversary is provided with access to signatures for a set of messages chosen by him except he can now select the set *after* knowing the public key of the signer.
- *Adaptive Chosen Message*: This is the most general "natural" attack that any adversary can mount: the adversary is allowed to use the signer as an "oracle" in which he can request signatures from the signer of message on its choice[2].

Nowadays, standard signature schemes are usually designed to achieve existential unforgeability under adaptive chosen message attack (`uf-cma`). That is, forgery of signatures on new messages not previously signed are impossible even if the adversary is given signatures on any other messages of its choice. However, most signature schemes are randomized and allow many possible signatures for a single message. In this case, the notion `uf-cma` security of signature does not rule out

[1] By "new", we mean the corresponding signature of this message has not been provided to the adversary.

[2] The choice may depend on the signer's public key as well as previously obtained signatures.

the possibility of producing a new signature on a previously signed message. In other words, the adversary, given a message/signature pair (m, σ), maybe able to forge a new valid signature $\sigma' \neq \sigma$ on the same message m. In some applications, a stronger security notion, called *strong unforgeability* [1], is desirable.

Strong unforgeability ensures the adversary cannot even produce a new signature for a previously signed message. Consequently, any signature that passes the verification algorithm must be coming from the legitimate signer.

Strongly unforgeable signatures are useful in many different kinds of applications, such as building chosen-ciphertext secure encryption schemes [12,11], group signatures [2,7], signcryption [1] and authenticated key exchange [16]. Unfortunately, many signature schemes in the literature are *not* strongly unforgeable.

Sometimes existential unforgeability is also referred to as weak unforgeability (as in [21]) to emphasize its difference with strong unforgeability.

1.1 Generic Security-Amplifying Methods for Ordinary Digital Signatures

Several existing works [8,22,21,15,5,18] studied the problem of converting a signature scheme satisfying a weaker security notion into a stronger one with the help of some seemingly simpler cryptographic primitive, such as one-time signature or chameleon hash function. For instance, various generic transforms of signatures that are existential unforgeable under adaptive chosen message attack (`uf-cma`) to strongly unforgeable under adaptive chosen message attack (`suf-cma`) have been proposed [8,22,21,15,5]. Moreover, methods of converting signatures that are existentially unforgeable under generic chosen message attack (`uf-gma`) to `uf-cma` secure digital signatures have been studied in [13,17,20,6,23,18]. Combination of these methods yields generic transform of digital signatures offering `uf-gma` security to `suf-cma` security.

1.2 Our Result

We propose an efficient generic transformation for any signature scheme secure against existential forgery under generic chosen message attack (`uf-gma`) to a strongly unforgeable one under adaptive chosen message attack (`suf-cma`) in the standard model. Our transform is *universal* in the sense we treat the underlying signature as a blackbox and do not exploit any of its internal structure. Moreover, our transform only requires *one* chameleon hash function with a special property called *Given-Target One-Wayness* (GTOW). We observe that several existing chameleon hash functions exhibit this property. Therefore, we are not required to construct a specially designed chameleon hash function. Employing one of the most efficient chameleon hash functions, our transformation only increases the resulting signature by *one* group element. Hence, we refer to our transform as a *short universal transform*. Our transformation outperforms previous transformations in the following ways:

1. It is the shortest generic transformation, in terms of signature size expansion, which results in suf-cma secure signature in the standard model. Previous transformation either requires the random oracle model [22], or increases the signature size by at least 2 group elements [15], or only works for signature scheme with specific structure (e.g., partitioned signatures [8]).
2. While our generic transformation can convert any uf-gma secure signature to suf-cma secure signature directly, the efficiency of ours is comparable to those which only transform signatures from uf-gma secure to uf-cma secure in the standard model [13,17,20].

The features of existing transformations are summarized in Fig. 1. We compare the existing transformability (in the sense of uf-gma, uf-cma and suf-cma), signature size (if the transformation relies on another primitive such as chameleon hash function or one-time signature, we consider the most efficient one) and the security model.

It is easily observable that our transform is very efficient and only produces one group element in terms of signature size expansion.

1.3 Related Works

(uf-cma TO suf-cma): Recently, some techniques have been proposed to convert uf-cma secure signature schemes to suf-cma secure signatures. Boneh *et al.* [8] proposed the first conversion of such kind. However, their conversion is *not* universal. It can be only applied to some specific type of signatures, called *partitioned* signatures. Although many schemes fall in this class, some do not, such as DSA signature [19]. Teranishi *et al.* [22] proposed two universal conversions. Different from [8], they can be used to convert any kind of signatures. The first one requires the random oracle model and the second one works in the standard model. Both schemes rely on the hardness of the discrete-logarithm problem. The signature size increases by 1 element for the scheme secure in the random oracle model and by 2 elements for the scheme secure in the standard model. Steinfeld *et al.* [21] proposed another transformation[3] in the standard model which requires two chameleon hash functions. Similar to [22], the signature size also increases by two elements. All of the above transforms are based on chameleon hash functions or similar mechanisms. Huang *et al.* [15] proposed a generic transformation that makes use of strong one-time signature. The signature size increases by more than 2 to 3 elements, due to the public key of the one-time signature which is included in the final signature. Bellare *et al.* [5] employed a similar approach, by using a two-tier signature to produce the transformation. The resulting signature size also increases by at least 2 elements.

(uf-gma TO uf-cma): A signature which is uf-gma secure can be converted to uf-cma secure in the random oracle model, by using a hash function of the messages for signatures without changing any other algorithms. The random

[3] Their transformation is also applicable to uf-gma secure signature, as noted in the remark of [21].

	uf-gma	uf-cma	suf-cma	Increase in Signature Size (group element)	Standard Model	Remarks
Boneh *et al.* [8]		→		+1	✓	applicable to partitioned signatures only
Teranishi *et al.* [22] protocol 1		→		+1	×	RO only
Teranishi *et al.* [22] protocol 2		→		+2	✓	
Steinfeld *et al.* [21]	→	→		+2	✓	
Huang *et al.* [15]		→		+2	✓	
[13, 17, 20]	→			+1	✓	
Li *et al.* [18]	→			+2	✓	
Our Transformation	→	→		+1	✓	

Fig. 1. Comparison of different transformations

oracle allows the challenger to back patch the messages into different values that are fixed at the beginning [6,23].

Another approach is to use a chameleon hash function. A chameleon hash function is a kind of hash function with a trapdoor. By knowing the trapdoor information, one can find a pair of collision easily. Without this trapdoor, it is just similar to a normal hash function. The signer can first sign any value with the weak signature scheme. Then it can sign the real message from the signature on any value, by using the property of chameleon hash function with the knowledge of the trapdoor information. Both the public key and the signature size will be increased by at least one element. This technique has been used in [13,17,20].

Recently Li *et al.* [18] proposed another method to convert a `uf-gma` secure signature to a `uf-cma` secure one, by signing the message twice using the same weakly secure signature scheme. The signature size is increased by two elements.

1.4 Organization

The rest of the paper is organized as follows. We review some mathematical and security definition in Section 2. We introduce the new property *Given-Target One-Wayness* of chameleon hash function in Section 3 and give two examples to show that some existing classic chameleon hash functions possess this property. In Section 4 we present our generic transformation. Finally, we conclude the paper in Section 5.

2 Definition

2.1 Mathematical Assumption

Definition 1 (Discrete Log (DL) Assumption). *Let g be a generator in group G. Given an element $y \in G$ chosen at random, find an integer x such that $y = g^x$. An adversary $\mathcal{A}$ has at least an ϵ advantage if*

$$\Pr[\mathcal{A}(y, g, G) = x \mid y = g^x] \geq \epsilon$$

We say that the (ϵ, τ)-DL assumption holds if no algorithm running in time at most τ can solve that DL problem with advantage at least ϵ.

Definition 2 (One-More Discrete Log (OMDL) Assumption). *[4,13] Let g be a generator in group G. The adversary is given access to a discrete log solver oracle access $\mathcal{O}_{DL}$ in which when the adversary inputs an elements $y' \in G$ the oracle returns an element x' such that $y' = g^{x'}$, and a challenge oracle, which, returns a challenge point $y \in G$ each time it is invoked. Let $y_1, \ldots, y_n \in G$ be the challenges returned by the challenge oracle. The adversary is asked to output x_i such that $y_i = g^{x_i}$, for $i = 1, \ldots, n$ with the restriction that it can query $\mathcal{O}_{DL}$ for at most $n - 1$ times. An adversary $\mathcal{A}$ has at least an ϵ advantage if*

$$\Pr[\mathcal{A}^{\mathcal{O}_{DL}}(y_1, \ldots, y_n, g, G) = (x_1, \ldots, x_n) \mid y_1 = g^{x_1}, \ldots, y_n = g^{x_n}\ ] \geq \epsilon$$

We say that the (n, ϵ, τ)-OMDL assumption holds if no algorithm running in time at most τ can solve that OMDL problem with advantage at least ϵ.

2.2 Security Definition

Definition of Signature: We first review the definition of a digital signature scheme from [14] below.

Definition 3. *A signature scheme SIG is defined by three algorithms:*

- KeyGen is a probabilistic polynomial-time (PPT) algorithm that takes a security parameter k as input. It returns a public key pk and a secret key sk.
- Sign is a PPT algorithm taking as input (sk, m) where m is a message from a message space $\mathcal{S}$. It outputs a signature σ.
- Verify is a deterministic polynomial-time algorithm taking (pk, m, σ) as input. It returns either valid or invalid.

For correctness, we require that $\mathsf{valid} \leftarrow \mathsf{Verify}(pk, m, \sigma)$ where $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$.

Security of Signature: We review three levels of security for a signature, namely, *existential unforgeability against generic chosen message attack* (`uf-gma`) [14], *existential unforgeability against adaptive chosen message attack* (`uf-cma`) [14] and *the strong unforgeability against adaptive chosen message attack* [1]. We start with the most common one, `uf-cma`.

- `uf-cma`: Existential unforgeability against adaptive chosen message attack can be defined using the following game called **Game-UFCMA**:
  Setup: A public/secret key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$ is generated and adversary $\mathcal{A}$ is given the public key pk.

Query: $\mathcal{A}$ runs for time t and issues q signing queries to a signing oracle in an adaptive manner. For each i, $1 \leq i \leq q$, $\mathcal{A}$ chooses a message m_i based on the message-signature pairs that $\mathcal{A}$ has already seen, and obtains in return a signature σ_i on m_i from the signing oracle. That is, $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$.

Forge: $\mathcal{A}$ outputs a forgery (m^*, σ^*) and halts. $\mathcal{A}$ wins if

1. σ^* is a valid signature on message m^* under the public key pk. That is, $\mathsf{valid} \leftarrow \mathsf{Verify}(pk, m^*, \sigma^*)$; and
2. m^* has never been queried. That is, $m^* \notin \{m_1, \ldots, m_q\}$.

Definition 4 (uf-cma). *A signature scheme SIG=(KeyGen, Sign, Verify) is (t, q, ϵ)-existentially unforgeable against adaptive chosen message attacks, if any adversary with running time t wins the **Game-UFCMA** with probability at most ϵ after issuing at most q signing queries.*

– uf-gma: In a uf-gma game, the adversary has to submit all signature queries before seeing the public key. We refer to this game as **Game-UFGMA** which is defined as follows:

Query: The adversary $\mathcal{A}$ submits a list of q messages $m_1, \ldots, m_q$.

Setup: $\mathcal{A}$ is given a public key pk together with a list of message-signature pairs (m_i, σ_i) for $1 \leq i \leq q$.

Forge: Same as in **Game-UFCMA**.

Definition 5 (uf-gma). *A signature scheme SIG=(KeyGen, Sign,Verify) is (t, q, ϵ)-existentially unforgeable against generic chosen message attacks, if any adversary with running time t wins the **Game-UFGMA** with probability at most ϵ after issuing at most q signing queries.*

– suf-cma: One of the restrictions for adversary $\mathcal{A}$ in **Game-UFCMA** (resp. **Game-UFGMA**) is that the forging message m^* must be new. We can relax this restriction to obtain the notion of ***strong*** *existential unforgeable against adaptive chosen message attacks*, such that $\mathcal{A}$ forges a new valid signature on a message that could have been signed previously. We refer to this new game as **Game-SUFCMA** which is defined as follows:

Setup: Same as in **Game-UFCMA**.

Query: Same as in **Game-UFCMA**.

Forge: $\mathcal{A}$ outputs a forgery (m^*, σ^*) and halts. $\mathcal{A}$ wins if

1. σ^* is a valid signature on message m^* under the public key pk. That is, $\mathsf{valid} \leftarrow \mathsf{Verify}(pk, m^*, \sigma^*)$; and
2. $(m^*, \sigma^*) \notin \{(m_1, \sigma_1) \ldots, (m_q, \sigma_q)\}$.

Definition 6 (suf-cma). *A signature scheme SIG=(KeyGen, Sign,Verify) is (t, q, ϵ)-strongly existentially unforgeable against adaptive chosen message attacks, if any adversary with running time t wins the **Game-SUFCMA** with probability at most ϵ after issuing at most q signing queries.*

Chameleon Hash Function: In our generic transformation, we also require another primitive called the *chameleon hash function* [17,20]. We first describe the basic property of a chameleon hash function here, and introduce a new property in the next section. A chameleon hash function is a special type of hash function, whose collision resistance depends on the user's state of knowledge. Every chameleon hash function is associated with a pair of public key and private key, referred to as the hash key HK and the trapdoor key TK, respectively:

Definition 7 (Chameleon Hash Family). *A chameleon hash family consists of the following algorithms:*

- $\mathsf{HKey\text{-}Gen}$ is a PPT key generation algorithm that on input a security parameter k, outputs a pair (HK, TK).
- Hash_{HK} is a deterministic polynomial-time algorithm that on input a hash key HK and a pair $(M, R) \in \mathcal{M} \times \mathcal{R}$ where $\mathcal{M}$ is the message space and $\mathcal{R}$ is the space of the random element, output a hashed value C.
- $\mathsf{Collision\text{-}Finding}$ is a deterministic polynomial-time algorithm that on input a trapdoor and hash key pair (TK, HK), a pair $(M, R) \in \mathcal{M} \times \mathcal{R}$, and an additional message $M' \in \mathcal{M}$, outputs a value $R' \in \mathcal{R}$ such that $\mathsf{Hash}_{HK}(M, R) = \mathsf{Hash}_{HK}(M', R')$.

3 Given-Target One-Way Chameleon Hash

We introduce a new property called *given-target one-wayness* to chameleon hash functions. Informally speaking, it means that given a hash value, an attacker cannot find a pre-image without the trapdoor even if it is given another pre-image.

We examine some existing chameleon hash functions and show that at least two of the existing constructions satisfy this newly proposed property.

3.1 Definition

We define the property with the following game.

Definition 8 (Given-Target One-Wayness). *A chameleon hash family is possessing (t, q, ϵ) given-target one-wayness if no PPT adversary $\mathcal{A}$ with running time t, making at most q query can win in the following game with probability ϵ:*

Setup: *A hash key and trapdoor key pair $(HK, TK) \leftarrow \mathsf{HKey\text{-}Gen}(1^k)$ is generated and adversary $\mathcal{A}$ is given the hash key HK.*

Given-Target One-Way Query: *For the i-th query, $\mathcal{A}$ is given a hash value h_i. $\mathcal{A}$ submits a value M_i and is given R_i such that $\mathsf{Hash}_{HK}(M_i, R_i) = h_i$. Queries can be adaptive and interleaving.*

Output: *$\mathcal{A}$ outputs (M^*, R^*) and wins if $\mathsf{Hash}_{HK}(M^*, R^*) = h_i$ for some i and that $(M^*, R^*) \neq (M_i, R_i)$.*

We say a chameleon hash function is *given-target one-way* (GTOW) if it possesses given-target one-wayness.

3.2 Scheme under DL Assumption

We review the double trapdoor chameleon hash from [10] and show that it is GTOW under the discrete logarithm assumption.

The Scheme Scheme_{DL}: HKey-Gen. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. Randomly generate $x, y \in_R \mathbb{Z}_p$ and compute $u = g^x$, $v = g^y$. The message space $\mathcal{M}$ is $\mathbb{Z}_p$. The trapdoor key TK is (x, y) and the hash key HK is $(\mathbb{G}, p, g, u, v, \mathcal{M})$.

Hash_{HK}. For a message $M \in \mathcal{M}$, choose a random $\tilde{R} \in_R \mathbb{Z}_p$, $R \in_R \mathbb{Z}_p$ and compute

$$C = u^M v^{\tilde{R}} g^R$$

Collision-Finding. For a given pair $(M, \tilde{R}, R)$ and a message $M' \in \mathcal{M}$, in order to find $(\tilde{R}', R')$ such that $\mathsf{Hash}_{HK}(M, \tilde{R}, R) = \mathsf{Hash}_{HK}(M', \tilde{R}', R')$, first randomly generates $\tilde{R}' \in_R \mathbb{Z}_p$, then use the trapdoor key TK to compute

$$R' = (M - M')x + (\tilde{R} - \tilde{R}')y + R \bmod p$$

Lemma 1 (Scheme_{DL} is GTOW under DL assumption). *Assume there is an adversary $\mathcal{A}$ who can break the (τ, q, ϵ) given-target one-wayness of Scheme_{DL}, we can construct a simulator $\mathcal{S}$ to solve the (ϵ', τ')-DL problem, where*

$$\epsilon' \geq \frac{1}{2}\epsilon \qquad \tau = \tau'$$

Proof. Suppose there exists an adversary $\mathcal{A}$ in the sense of Definition 8. We construct a simulator $\mathcal{S}$, having black-box access to $\mathcal{A}$, that solves the DL problem.

Setup. $\mathcal{S}$ is given $\mathbb{G} = \langle w \rangle$ such that $|w| = p$ for some prime p and a value Y, and its goal is to output $y \in \mathbb{Z}_p$ such that $Y = w^y$. $\mathcal{S}$ randomly generates $x \in_R \mathbb{Z}_p$ and flips a fair coin $b \in_R \{0, 1\}$. If $b = 0$, computes $u = w^x$ and sets $v = Y$ and $g = w$. Otherwise, computes $u = w^x$ and sets $v = w$ and $g = Y$. $\mathcal{S}$ gives $(\mathbb{G}, p, g, u, v, \mathcal{M} := \mathbb{Z}_p)$ to $\mathcal{A}$ as HK of the trapdoor hash.

Query. For the i-th query, $\mathcal{S}$ randomly generates $m_i', \tilde{r}_i', r_i' \in_R \mathbb{Z}_p$, computes $h_i = u^{m_i'} v^{\tilde{r}_i'} g^{r_i'}$ and returns h_i to $\mathcal{A}$. Upon receiving the corresponding m_i from $\mathcal{A}$, $\mathcal{S}$ computes $r_i = (m_i' - m_i)x + r_i'$ and sets $\tilde{r}_i = \tilde{r}_i'$ if $b = 0$. Otherwise, $\mathcal{S}$ computes $\tilde{r}_i = (m_i' - m_i)x + \tilde{r}_i'$ and sets $r_i = r_i'$. Then, $\mathcal{S}$ returns $(\tilde{r}_i, r_i)$ to $\mathcal{A}$.

Output. Finally, $\mathcal{A}$ outputs $m^*, \tilde{r}^*, r^*$ such that $u^{m^*} v^{\tilde{r}^*} g^{r^*} = u^{m_i} v^{\tilde{r}_i} g^{r_i}$ for some i. If $b = 0$, $\mathcal{S}$ aborts if $\tilde{r}^* = \tilde{r}_i$. Otherwise $\mathcal{S}$ computes

$$y = \frac{(m_i - m^*)x + r_i - r^*}{\tilde{r}^* - \tilde{r}_i}$$

If $b = 1$, $\mathcal{S}$ aborts if $r^* = r_i$. Otherwise, $\mathcal{S}$ computes

$$y = \frac{(m_i - m^*)x + \tilde{r}_i - \tilde{r}^*}{r^* - r_i}$$

$\mathcal{S}$ returns y as the solution of the discrete logarithm problem.

Successful Simulation. Since $(m^*, \tilde{r}^*, r^*) \neq (m_i, \tilde{r}_i, r_i)$, at least one of the statements $\tilde{r}^* \neq \tilde{r}_i$ or $r^* \neq r_i$ is true. For each case, with probability $1/2$, $\mathcal{S}$ does not abort. The running time of $\mathcal{S}$ is just the same as $\mathcal{A}$. □

3.3 Scheme under One-More-DL Assumption

Next, we review the classic DL-based chameleon hash function [9,17] and show that it is GTOW under the one-more discrete logarithm assumption.

The Scheme Scheme_{OMDL}: $\mathsf{HKey\text{-}Gen}$. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. Randomly generate $x \in_R \mathbb{Z}_p$ and compute $u = g^x$. The message space $\mathcal{M}$ is $\mathbb{Z}_p$. The trapdoor key TK is (x) and the hash key HK is $(\mathbb{G}, p, g, u, \mathcal{M})$.

Hash_{HK}. For a message $M \in \mathcal{M}$, choose a random $R \in_R \mathbb{Z}_p$ and compute

$$C = u^M g^R$$

$\mathsf{Collision\text{-}Finding}$. For a given pair (M, R) and a message $M' \in \mathcal{M}$, in order to find (R') such that $\mathsf{Hash}_{HK}(M, R) = \mathsf{Hash}_{HK}(M', R')$, use the trapdoor key TK to compute

$$R' = (M - M')x + R \bmod p$$

Lemma 2 (Scheme_{OMDL} is GTOW under OMDL assumption). *Assume there is an adversary $\mathcal{A}$ who can break the (τ, q, ϵ) given-target one-wayness of Scheme_{OMDL}, we can construct a simulator $\mathcal{S}$ to solve the $(q-1, \epsilon', \tau')$-OMDL problem, where*

$$\epsilon' = \epsilon \qquad \tau = \tau'$$

Proof. Suppose there exists an adversary $\mathcal{A}$ in the sense of Definition 8. We construct a simulator $\mathcal{S}$, having black-box access to $\mathcal{A}$, that solves the one-more discrete logarithm problem.

Setup. $\mathcal{S}$ is given $\mathbb{G} = \langle g \rangle$ such that $|g| = p$ for some prime p access to two oracles, namely, $\mathcal{O}_Q$ and $\mathcal{O}_{DL}$. When invoked for the i-th time, $\mathcal{O}_Q$ returns an element $Y_i \in \mathbb{G}$ while $\mathcal{O}_{DL}(X_i)$ returns $x_i \in \mathbb{Z}_p$ such that $X_i = g^{x_i}$. The goal of $\mathcal{S}$ is to solves all the discrete logarithm of Y_i to base g while using less number of query to $\mathcal{O}_{DL}$. $\mathcal{S}$ invokes $\mathcal{O}_Q$, obtains Y_0 and sets $u = Y_0$. $\mathcal{S}$ gives $(\mathbb{G}, p, g, u, \mathcal{M} := \mathbb{Z}_p)$ to $\mathcal{A}$ as HK of the trapdoor hash.

Query. For the i-th query, $\mathcal{S}$ invokes $\mathcal{O}_Q$ to receive Y_i. $\mathcal{S}$ then sets $h_i = Y_i$ and returns h_i to $\mathcal{A}$. Upon receiving the corresponding m_i from $\mathcal{A}$, $\mathcal{S}$ invokes $\mathcal{O}_{DL}(\frac{h_i}{u^{m_i}})$ and obtains r_i such that $h_i = u^{m_i} g^{r_i}$. $\mathcal{S}$ returns r_i to $\mathcal{A}$.

Output. Finally, $\mathcal{A}$ outputs m^*, r^* such that $u^{m^*} g^{r^*} = u^{m_i} g^{r_i}$ for some i. $\mathcal{S}$ computes $y_0 = \frac{(r_i - r^*)}{m^* - m_i}$. Next, $\mathcal{S}$ computes $y_i = m_i y_0 + r_i$. $\mathcal{S}$ returns y_0 as the discrete logarithm of Y_0 and y_i's as the discrete logarithms of Y_i.

Successful Simulation. Clearly the successful probability and running time of $\mathcal{S}$ are the same as $\mathcal{A}$. □

4 Generic Transformation to SUF-CMA Secure Signature

4.1 Our Transformation

We build a generic and universal transformation which converts any `uf-gma` secure signature scheme $\text{SIG}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ (with message space $\mathcal{S}'$) to a `suf-cma` secure signature scheme SIG=(KeyGen, Sign, Verify). This transformation requires a give-target one-way chameleon hash function $\mathsf{Hash}_{HK} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{S}'$ (we use the notation described in Section 2.2) and a regular collision-resistant hash function $H : \{0,1\}^* \rightarrow \mathcal{M}$. The transformation is as follows:

- KeyGen: Generate a public / secret key pair $(pk', sk') \leftarrow \mathsf{KeyGen}'(1^k)$. Generate a hash key and trapdoor key $(HK, TK) \leftarrow \mathsf{HKey\text{-}Gen}(1^k)$. Set the public key as $pk = (pk', HK, \mathsf{Hash}_{HK}, H)$ and secret key as $sk = (sk', TK)$.
- Sign: On input a secret key sk and a message m, the following steps are carried out and a signature σ is generated:
  1. Randomly generate $m' \in_R \mathcal{M}$ and $r' \in_R \mathcal{R}$.
  2. Compute $h' \leftarrow \mathsf{Hash}_{HK}(m'; r')$.
  3. Compute $s' \leftarrow \mathsf{Sign}'(sk', h')$.
  4. Compute $h \leftarrow H(m||s')$.
  5. Using the trapdoor key TK to find r such that $\mathsf{Hash}_{HK}(h; r) = \mathsf{Hash}_{HK}(m'; r')$.
  6. Output the signature $\sigma \leftarrow (s', r)$.
- Verify: On input public key pk, message m and signature σ, output

$$\mathsf{Verify}'\Big(pk', \mathsf{Hash}_{HK}\Big(H(m||s'); r\Big), s'\Big)$$

4.2 Security Analysis

Theorem 1 (Strong Unforgeability). *Assume there is an adversary $\mathcal{A}$ who can break the (τ, q, ϵ) `suf-cma` security of the signature scheme, we can construct a simulator $\mathcal{S}$ to either break the $(\tau, q, \epsilon/2)$ `uf-gma` security of the underlying signature scheme, or the $(\tau, q, \epsilon/2)$ given-target one-wayness of the chameleon hash function Hash_{HK}.*

Proof. Suppose there exists an adversary $\mathcal{A}$, making q signing queries, wins Game-SUFCMA with non-negligible probability with SIG=(KeyGen, Sign, Verify), we show that either $\text{SIG}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ is not `uf-gma` secure or Hash_{HK} is not given-target one-way.

The proof is by reduction. Firstly, we classify the adversary $\mathcal{A}$ into two types. Suppose the q message-signature pairs $\mathcal{A}$ obtained in the Query phase are $\big(m_i,$

$(s'_i, r_i)\big)$ for $i = 1$ to q and the final forged message-signature pair is $\big(m^*, (s'^*, r^*)\big)$. We say $\mathcal{A}$ is of type I if $\mathsf{Hash}_{HK}\big(H(m^*||s'^*); r^*\big) = \mathsf{Hash}_{HK}\big(H(m_i||s'_i); r_i\big)$ for some $i = 1$ to q. Otherwise, $\mathcal{A}$ is of type II.

Next, we construct a simulator $\mathcal{S}$ which simulates GAME-SUFCMA for type I and II adversary differently, in such a way that the adversary cannot distinguish which kind of simulation it is in.

1. Type I Adversary (Breaking Given-Target One-wayness Property of Hash_{HK})
   (Setup.) $\mathcal{S}$ is given a chameleon hash function $\mathsf{Hash}_{HK} : \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{S}'$ without the trapdoor and its goal is to break given-target one-wayness as defined in Definition 8. It invokes KeyGen' and obtains (pk', sk'). Let H be a collision-resistant hash function. $\mathcal{S}$ gives $\mathcal{A}$ $(pk', HK, \mathsf{Hash}_{HK}, H)$ as the public key of the signature scheme SIG.
   (Query.) For a signature query m, $\mathcal{S}$ issues a Given-Target One-Way Query and obtains h_i. It sets $h' = h_i$ and computes $s' = \mathsf{SIG}'(sk', h')$ using the corresponding secret key sk'. It computes $h = H(m||s')$ and sets $m_i = h$. It sends m_i to the Given-Target One-Way Oracle and receives r_i such that $h_i = \mathsf{Hash}_{HK}(m_i, r_i)$. It sets $r = r_i$ and returns (s', r) as the corresponding signature.
   (Forgery.) Finally, $\mathcal{A}$ outputs a message m^* and a corresponding signature (s'^*, r^*) such that $\mathsf{Verify}'(pk', \mathsf{Hash}_{HK}(H(m^*||s'^*; r))), s'^*)$ returns valid.
   (Reduction.) Since $\mathcal{A}$ is of type I, there exists a query message m_i with resulting signature (s'_i, r_i) such that

$$\mathsf{Hash}_{HK}\big(H(m^*||s'^*); r^*\big) = \mathsf{Hash}_{HK}\big(H(m_i||s'_i); r_i\big).$$

   $\mathcal{S}$ returns $(H(m^*||s'^*), r^*)$ as the output that breaks the given-target one-wayness of Hash_{HK}.
2. Type II Adversary (Breaking $\mathtt{uf\text{-}gma}$ Security of SIG′)
   (Setup.) $\mathcal{S}$ invokes $\mathsf{HKey\text{-}Gen}(1^k)$ and obtains (HK, TK) for the chameleon hash function Hash_{HK}. $\mathcal{S}$ randomly generates m'_i, r'_i and computes $h'_i = \mathsf{Hash}_{HK}(m'_i; r'_i)$ for $i = 1$ to q where q is the total number of signature query. $\mathcal{S}$ submits $h'_1, \cdots, h'_q$ to the challenger as the chosen messages and obtains a signature scheme SIG′ with public key pk' and signatures of s'_i for $i = 1$ to q as the signature of h'_i under public key pk'. $\mathcal{S}$ gives $\mathcal{A}$ $(pk', HK, \mathsf{Hash}_{HK}, H)$ as the public key of the signature scheme SIG.
   (Query.) For the i-th query, $\mathcal{S}$ is given a message m_i. It computes $h_i = H(m_i||s'_i)$ and find r_i such that $\mathsf{Hash}_{HK}(h_i; r_i) = h'_i$ using the trapdoor TK. Return (s'_i, r_i) to $\mathcal{A}$ as the corresponding signature.
   (Forgery.) Finally, $\mathcal{A}$ outputs a message m^* and a corresponding signature (s'^*, r^*) such that $\mathsf{Verify}'(pk', \mathsf{Hash}_{HK}(H(m^*||s'^*; r))), s'^*)$ returns valid.
   (Reduction.) $\mathcal{S}$ computes $M^* = \mathsf{Hash}_{HK}(H(m^*||s'^*; r)))$ and returns (M^*, s'^*) as the forged message-signature pair to the underlying signature scheme SIG′.

For the probability analysis, it is easy to see that for each type it can be perfectly simulated, so that each type occurs with the same probability as in the real attack. Hence either the attacker of type I or type II succeeds with probability $\epsilon/2$, as claimed. The running time should be the same. □

5 Conclusion

We presented an efficient universal transform that directly converts any `uf-gma` secure signatures into `suf-cma` secure. Our transform is the shortest generic transformation, in terms of signature size expansion. While our generic transformation can convert any `uf-gma` secure signature to `suf-cma` secure signature directly, the efficiency of ours is comparable to those which only transform signatures from `uf-gma` secure to `uf-cma` secure in the standard model. We make use of a new property in chameleon hash functions called *given-target one-wayness*. We demonstrated that this property exists in at least two of the existing constructions of chameleon hash functions in the literature, and henceforth, we do not need to create a specially designed chameleon hash function to be incorporated in our transform.

References

1. An, J., Dodis, Y., Rabin, T.: On the security of joint signatures and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002)
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000)
3. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. J. Cryptology 16(3), 185–215 (2003)
4. Bellare, M., Palacio, A.: Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer, Heidelberg (2002)
5. Bellare, M., Shoup, S.: Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 201–216. Springer, Heidelberg (2007)
6. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
7. Boneh, D., Boyen, X., Shacham, H.: Short group signatures using strong Diffie-Hellman. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
8. Boneh, D., Shen, E., Waters, B.: Strongly unforgeable signatures based on computational Diffie-Hellman. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 229–240. Springer, Heidelberg (2006)
9. Boyar, J., Kurtz, S.A., Krentel, M.W.: A discrete logarithm implementation of perfect zero-knowledge blobs. J. Cryptology 2(2), 63–76 (1990)

10. Bresson, E., Catalano, D., Gennaro, R.: Improved on-line/off-line threshold signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 217–232. Springer, Heidelberg (2007)
11. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
12. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. SIAM J. Computing 30(2), 391–437 (2000)
13. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital signatures. J. Cryptology 9(1), 35–67 (1996)
14. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Computing 17(2), 281–308 (1998)
15. Huang, Q., Wong, D.S., Zhao, Y.: Generic transformation to strongly unforgeable signatures. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 1–17. Springer, Heidelberg (2007)
16. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
17. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS, The Internet Society (2000)
18. Li, J., Kim, K., Zhang, F., Wong, D.S.: Generic security-amplifying methods of ordinary digital signatures. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 224–241. Springer, Heidelberg (2008)
19. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2 (January 2000)
20. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 355–367. Springer, Heidelberg (2001)
21. Steinfeld, R., Pieprzyk, J., Wang, H.: How to strengthen any weakly unforgeable signature into a strong unforgeable signature. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 357–371. Springer, Heidelberg (2006)
22. Teranishi, I., Oyama, T., Ogata, W.: General conversion for obtaining strongly existentially unforgeable signatures. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 191–205. Springer, Heidelberg (2006)
23. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 277–290. Springer, Heidelberg (2004)

DR@FT: Efficient Remote Attestation Framework for Dynamic Systems*

Wenjuan Xu[1], Gail-Joon Ahn[2], Hongxin Hu[2],
Xinwen Zhang[3], and Jean-Pierre Seifert[4]

[1] Frostburg State University
wxu@frostburg.edu
[2] Arizona State University
gahn@asu.edu ,hxhu@asu.edu
[3] Samsung Information Systems America
xinwen.z@samsung.com
[4] Deutsche Telekom Lab and Technical University of Berlin
jean−pierre.seifert@telekom.de

Abstract. Remote attestation is an important mechanism to provide the trustworthiness proof of a computing system by verifying its integrity. In this paper, we propose an innovative remote attestation framework called DR@FT for efficiently measuring a target system based on an information flow-based integrity model. With this model, the high integrity processes of a system are first verified through measurements and these processes are then protected from accesses initiated by low integrity processes. Also, our framework verifies the latest state changes in a dynamic system instead of considering the entire system information. In addition, we adopt a graph-based method to represent integrity violations with a ranked violation graph, which supports intuitive reasoning of attestation results. Our experiments and performance evaluation demonstrate the feasibility and practicality of DR@FT.

1 Introduction

In distributed computing environments, it is crucial to measure whether remote parties run buggy, malicious application codes or are improperly configured by rogue software. Remote attestation techniques have been proposed for this purpose through analyzing the integrity of remote systems to determine their trustworthiness. Typical attestation mechanisms are designed based on the following steps. First, an attestation requester (*attester*) sends a challenge to a target system (*attestee*), which responds with the evidence of integrity of its hardware and software components. Second, the attester derives runtime properties of the attestee and determines the trustworthiness of the attestee. Finally and optionally, the attester returns the attestation result, such as integrity measurement status, to the attestee. Remote attestation can help reduce potential risks that are caused by a tampered system.

* The work of Gail-J.Ahn and Hongxin Hu was partially supported by National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360).

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 182–198, 2010.

Various attestation approaches and techniques have been proposed. Trusted Computing Group (TCG) [2] specifies trusted platform module (TPM) which can securely store and provide integrity measurements of systems to a remote party. Integrity measurement mechanisms have been proposed to facilitate the capabilities of TPM at application level. For instance, Integrity Measurement Architecture (IMA) [12] is an implementation of TCG approach to provide verifiable evidence with respect to the current run-time state of a measured system. Several attestation methods have been proposed to accommodate privacy properties [7], system behaviors [8], and information flow model [9]. However, these existing approaches still need to cope with the efficiency when attesting a platform where its *system state* frequently changes due to system-centric events such as security policy updates and software package installations. Last but not least, existing attestation mechanisms do not have an effective and intuitive way for presenting attestation results and reflecting such results while resolving identified security violations.

Towards a systematic attestation solution, we propose an efficient remote attestation framework, called Dynamic Remote Attestation Framework and Tactics (DR@FT) to address aforementioned issues. Our framework is based on system integrity property with a *domain-based isolation* model. With this property, the high integrity processes of a system are first verified through measurements and these processes are then protected from accesses initiated by low integrity processes. In other words, the high integrity process protection is verified by analyzing the system's security policy, which specifies system configurations with system behaviors including application behaviors. Having this principle in place, DR@FT enables us verify whether certain applications in the attestee satisfy integrity requirements as part of attestation. Towards attesting a dynamic nature of the systems, DR@FT verifies the latest changes in a system state, instead of considering the entire system information for each attestation inquiry. Through these two tactics, our framework attempts to efficiently attest the target system. Also, DR@FT adopts a graph-based analysis methodology for analyzing security policy violations, which helps cognitively identify suspicious information flows in the attestee. To further improve the efficiency of security violation resolution, we propose a ranking scheme for prioritizing the policy violations, which provides a method for describing the *trustworthiness* of different system states with *risk levels*.

This paper is organized as follows. Section 2 overviews existing attestation work and system integrity models. Section 3 describes a domain-based isolation model which gives the theoretical foundation of DR@FT. Section 4 presents the design requirements and attestation procedures of DR@FT, followed by policy analysis methods and their usages in Section 5. We elaborate the implementation details and evaluation results in Section 6. Section 7 concludes this paper and examines some limitations of our work.

2 Background

2.1 Attestation

The TCG specification [2] defines mechanisms for a TPM-enabled platform to report its current hardware and software configuration status to a remote challenger. A TCG attestation process is composed of two steps: (i) an attestee platform measures hardware and software components starting from BIOS block and generates a hash value.

The hash value is then stored into a TPM Platform Configuration Register (PCR). Recursively, it measures BIOS loader and operating system (OS) in the same way and stores them into TPM PCRs; (ii) an attester obtains the attestee's digital certificate with an attestation identity key (AIK), AIK-signed PCR values, and a measurement log file from the attestee which is used to reconstruct the attestee platform configuration, and verifies whether this configuration is acceptable. From these steps we notice that TCG measurement process is composed of a set of sequential steps up to the bootstrap loader. Thus, TCG does not provide effective mechanisms for measuring a system's integrity beyond the system boot, especially considering the randomness of executable contents loaded by an running OS.

IBM IMA [12] extends TCG's measurement scope to application level. A measurement list *M* is stored in OS kernel and composed of m_0 ... m_i corresponding to loaded executable application codes. For each loaded m_i, an aggregated hash H_i is generated and loaded into TPM PCR, where H_0=H(m_0), and H_i=H(H_{i-1} || H(m_i)). Upon receiving the measurements and TPM-signed hash value, the attester proves the authentication of measurements by verifying the hash value, which helps determine the integrity level of the platform. However, IMA requires to verify the entire components of the attestee platform while the attestee may only demand the verification of certain applications. Also, the integrity status of a system is validated by testing each measurement entry independently, focusing on the high integrity processes. However, it is impractical to disregard newly installed untrusted applications or data from the untrusted network.

PRIMA [9] is an attestation work based on IMA and CW-Lite integrity model [14]. PRIMA attempts to improve the efficiency of attestation by only verifying codes, data, and information flows related to trusted subjects. On one hand, PRIMA needs to be extended to capture the dynamic nature of system states such as software and policy updates, which could be an obstacle for maintaining its efficiency. On the other hand, PRIMA represents an attestation result with binary decision (trust or distrust) and does not give semantic information about how much the attestee platform can be trusted.

Property-based attestation [7] is an effort to protect the privacy of a platform by collectively mapping related system configurations to attestation properties. For example, "SELinux-enabled" is a property which can be mapped to a system configuration indicating that the system is protected with an SELinux policy. That is, this approach prevents the configurations of a platform from being disclosed to a challenger. However, due to the immense configurations of the hardware and software of the platform, mapping all system configurations to properties is infeasible and impractical.

2.2 Integrity Models

To describe the integrity status for a system, there exist various information flow-based integrity models such as Biba [5], LOMAC [16], Clark-Wilson [13], and CW-Lite [14]. Biba integrity property is fulfilled if a high integrity process cannot read/execute a lower integrity object, nor obtain lower integrity data in any other manner. LOMAC allows high integrity processes to read lower integrity data, while downgrading the process's integrity level to the lowest integrity level that has ever been activated. Clark-Wilson provides a different view of dependencies, which states information can flow from low integrity objects to high integrity objects through a specific program called transaction

procedures (TP). Later, the concept of TP is evolved as a filter in the CW-Lite model. The filter can be a firewall, an authentication process, or a program interface for downgrading or upgrading the privileges of a process.

With existing integrity models, there is a gap between concrete the measurements of a system's components and the verification of its integrity status. We believe an application-oriented and domain-centric approach accommodates the requirements of attestation evaluation better than advocating an abstract model. For example, in a Linux system, a subject in one of traditional integrity models can correspond to a set of processes, belonging to a single application domain. For instance, an Apache domain may include various process types such as `httpd_t`, `http_sysadm_devpts_t`, and `httpd_ prewikka_script_t`. All of these types can have information flows among them, which should be regarded as a single integrity level. Also, sensitive objects in a domain should share the same integrity protection of its subjects. To comprehensively describe the system integrity requirements, in this paper, we propose a domain-based isolation approach as discussed in the subsequent section.

3 Domain-Based Isolation

According to TCG and IMA, the trustworthiness of a system is described with the measured integrity values (hash values) of loaded software components. However, the measurement efficiency and attestation effectiveness are major problems of these approaches since (i) too many components have to be measured and tracked, and (ii) too many known-good hash values are required from different software vendors or authorities. Fundamentally, this requires that in order to trust a single application of a system, every software component in the system has to be trusted; otherwise the attestation result should be negative. In our work, we believe that the trustworthiness of a system is tightly related to the integrity status, which is, in turn, described by a set of integrity rules that are enforced by the system. If any of the rules is violated, it should be detected. Hence, in order to trust a single application domain, we just need to ensure the *system TCB*–including reference monitor and integrity rules protecting the target application domain–are not altered.

Based on this anatomy, we introduce domain-based isolation principles for integrity protection, which are the criteria to describe the integrity status of a system and thus its trustworthiness. We first propose general methodologies to identify high integrity processes, which include *system TCB* and *domain TCB*, and then we specify security rules for protecting these high integrity processes. System TCB (TCB_s) is similar to the concept of traditional TCB [3], which can be identified along with the subjects functioning as the reference monitor of a system [4]. Applying this concept to SELinux [15], for example, subjects functioning as reference monitor such as `checkpolicy` and `loading policy` belong to system TCB. Also, subjects used to support reference monitor such as `kernel` and `initial` should also be included into system TCB. With this approach, an initial TCB_s can be identified, and other subjects such as `lvm` and `restorecon` can be added into TCB_s based on their relationships with the initial TCB_s. Other optional methods for identifying TCB_s are proposed in [10]. Considering the similarity of operating systems and configurations, we expect that the results

would be similar. Furthermore, for attestation purpose, TCB_s also includes integrity measurement and reporting components, such as kernel level integrity measurement agent [1] and attestation request handling agent.

In practice, other than TCB_s, an application or user-space service also can affect the integrity thus the behavior of a system. An existing argument [3] clearly states the situation: "A network server process under a UNIX-like operating system might fall victim to a security breach and compromise an important part of the system's security, yet is not part of the operating system's TCB." Accordingly, a comprehensive mechanism of policy analysis for TCB identification and integrity violation detection is desired. Hence, we introduce a concept called information domain TCB (or simply *domain TCB,* TCB_d). Let *d* be an information domain functioning as a certain application or service through a set of related subjects and objects, domain *d*'s TCB or TCB_d is composed of a set of subjects and objects in information domain *d* which have the same level of security sensitivity. By the same level of security sensitivity, we mean that, if information can flow to some subjects or objects of the domain, it can flow to all others in the domain. That is, they need the same level of integrity protection. Prior to the identification of TCB_d, we first identify the information domain *d* based on its main functions and relevant information flow associated with these functions. For example, a running web server domain consists of many subjects–such as `httpd` process, plugins, tools, and other objects–such as data files, configuration files, and logs.

The integrity of an object is determined by the integrity of subjects that have operations on this object. All objects dependent on TCB_d subjects are classified as TCB protected objects or resources. Thus we need to identify all TCB_d subjects from an information domain and verify the assurance of their integrity. To ease this task, a minimum TCB_d is first discovered. In the situation that the minimum TCB_d subjects have dependency relationships with other subjects, these subjects should be added to domain TCB, or the dependencies should be removed. Based on these principles, we first identify initial TCB_d subjects which are predominant subjects for the information domain *d*. We further discover other TCB_d subjects considering subject dependency relationships with the initial TCB_d through *information flow transitions*, which means that the subjects that can only flow to and from the initial TCB_d subjects should be included into TCB_d. For instance, for a web server domain, `httpd` is the subject that initiates other web server related processes. Hence, `httpd` is the predominant subject and belongs to TCB_d. Then, based on all possible information flows to `httpd`, we identify other subjects such as `httpd-suexec` in TCB_d.

To protect the identified TCB_s and TCB_d, we develop principles similar to those in Clark-Wilson [13]. Clark-Wilson leverages transaction procedures (TP) to allow information flow from low integrity to high integrity processes. Hence, we also develop the concept of filters. Filters can be processes or interfaces [11] that normally are distinct input information channels and are created by a particular operation such as `open()`, `accept()`, or other calls that enable data input. For example, `su` process allows a low integrity process (e.g., staff) being changed to be a high integrity process (e.g., root) by executing `passwd` process, thus `passwd` can be regarded as a filter for processes run by root privilege. Also, high integrity process (e.g., `httpd` administration) can accept low integrity information (e.g, network data) through the secure channel such as `sshd`.

Consequently, `sshd` can be treated as another example of filter for higher privilege processes. With the identifications of TCB_s, TCB_d and filters, for information domain `d`, all the other subjects in a system are categorized as NON-TCB.

Definition 1. Domain-based isolation *is satisfied for an information domain* d *if information flows are from* TCB_d*; or information flows are from* TCB_s *to either* TCB_d *or* TCB_d *protected resources; or information flows are from NON-TCB to either* TCB_d *or* TCB_d *protected resources via filter(s).*

4 Design of DR@FT

DR@FT consists of three main parties: an attestee (the target platform), an attester (attestation challenger), and a trusted authority, as shown in Figure 1. The attestee is required to provide its system state information to the attester to be verified. Here, we assume that an attestee initially is in a *trusted system state*. After certain system behaviors, the system state is changed to a new state.

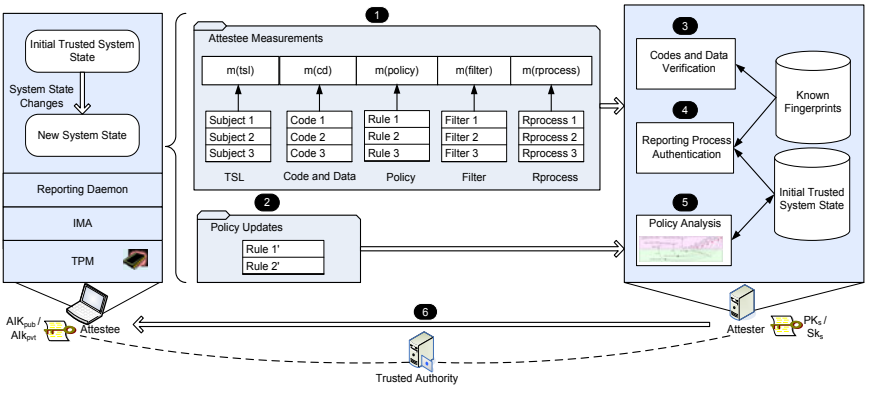


Fig. 1. Overview of DR@FT

An attestation reporting daemon on the attestee gets the measured new state information (step 1) with IMA, and generates the policy updates (step 2). This daemon then gets AIK-signed PCR value(s) and sends to the attester. After the attester receives and authenticates the information, with the attestee's AIK public key certificate from the trusted authority, it verifies the attestee integrity through codes and data verification (step 3), reporting process authentication (step 4) and policy analysis (step 5).

4.1 System State and Trust Requirement

For the attestation purpose, the system state is a snapshot of an attestee system at a particular moment, where the factors characterizing the state can influence the system integrity in any future moment of the system. Based on the domain-based isolation, the attestee system integrity can be represented via information flows, which are characterized by the trusted subject list, filters, policies, and the trustworthiness of TCB_s. Based on these, we define the system state of the attestee as follows.

Definition 2. *A* system state *at the time period i is a tuple T_i=\{ TSL_i, CD_i, $Policy_i$, $Filter_i$, $RProcess_i$ \}, where*

- *TSL_i=\{s_0, s_1....s_n\} represents a set of high integrity processes which corresponds to the set of subjects s_0, s_1....s_n in TCB_s and TCB_d.*
- *CD_i=\{cd (s_0), cd (s_1)....cd (s_n)\} is a set of codes and data for loading a subject $s_j \in TSL_i$.*
- *$Policy_i$ is the security policy currently configured on the attestee.*
- *$Filter_i$ is a set of processes defined to allow information flow from low integrity processes to high integrity processes.*
- *$RProcess_i$ represents a list of processes that measure, monitor, and report the current TSL_i, CD_i, $Filter_i$ and $Policy_i$ information. IMA agent and the attestation reporting daemon are the examples of the $RProcess_i$.*

According to this definition, a system state does not include a particular application's running state such as its memory page and CPU context (stacks and registers). It only represents the security configuration or policy of an attestee system. A system *state transition* indicates one or more changes in TSL_i, CD_i, $Policy_i$, $Filter_i$, or $RProcess_i$. A system state T_i *is trusted* if TSL_i belongs to TCB_s and TCB_d; CD_i does not contain untrusted codes and data; $Policy_i$ satisfies domain-based isolation; $Filter_i$ belongs to the defined filter in domain-based isolation; and $RProcess_i$ codes and data do not contain malicious codes and data and these $RProcess_i$ processes are integrity protected from the untrusted processes via $Policy_i$.

As mentioned, we assume there exists an initial trusted system state T_0 which satisfies. Through changing the variables in T_0, the system transits to states T_1, T_2 ... T_i. The attestation purpose is to verify if any of these states is trusted.

4.2 Attestation Procedures

Attestee Measurements. The measurement at the attestee side has two different forms, depending on *how much* the attestee system changes. In case any subject in TCB_s is updated, the attestee must be fully remeasured from the system reboot and the attester needs to attest it completely, as this subject may affect the integrity of subjects in $RProcess$ of the system such as the measurement agent and reporting daemon. After the reboot and all TCB_s subjects are remeasured, a trusted initial system state T_0 is built. To perform this re-measurement, the attestee measures a state T_i and generates the measurement list M_i which is added by Trusted subject list (TSL_i) measurement, Codes and data (CD_i) measurement,Policy ($Policy_i$) measurement, Filter ($Filter_i$) measurement and Attestation Process ($RProcess_i$) measurement. Also, $\mathcal{H}(M_i)$ is extended to a particular PCR of the TPM, where $\mathcal{H}$ is a hash function such as SHA1.

In another case, where there is no TCB_s subject updated and the TSL_i or $Filter_i$ subjects belonging to TCB_d are updated, the attestee only needs to measure the updated codes and data loading the changed TSL or filter subject, and generates a measurement list M_i. The generation of this measurement list is realized through the run-time measurement supported by the underlying measurement agent.

To support both types of measurements, we develop an attestation reporting daemon which monitors the run-time measurements of the attestee. In case the run-time

measurements for the TCB_s are changed, the attestee is required to be rebooted and fully measured with IMA. The measurements values are then sent to the attester by the daemon. On the other side, the changed measurement value is measured by IMA and captured with the reporting daemon only if the measurement for TCB_d is changed. Obviously, this daemon should be trusted and is included as part of TCB_s. That is, its codes and data are required to be protected with integrity policy and corresponding hash values are required to be stored at the attester side.

Policy Updates. To analyze if the current state of the attestee satisfies domain-based integrity property, the attester requires information about the current security policy loaded at the attestee side. Due to the large volume of policy rules in a security policy, sending all policy rules in each attestation and verifying all of them by the attester may cause the performance overhead. Hence, in DR@FT, the attestee only generates policy updates from the latest attested trusted state and sends them to the attester for the attestation of such updates.

To support this mechanism, we have the attestation reporting daemon monitor any policy update on attestee system and generate a list of updated policy rules. Note that the policy update comparison is performed between the current updated policy and the stored trusted security policy $Policy_0$ or previously attested and trusted $Policy_{i-1}$. The complexity of this policy update algorithm is *O(nr)*, where *nr* represents the number of the policy rules in the new policy file $Policy_i$.

Codes and Data Verification. With received measurement list M_i and AIK-signed PCRs, the attester first verifies the measurement integrity by re-constructing the hash values and compares with PCR values. After this is passed, the attester performs the analyses. Specifically, it obtains the hash values of CD_i and checks if they corresponds to known-good fingerprints. Also, the attester needs to assure that the TSL_i belongs to TCB_s and TCB_d. In addition, the attester also gets the hash value of $Filter_i$ and ensures that they belong to the filter list defined on the attester side. In case this step successes, the attester has the assurance that target processes on attestee side are proved without containing any untrusted code or data, and the attester can proceed to next steps. Otherwise, the attester sends a proper attestation result denoting this situation.

Authenticating Reporting Process. To prove that the received measurements and updated policy rules are from the attestee, the attester authenticates them by verifying that all the measurements, updates and integrity measurement agent processes in the attestee are integrity protected. That is, the $RProcess_i$ does not contain any untrusted codes or data and its measurements correspond to PCRs in the attester. Also, there is no integrity violated information flow to these processes from subjects of TSL_i, according to the domain isolation rules. Note that these components can also be updated, but after any update of these components, the system should be fully re-measured and attested from boot time as aforementioned, i.e., to re-build a trusted initial system state T_0.

Policy Analysis by Attester. DR@FT analyzes policy using a graph-based analysis method. In this method, a policy file is first visualized into a graph, then this policy graph is analyzed against pre-defined security model such as our domain-based isolation, and a policy violation graph is generated. The main goal of this approach is to give

semantic information of attestation result to the attestee, such that its system or security administrator can quickly and intuitively obtain any violated integrity configuration.

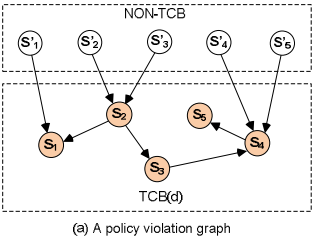
Note that verifying all the security policy rules in each attestation request decrease the efficiency, as loading policy graph and checking all the policy rules one by one cost a lot of time. Thus, we need to develop an efficient way for analyzing the attestee policy. In our method, the attester stores the policy of initial trusted system state T_0 or the latest trusted system state T_i, and its corresponding policy graph is loaded which does not have any policy violation. Upon receiving the updated information from the attestee, the attester just needs to analyze these updates to see if there is new information flow violating integrity.

Through this algorithm, rather than analyzing all the policy rules and all information flows for each attestation, we verify the new policy through only checking the updated policy rules and the newly identified information flow. The complexity of this policy analysis algorithm is *O(nn +nl +nt)*, where *nn* represents the number of changed subjects and objects, *nl* is the number of changed subjects and objects relationship in the policy update file; and *nt* represents the number of changed TCB in the TSL file.

Attestation Result Sending to Attester. In case the attestation is successful, a new trusted system state is developed and the corresponding information is stored at the attester side for subsequent attestations. On the other hand, if the attestation fails, there are several possible attestation results including CD_i *Integrity Fail*, CD_i *Integrity Success,* $RProcess_i$ *Unauthenticated, and* $Policy_i$ *Fail/Success*, and CD_i *Integrity Success,* $RProcess_i$ *Authenticated, and* $Policy_i$ *Fail /Success*. To assist the attestee reconfiguration, the attester also sends a representation of the policy violation graph to the attestee. Moreover, with this policy violation graph, the attester specifies the violation ranking and the trustworthiness of the attestee, which is explained in next section.

5 Integrity Violation Analysis

As we discussed in Section 1, existing attestation solutions such as TCG and IMA lack the expressiveness of the attestation result. In addition to their boolean-based response for attestation result, DR@FT adopts a graph-based policy analysis mechanism, where a policy violation graph can be constructed for identifying all policy violations on the



(a) A policy violation graph

Subject	SubjectRank
S_1	0.2799
S_2	0.3999
S_3	0.0799
S_4	0.4319
S_5	0.3456

(b) SubjectRank in TCB(d)

Path	PathRank
$\langle S'_1, S_1 \rangle$	0.2799
$\langle S'_2, S_2 \rangle$	0.8104
$\langle S'_3, S_2 \rangle$	0.8104
$\langle S'_4, S_4 \rangle$	0.6048
$\langle S'_5, S_4 \rangle$	0.6048

(c) PathRank between NON-TCB and TCB(d)

Fig. 2. Example policy violation graph and rank. The SubjectRank and PathRank indicate the severity of violating paths.

attestee side. We further introduce a risk model built on a ranking scheme, which gives the implication of how severe the discovered policy violations are, and how to efficiently resolve them.

5.1 Policy Violation Graph

Based on domain-based isolation model, we can find out two kinds of *violation paths*, *direct violation paths* and *indirect violation paths*. A direct violation path is a one-hop path through which an information flow can go from a low integrity subject to a high integrity subject. We observe that information flows are transitive in general. Therefore, there may exist information flows from a low integrity subject to a high integrity subject via several other subjects. This multi-hop path is called indirect violation path. All direct and indirect violation paths belonging to a domain can construct a policy violation graph for this domain.

Definition 3. *A policy violation graph for a domain* d *is a directed graph* $G^v = (V^v, E^v)$*:*

- $V^v \subseteq V^v_{NTCB} \cup V^v_{TCBd} \cup V^v_{TCB}$ *where* V^v_{NTCB}, V^v_{TCBd} *and* V^v_{TCB} *are subject vertices containing in direct or indirect violation paths of domain* d *and belong to NON-TCB,* TCB_d*, and* TCB_s*, respectively.*
- $E^v \subseteq E^v_{Nd} \cup E^v_{dT} \cup E^v_{NT} \cup E^v_{NTCB} \cup E^v_{TCBd} \cup E^v_{TCB}$ *where* $E^v_{Nd} \subseteq (V^v_{NTCB} \times V^v_{TCBd})$, $E^v_{dT} \subseteq (V^v_{TCBd} \times V^v_{TCB})$, $E^v_{NT} \subseteq (V^v_{NTCB} \times V^v_{TCB})$, $E^v_{NTCB} \subseteq (V^v_{NTCB} \times V^v_{NTCB})$, $E^v_{TCBd} \subseteq (V^v_{TCBd} \times V^v_{TCBd})$, *and* $E^v_{TCB} \subseteq (V^v_{TCB} \times V^v_{TCB})$, *and all edges in* E^v *are contained in direct or indirect violation paths of domain* d*.*

Figure 2 (a) shows an example of policy violation graph which examines information flows between NON-TCB and TCB_d[1]. Five direct violation paths are identified in this graph: ¡S'_1, S_1¿, ¡S'_2, S_2¿, ¡S'_3, S_2¿, ¡S'_4, S_4¿, and ¡S'_5, S_4¿, crossing all the boundaries between NON-TCB and TCB_d. Also, eight indirect violation paths exist. For example, ¡S'_2, S_5¿ is a four-hop violation path passing through other three TCB_d subjects S_2, S_3, and S_4.

5.2 Ranking Policy Violation Graph

In order to explore more features of policy violation graphs and facilitate efficient policy violation detection and resolution, we introduce a scheme for ranking policy violation graphs. There are two steps to rank a policy violation graph. First, TCB_d subjects in the policy violation graph are ranked based on dependency relationships among them. The rank of a TCB_d subject shows reachable probability of low integrity information flows from NON-TCB subjects to the TCB_d subject. In addition, direct violation paths in the policy violation graph are evaluated based on the ranks of TCB_d subjects to indicate severity of these paths which allow low integrity information to reach TCB_d subjects. The ranked policy violation graphs are valuable for a system administrator

[1] Similarly, the information flows between NON-TCB and TCB_s, and between TCB_d and TCB_s can be examined accordingly.

as they need to estimate the *risk level* of a system and provide a guide for choosing appropriate strategies for resolving policy violations efficiently.

Ranking Subjects in TCB_d. Our notation of *SubjectRank* (SR) in policy violation graphs is a criterion that indicates the likelihood of low integrity information flows coming to a TCB_d subject from NON-TCB subjects through direct or indirect violation paths. The ranking scheme we introduce in this section adopts a similar process of rank analysis applied in hyper-text link analysis system, such as Google's PageRank [6] that utilizes a link structure provided by hyperlinks between web pages to gauge their importance. Comparing with PageRank which focuses on analyzing a web graph where the entries are *any* web pages contained in the web graph, the entries of low integrity information flows to TCB_d subjects in a policy violation graph are only identified NON-TCB subjects.

Consider a policy violation graph with *N* NON-TCB subjects, and s_i is a TCB_d subject. Let $N(s_i)$ be the number of NON-TCB subjects from which low integrity information flows could come to s_i, $N^{'}(s_i)$ the number of NON-TCB subjects from which low integrity information flows could *directly* reach to s_i, $In(s_i)$ a set of TCB_d subjects pointing to s_i, and $Out(s_j)$ a set of TCB_d subjects pointed from s_j. The probability of low integrity information flows reaching a subject s_i is given by:

$$SR(s_i) = \frac{N(s_i)}{N}\left(\frac{N^{'}(s_i)}{N(s_i)} + \left(1 - \frac{N^{'}(s_i)}{N(s_i)}\right) \sum_{s_j \in In(s_i)} \frac{SR(s_j)}{|Out(s_j)|}\right) \quad (1)$$

SubjectRank can be interpreted as a *Markov Process*, where the states are TCB_d subjects, and the transitions are the links between TCB_d subjects which are all evenly probable. While a low integrity information flow attempts to reach a high integrity subject, it should select an entrance (a NON-TCB subject) which has the path(s) to this subject. Thus, the possibility of selecting correct entries to a target subject is $\frac{N(s_i)}{N}$. After selecting correct entries, there still exist two ways, through direct violation paths or indirect violation paths, to reach a target subject. Therefore, the probability of flow transition from a subject is divided into two parts: $\frac{N^{'}(s_i)}{N(s_i)}$ for direct violation paths and $1 - \frac{N^{'}(s_i)}{N(s_i)}$ for indirect violation paths. The $1 - \frac{N^{'}(s_i)}{N(s_i)}$ mass is divided equally among the subject's successors s_j, and $\frac{SR(s_j)}{|Out(s_j)|}$ is the rank value derived from s_j.

Figure 2 (b) displays a result of applying Equation (1) to the policy violation graph showing in Figure 2 (a). Note that even though subject s_4 has two direct paths from NON-TCB subjects like subject s_2, the rank value of s_4 is higher than the rank value of s_2, because there is another indirect flow path to s_4 (via s_3).

Ranking Direct Violation Path. We further define *PathRank* (PR) as the rank of a direct violation path[2], which is a criterion reflecting the severity of the violation path through which low integrity information flows may come to TCB_d subjects. Direct violation paths are regarded as the entries of low integrity data to TCB_d in policy violation

[2] It is possible that a system administrator may also want to evaluate indirect violation paths for violation resolution. In that case, our ranking scheme could be adopted to rank indirect violation paths as well.

graph. Therefore, the ranks of direct violation paths give a guide for system administrator to adopt suitable defense countermeasures for solving identified violations.

To calculate *PathRank* accurately, three conditions are needed to be taken into account: (1) the number of TCB_d that low integrity flows can reach through this direct violation path; (2) SubjectRank of reached TCB_d subjects; and (3) the number of hops to reach a TCB_d subject via this direct violation path.

Suppose $< s_i^{'}, s_j >$ is a direct violation path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_j in a policy violation graph. Let $Reach(< s_i^{'}, s_j >)$ be a function returning a set of TCB_d subjects to which low integrity information flows may go through a direct violation path $< s_i^{'}, s_j >$, $SR(s_l)$ the rank of a TCB_d subject s_l, and $H_s(s_i^{'}, s_l)$ a function returning the hops of the shortest path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_l. The following equation is utilized to compute a rank value of the direct violation path $< s_i^{'}, s_j >$.

$$PR(< s_i^{'}, s_j >) = \sum_{s_l \in Reach(< s_i^{'}, s_j >)} \frac{SR(s_l)}{H_s(s_i^{'}, s_l)} \tag{2}$$

Figure 2 (c) shows the result using the above-defined equation to calculate the *PathRank* of the example policy violation graph. For example, $< s_2^{'}, s_2 >$ has a higher rank than $< s_1^{'}, s_1 >$, because $< s_2^{'}, s_2 >$ may result in low integrity information flows to reach more or important TCB_d subjects than $< s_1^{'}, s_1 >$.

5.3 Evaluating Trustworthiness

Let P_d be a set of all direct violation paths in a policy violation graph. The entire rank, which can be considered as a risk level of the system, can be computed as follows:

$$RiskLevel = \sum_{< s_i^{'}, s_j > \in P_d} PR(< s_i^{'}, s_j >) \tag{3}$$

The calculated risk level could reflect the trustworthiness of an attestee. Generally, the lower risk level indicates the higher trustworthiness of a system. When an attestation is successful and there is no violation path being identified, the risk level of the attested system is *zero*, which means an attestee has the highest trustworthiness. On the other hand, when an attestation is failed, corresponding risk level of a target system is computed. A *selective service* could be achieved based on this fine-grained attestation result. That is, the number of services provided by a service provider to the target system may be decided with respect to the trust level of the target system. On the other hand, a system administrator could refer to this attestation result as the evaluation of her system as well as guidelines since this quantitive response would give her a proper justification to adopt countermeasures for improving the system's trustworthiness.

6 Implementation and Evaluation

We have implemented DR@FT to evaluate its effectiveness and performance. Our attestee platform is a Lenovo ThinkPad X61 with Intel Core 2Duo Processor L7500

1.6GHz, 2 GB RAM, and Atmel TPM. We enable SELinux with the default policy based on the current distribution of SELinux [15]. To measure the attestee system with TPM, we update the Linux kernel to 2.6.26.rc8 with the latest IMA implementation [1], where SELinux hooks and IMA functions are enabled. Having IMA enabled, we configure the measurement of the attestee information. After the attestee system kernel is booted, we mount the `sysfs` file system and inspect the measurement list values in `ascii_runtime_measurements` and `ascii_bios_measurements`.

6.1 Attestation Implementation

We start from a legitimate attestee and make measurements of the attestee system for the later verification. To invoke a new attestation request from the attester, the attestation reporting daemon runs in the attestee and monitors the attestee system. This daemon is composed of two main threads: One monitors and gets the new system state measurements, and the other monitors and obtains the policy updates of the attestee. The daemon is also measured and the result can be obtained through the legitimate attestee. Thus the integrity of the daemon can be verified later by the attester. In case the attestee system state is updated due to new software installation, changing policy, and so on, an appropriate thread of the daemon automatically obtains the new measurement values as discussed in 4. The daemon then securely transfers the attestation information to the attester based on the security mechanisms supported by the trusted authority.

After receiving the updated system information from the attestee, the measurement module of the attester checks the received measurements against the stored PCR to prove its integrity. To analyze the possible revised attestee policy, the policy analysis module is developed as a daemon, which is ported from a policy analysis engine. We extend the engine to identify violated information flows from the updated policy rules based on domain-based isolation rules. We also accommodate the algorithm presented in Section 4.2, as well as our rank scheme to evaluate the trustworthiness of the attestee.

6.2 Evaluation

To assess the proposed attestation framework, we attest our testbed platform with Apache web server installed. To configure the trusted subject list of the Apache domain, we first identify the TCB_s based on the reference monitor-based TCB identification, including the integrity measurement, monitoring agents, and daemon. For TCB_d of the Apache, we identify the Apache information domain, Apache TCB_d, including `httpd_t` and `httpd_suexec_t`, and the initial filters `sshd_t, passwd_t, su_t`, through the domain-based isolation principles. Both TCB_s and TCB_d are identified with a graphical policy analysis tool [17]. We then install the unverified codes and data to evaluate the effectiveness of our attestation framework.

Installing Malicious Code. We first install a Linux rootkit, which gains administrative control without being detected. Here, we assign the rootkit with the domain `unconfined_t` that enables information flows to domain `initrc_t` labeling `initrc` process, which belongs to TCB_s of the attestee. Following the framework proposed in Section 4, the attestee system is measured from the bootstrap with configured IMA. After

getting the new measurement values, the reporting daemon sends these measurements to the attester. Note that there is no policy update in this experiment. Different from IMA, we only measure the TCB_s and TCB_d subjects. After getting the measurements from the attestee, attester verifies them by trying to match the measured hash values. Partial of our measurement shows the initial measurements of the `initrc` (in a trusted initial system state) and the changed value because of the installed rootkit. The difference between these two measurements indicates the original `initrc` is altered, and the attester confirms that the attestee is not in a trusted state.

Installing Vulnerable Software. In this experimentation, we install a vulnerable software called Mplayer on the attestee side. Mplayer is a media player and encoder software which is susceptible to several integer overflows in the real video stream dumuxing code. These flaws allow an attacker to cause a denial of service or potentially execution of the arbitrary code by supplying a deliberately crafted video file. After a Mplayer is installed, a Mplayer policy module is also loaded into the attestee policy. In this policy module, there are several different subjects such as `staff_mplayer_t`, `sysadm_mplayer_t`. Also, some objects are defined in security policies such as `user_mplayer_home_t` and `staff_mplayer_home_t`.

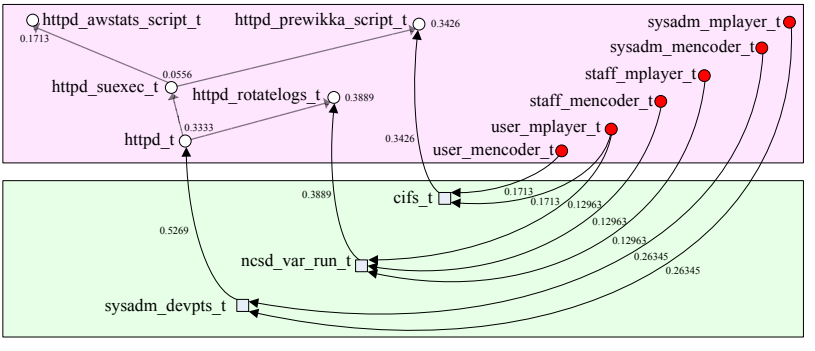


Fig. 3. Information flow verification of Mplayer. The links show the information flow from Mplayer (filled circle nodes) to Apache (unfilled nodes). The rank values on the paths indicate the severity of the corresponding violation paths.

After the Mplayer is installed, the attestation daemon finds that the new measurement of Mplayer is generated and the security policy of the system is changed. As the Mplayer does not belong to TCB_s and Apache TCB_d, the attestation daemon does not need to send the measurements to the attester. Consequently, the daemon only computes the security policy updates and sends the information to the attester.

Upon receiving the updated policies, we analyze these updates and obtain a policy violation graph as shown in Figure 3. Through objects such as `cifs_t, sysadm_devtps_t, ncsd_var_run_t`, information flows from Mplayer can reach Apache domain. In addition, rank values are calculated and shown in the policy violation graph, which guides effective violation resolutions. For example, there are three higher ranked

paths including path from sysadm_devpts_t to httpd_t, from ncsd_var_run_t to httpd_rotatelogs_t, and from cifs_t to httpd_prewikka_script_t. Meanwhile, a risk level value (1.2584) reflecting the trustworthiness of the attestee system is computed based on the ranked policy violation graph.

Once receiving the attestation result shown in Figure 3, the attestee administrator solves the violation that has the higher rank than others. Thus, the administrator can first resolve the violation related to httpd_t through introducing httpd_sysadm_devpts_t.

```
allow httpd_t httpd_sysadm_devtps_t:chr_file {ioctl read write
getattr lock append};
```

After the policy violation resolution, the risk level of the attestee system is lowered to *0.7315*. Continuously, after the attestee resolves all the identified policy violations and the risk level is decreased to be *zero*, the attestation daemon gets a new policy update file and sends it to the attester. Upon receiving this file, the attester verifies whether these information flows violate domain-based isolation integrity rules since these flows are within the NON-TCB–even though there are new information flow compared to the stored $Policy_0$. Thus, an attestation result is generated which specifies the risk level (in this case, *zero*) of the current attestee system. Consequently, a new trusted system state is built for the attestee. In addition, the information of this new trusted system state is stored in the attester side for the later attestation.

6.3 Performance

To examine the scalability and efficiency of DR@FT, we investigate how well the attestee measurement agent, attestation daemon, and the attester policy analysis module scale along with the increased complexity, and how efficiently DR@FT performs by comparing it with the traditional approaches.

In DR@FT, the important situations influencing the attestation performance include system updates and policy changes. Hence, we evaluate the performance of DR@FT by changing codes and data to be measured and modifying the security policies. Based on our study, we observe that normal policy increased or decreased no more than 40KB when installing or uninstalling software. Also, a system administrator does not make the enormous changes over the policy. Therefore the performance is measured with the range from zero to around 40KB in terms of policy size.

Performance on the attestee side. Based on DR@FT, the attestee has three main factors influencing the attestation performance. (1) Time spent for the measurement: Based on our experimentation, the measurement time increases roughly linearly with the size of the target files. For example, measuring policy files with 17.2MB and 20.3MB requires 14.63 seconds and 18.26 seconds, respectively. Measuring codes around 27MB requires 25.3sec. (2) Time spent for identifying policy updates $T_{Pupdate}$: Based on the specification in Section 4, policy updates are required to be identified and sent to the attester. As shown in Table 1, for a system policy which is the size of 17.2MB at its precedent state, the increase of the policy size requires more time for updating the

Table 1. Attestation Performance Analysis (in seconds)

Policy Change	Dynamic				Static		
Size	$T_{Pupdate}$	T_{send}	$T_{Panalysis}$	Overhead	T_{Psend}	$T_{Panalysis}$	Overhead
0	0.23	0	0	0.23	14.76	90.13	104.89
-0.002MB (Reduction)	0.12	0.002	0.02	0.14	14.76	90.11	104.87
-0.019MB (Reduction)	0.08	0.01	0.03	0.12	14.74	89.97	104.34
-0.024MB (Reduction)	0.04	0.02	0.03	0.09	14.74	89.89	104.23
0.012MB (Reduction)	0.37	0.01	0.03	0.41	14.77	90.19	104.96
0.026MB (Addition)	0.58	0.02	0.03	0.63	14.78	90.33	105.11
0.038MB (Addition)	0.67	0.03	0.04	0.74	14.79	90.46	105.25

policy and vice versa. (3) Time spent for sending policy updates T_{Psend}: Basically, the more policy updates, the higher overhead was observed.

Performance on the attester side. In DR@FT, the measurement verification is relatively straightforward. At the attester side the time spent for policy analysis $T_{Panalysis}$ mainly influences its performance. As shown in Table 1, the analysis time roughly increases when the policy change rate increases.

Comparison of dynamic and static attestation. To further specify the efficiency of DR@FT, we compare the overhead of DR@FT with a static attestation. In the static approach, the attestee sends all system state information to an attester, and the attester verifies the entire information step by step. As shown in Table 1, the time spent for static attestation is composed of T_{Psend} and $T_{Panalysis}$, which represent the time for sending policy module and analyzing them, respectively. Obviously, the dynamic approach can dramatically reduce the overhead compared to the static approach. It shows that DR@FT is an efficient way when policies on an attestee are updated frequently.

7 Conclusion

We have presented a dynamic remote attestation framework called DR@FT for efficiently verifying if a system satisfies integrity protection property and indicates integrity violations which determine its trustworthiness level. The integrity property of our work is based on an information flow-based domain isolation model, which is utilized to describe the integrity requirements and identify integrity violations of a system. To achieve the efficiency and effectiveness of remote attestation, DR@FT focuses on system changes on the attestee side. We have extended a powerful policy analysis engine to represent integrity violations with the rank scheme. In addition, our results showed that our dynamic approach can dramatically reduce the overhead compared to static approach. We believe such an intuitive evaluation method would help system administrators reconfigure the system with more efficient and strategic manner.

There are several limitations of our attestation framework. First, DR@FT can attest dynamic system configurations, but it does not attest the trustworthiness of dynamic contents such as application state and CPU context. Second, our risk evaluation does not explain under what kind of condition, what range of the risk value is acceptable for the attestee or attester. In addition, in our work, all verification work is done at the

attester side. There is a possibility that the attester can delegates some attestation tasks to trusted components at the attestee side. In the future, we would further investigate these issues.

References

1. LIM Patch, http://lkml.org/lkml/2008/6/27
2. Trusted computing group, https://www.trustedcomputinggroup.org/home
3. Trusted Computer System Evaluation Criteria. United States Government Department of Defense (DOD), Profile Books (1985)
4. Anderson, A.P.: Computer security technology planning study. ESD-TR-73-51 II (1972)
5. Biba, K.J.: Integrity consideration for secure compuer system. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass. (1977)
6. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer networks and ISDN systems 30(1-7), 107–117 (1998)
7. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R., Stüble, C.: A protocol for property-based attestation. In: ACM STC (2006)
8. Haldar, V., Chandra, D., Franz, M.: Semantic remote attestation: a virtual machine directed approach to trusted computing. In: USENIX Conference on Virtual Machine Research And Technology Symposium (2004)
9. Jaeger, T., Sailer, R., Shankar, U.: Prima: policy-reduced integrity measurement architecture. In: ACM SACMAT (2006)
10. Jaeger, T., Sailer, R., Zhang, X.: Analyzing integrity protection in the selinux example policy. In: USENIX Security (2003)
11. Provos, N., Friedl, M., Honeyman, P.: Preventing privilege escalation. In: 12th USENIX Security Symposium, p. 11 (August 2003)
12. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: USENIX Security (2004)
13. Sandhu, R.S.: Lattice-based access control models. IEEE Computer 26(11), 9–19 (1993)
14. Shankar, U., Jaeger, T., Sailer, R.: Toward automated information-flow integrity verification for security-critical applications. In: NDSS (2006)
15. Smalley, S.: Configuring the selinux policy (2003), http://www.nsa.gov/SELinux/docs.html
16. Fraser, T.: Lomac: Low water-mark integrity protection for cots environment. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2000)
17. Xu, W., Zhang, X., Ahn, G.-J.: Towards system integrity protection with graph-based policy analysis. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 65–80. Springer, Heidelberg (2009)

Website Fingerprinting and Identification Using Ordered Feature Sequences

Liming Lu, Ee-Chien Chang, and Mun Choon Chan*

Department of Computer Science, School of Computing
National University of Singapore
{luliming,changec,chanmc}@comp.nus.edu.sg

Abstract. We consider website fingerprinting over encrypted and proxied channel. It has been shown that information on packet sizes is sufficient to achieve good identification accuracy. Recently, traffic morphing [1] was proposed to thwart website fingerprinting by changing the packet size distribution so as to mimic some other website, while minimizing bandwidth overhead. In this paper, we point out that packet ordering information, though noisy, can be utilized to enhance website fingerprinting. In addition, traces of the ordering information remain even under traffic morphing and they can be extracted for identification. When web access is performed over OpenSSH and 2000 profiled websites, the identification accuracy of our scheme reaches 81%, which is 11% better than Liberatore and Levine's scheme presented in CCS'06 [2]. We are able to identify 78% of the morphed traffic among 2000 websites while Liberatore and Levine's scheme identifies only 52%. Our analysis suggests that an effective countermeasure to website fingerprinting should not only hide the packet size distribution, but also aggressively remove the ordering information.

Keywords: Traffic analysis, side channel attack, edit distance, privacy, anonymity.

1 Introduction

Website fingerprinting aims to identify the website accessed in some low latency, encrypted tunnel. The "fingerprint" of a website is typically profiled on side channel features observed from the stream of packets sent and received to access the website. Let us call such stream the HTTP stream. From the perspective of web clients, website fingerprinting raises the privacy concern and indicates the need for further anonymization. On the other hand, such technique aids legitimate wardens track web accesses over encrypted channel.

Since neither the IP address nor domain name of the website is available due to encryption and proxies, it is natural to fingerprint websites using packet size related features [2] [3] [4]. Simple defenses to website fingerprinting have been proposed in previous works, including variations of packet padding. However,

* This work is partially supported by Grant R-252-000-413-232/422/592 from TDSI.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 199–214, 2010.

the large bandwidth overhead they cost leads to insufficient incentives for deployment. "Traffic morphing" [1] is designed as a scheme to evade detection by warden, and at the same time, incurs small bandwidth overhead. Traffic morphing alters the packet size distribution of an HTTP stream, to make it appear as if it is from another website. A morphing matrix is pre-computed to transform a source distribution to the target, minimizing the total increase in packet sizes.

In this paper, we propose a website fingerprinting scheme that exploits information on packet ordering as well as on packet sizes. Our scheme is motivated by an observation on the regularity of browsers' behavior in retrieving a webpage through HTTP, which leads to a robust ordering of certain packets transmitted. Experimental results show that sufficient ordering information is retained for accurate website fingerprinting under reasonable amount of noise.

Our scheme is evaluated on a classification problem that identifies an HTTP stream among a list of profiled websites, as well as a detection problem in which it is uncertain whether the test website belongs to the profiled set. We tested our scheme on datasets containing up to 2000 websites, with traces collected over two months. For the classification problem, we test our scheme with full-length OpenVPN test streams, i.e., 30-second long encrypted HTTP streams tunneled through a VPN server, and the fingerprint identification accuracy is 97% among 1000 websites. We also test with partial SSH traces that capture only the first few seconds of encrypted communication, the accuracy is 81% among 2000 websites, still over 10% better than the previous scheme. For the detection problem, our scheme presents an equal error rate at 7%, which is significantly better than the previous scheme at 20%. Hence, for both types of evaluations, our scheme yields superior identification accuracy compared to previous schemes.

Furthermore, our scheme is resilient to the traffic morphing technique by Wright *et al.* [1]. When traffic is morphed according to unigram (i.e. single packet size) distribution, our scheme distinguishes the morphed traffic from its target with a success rate of 99% versus 25% for the previous scheme. When traffic is morphed according to bigram (i.e. two adjacent packet sizes) distribution, our scheme distinguishes 98.7% versus 22.5% for the previous scheme. When bigram morphed traffic are mixed with traces of 2000 other websites, our scheme correctly identifies 78% of the morphed traffic, while the previous scheme identifies 52%. Of course, our scheme may not improve the fingerprinting accuracy if packets are padded or randomly morphed without concern of the bandwidth overhead. For example, padding all packets to the path maximum transmission unit will remove the ordering information, thus no improvement will be made.

In addition, we analyze the consistency of our website fingerprints, with respect to static and dynamic websites, and different HTTP pipelining configurations. We find that only 6% of the websites need reprofiling after a month. The results verify that our feature selection generates consistent fingerprints.

Our analysis suggests that both size and ordering features should be removed in countermeasures to website fingerprinting. We propose some countermeasures that engender randomization in packets or HTTP requests. The countermeasures are low in bandwidth overhead, though they may trade off the response time.

Here we summarize the main contributions of our paper. (*i*) We provide a concrete and efficient scheme that utilizes packet ordering information for website fingerprinting. It improves the identification accuracy over previous schemes. (*ii*) The website fingerprinting scheme we propose is able to withstand traffic morphing that considers little packet ordering information.

The rest of this paper is organized as follows. The related work is reviewed in Sect. 2. Our website fingerprinting model is presented in Sect. 3. Our method to handle traffic morphing is described in Sect. 4. Experiments and analysis on the effectiveness and robustness of our scheme are presented in Sect. 5. We suggest some countermeasures in Sect. 6 and conclude this paper in Sect. 7.

2 Related Work

Side channel information has been utilized in a wide range of traffic analysis applications. For example, keystroke intervals are used for password guessing over SSH [5]; the sequence of bit rates is used to identify the streaming video encoded with variable bit rate schemes [6]; packet sizes and rates are used to reveal the presence of Skype traffic [7]; packet size, timing and direction are used to infer the application layer protocol [8].

Several recent works [2] [3] [4] [9] have looked at the issue of using traffic analysis to identify websites accessed. Yet some of their assumptions have been invalidated with changes in HTTP and web browser behaviors. For example, browsers were assumed not to reuse a TCP connection to download multiple embedded objects [3] and object requests were assumed non-pipelined [4]. Hence their approaches to determine an object size by accumulating the data received either through a TCP connection [3] or between adjacent HTTP requests [4] no longer work. Liberatore *et al.* [2] used the set composed of (direction, packet size) pairs as fingerprint, and used Jaccard's coefficient as the similarity measure ($\frac{|X \cap Y|}{|X \cup Y|}$, where X and Y are the sets representing a website profile and a test fingerprint, and $|A|$ denotes the size of set A). These schemes only exploit side channel features related to sizes but not ordering.

Defences against website fingerprinting have been proposed, such as variants of packet padding. Padding every packet to the size of path Maximum Transmission Unit (MTU) can thwart size related traffic analysis, but the amount of overhead it causes is nearly 150% of the actual data [2]. "Mice-elephant" packet padding incurs relatively less overhead, at nearly 50% growth in the data transmitted. It pads packets to two sizes, either a small size for control packets, or to the path MTU for data packets. Wright *et al.* [1] proposed traffic morphing as a bandwidth efficient alternative to packet padding. It transforms the source packet size distribution to mimic a target website, by splitting or padding the packets. With limited consideration on packet ordering, the morphing technique targets at fingerprinting schemes that only exploit information on packet sizes.

Edit distance [10] is employed in our scheme to measure the similarity of fingerprints. It computes the minimum total cost of insertion, deletion and substitution to make two sequences identical. Edit distance has been applied in a

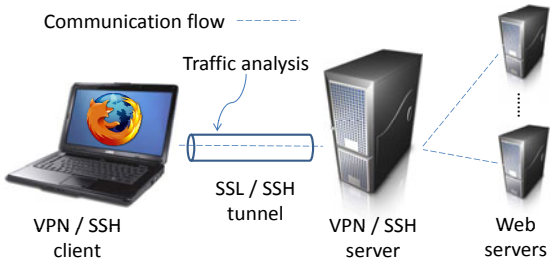


Fig. 1. An illustration of traffic analysis setup

wide range of applications, such as spell checker in Google [11] and in Microsoft Word [12], identifying plagiarism, comparing DNA sequences [13] [14], conducting fuzzy search in EXCEL [15] and evaluating dialect distances [16]. Our work is the first in applying it to match the features of network traffic.

3 Our Website Fingerprinting Scheme

Web contents are retrieved from the server using HTTP and presented by the client browser. SSH, SSL or its successor TLS provides a layer of protection to data secrecy by encrypting the HTTP traffic before transmitting them over TCP/IP. An SSH or SSL/TLS server acts as a proxy, providing a tunnel such that the web server address is encapsulated into the packet payload and encrypted. Website fingerprinting is performed over the encrypted and tunneled HTTP stream.

3.1 Model

We use passive traffic analysis to fingerprint websites. Two important observations enable us to adopt this model: (*i*) webpage download by HTTP is highly structured; and (*ii*) encryption and proxy do not severely alter the packet sizes, nor the packet ordering. The advantage of passive traffic analysis is that the presence of a warden is completely transparent.

The HTTP streams for analysis are captured over the protected tunnel between client and the VPN or SSH server, as shown in Fig. 1. Website fingerprints are extracted from the HTTP streams. Several fingerprint instances form the profile of a website. A testing stream is compared to each profile for identification.

Traffic model. Our fingerprinting model supports most browser configurations and accommodates the following connection patterns (*i*) multiple TCP connections can be opened to download in parallel different embedded objects, (*ii*) each TCP connection can be reused to download multiple objects, (*iii*) HTTP requests can be pipelined in each connection, and (*iv*) all sessions are multiplexed onto a single port.

The connection patterns speed up the webpage download, but they increase the difficulty to fingerprint websites. HTTP pipelining means a client is allowed to make multiple requests without waiting for each response. Because of multiple TCP connections and HTTP pipelining, data packets of different embedded objects can interleave, thus object sizes cannot be determined by accumulating the amount of data received between consecutive HTTP requests. Multiplexing communication sessions hides the number of TCP connections opened and the number of objects in a webpage.

As in previous works [2] [3] [4] [9], we assume browser caching is disabled. If it is enabled, we can determine if a user has accessed some sensitive website using the attack in [17].[1]

Problem scenarios. We tackle two problem scenarios, classification and detection. In both scenarios, a set D of, say 1000, websites are profiled.

- **Classification.** Given a test stream which is known to be a visit to a website in D, identify the website. This is the same problem addressed in previous work.
- **Detection.** Given a test stream, determine whether it is a visit to a website in D, and identify the website if it is. We examine the false positive rate (FPR) and false negative rate (FNR) in identification. This problem has not been addressed in previous work.

The classification scenario requires the fingerprints of a website be sufficiently similar. While the detection scenario further requires fingerprints of different websites be sufficiently dissimilar.

Noise model. There are a few sources of noise that degrades the consistency of HTTP streams, even if the website contents have not changed. Connection multiplexing and HTTP pipelining are the main sources affecting the ordering of objects. In addition, the order of object requests may be browser specific, and the dynamics in network condition causes some random noise.

3.2 Fingerprint Feature Selection

A straightforward approach to represent fingerprint of an HTTP stream is by using the sequence of all packet sizes and directions, i.e. fingerprint $F = \langle(s_1, d_1), (s_2, d_2), ...\rangle$ where s_i is i-th packet size, and d_i is its direction. Clearly, the approach makes fingerprint comparison inefficient, since each sequence easily contains over a thousand elements. Yet more importantly, the identification accuracy will be affected, because the ordering of some packets often changes due to various noises. Hence, we apply domain knowledge to select features.

Firstly, packets with sizes smaller than a threshold are considered control packets and discarded from fingerprints. Secondly, we keep only the non-MTU

[1] User is issued requests on some of the website's objects. From the response time, we determine if the objects are cached and hence if the user has visited the website.

(Maximum Transmission Unit) downloading packets as features of HTTP responses. The reasons why we exclude the MTU downloading packets are explained below.

Ideally, we would like to monitor object sizes rather than packet sizes, as objects are less variable and more characteristic to a website. However, data of different objects interleave in transmission due to multiple TCP connections and HTTP pipelining. It is difficult to associate the data blocks to their respective objects. Fortunately, HTTP transmission is not random. Majority of web servers transfer data in chunks. In each data chunk, all packets are sized as path MTU, except the last packet transferring the remaining data. Therefore, although we cannot estimate object sizes, we can leverage the last packet size of a data chunk, as it is specific to an object component. As for packet ordering, packets that change their order tend to be the intermediate packets of different objects. By filtering the intermediate packets, we reduce the probability of fingerprint inconsistency.

Thirdly, we design the fingerprint of an HTTP stream to be two sequences, representing the request and response features respectively. Although a request must precede its corresponding response, the order of other requests to this response is not deterministic, but sensitive to the pipelining configuration. Hence the request and response features are not merged into a single sequence.

Therefore, to fingerprint a website over encrypted tunnels, the side channel features we select are (i) **sequence of HTTP request sizes** and (ii) **sequence of HTTP response sizes** (except MTU packets). Note that *the number of objects* and *the number of components in object responses* are two implicit features represented by the lengths of these two sequences.

The features we select closely adhere to the webpage content and layout. The request sequence reveals the relative locations of embedded objects in a webpage and their URL lengths. The response sequence indicates the download completion order of object components and the sizes of their last packets.

3.3 Fingerprint Similarity Measurement

Edit distance measures the number of edit operations to make two strings identical. *Levenshtein distance* is a form of edit distance that allows insertion, deletion or substitution of a single character. It is commonly computed using Wagner-Fischer's algorithm [18]. We use Levenshtein distance to measure the similarity between two website fingerprints.

We normalize the edit distance by $\max(|x|, |y|)$, where $|x|$ denotes the length of sequence x. We define $similarity = 1 - distance$, to convert the distance measurement into similarity. Given two fingerprints, two similarity values measured on the object request sequences and on the non-MTU response sequences are combined as follows:

$$\alpha \cdot sim_{HTTPget} + (1 - \alpha) \cdot sim_{nonMTUpkts},$$

where α is a tunable parameter, indicating the degree of reliability of the similarity values. We set $\alpha = 0.6$ in our experiments, as the sequence of object requests

is more stable. The probability of a test stream coming from a particular website is determined as the maximum similarity between the fingerprint of this stream and the various fingerprints in the website profile.

Edit distance is chosen as our similarity measure because (*i*) fingerprint instances vary as a result of network dynamics and webpage updates, whose effects are shown as insertion, deletion or substitution in the sequences, (*ii*) it considers the order information, which differs from Jaccard's coefficient that treats the feature values as a set, and (*iii*) the length information of fingerprint sequences are encapsulated for evaluation.

Example. Given two fingerprints $F_A = (A_{req}, A_{res})$ and $F_B = (B_{req}, B_{res})$, where $A_{req} = \langle 436, 516, 500, 532\rangle$, $A_{res} = \langle 548, 628, 1348, 1188, 596, 372, 980\rangle$, $B_{req} = \langle 436, 516, 500, 253, 500\rangle$ and $B_{res} = \langle 548, 628, 1348, 1188, 436, 412, 1268\rangle$. The normalized edit distance between the request sequences A_{req} and B_{req} is 2/5. The distance between the responses A_{res} and B_{res} is 3/7. Hence the similarity between F_A and F_B is 0.59 when $\alpha = 0.6$.

4 Website Fingerprinting under Traffic Morphing

Traffic morphing [1] aims to provide a bandwidth efficient countermeasure to website fingerprinting. Unigram morphing changes the distribution of single packet size, while bigram morphing changes the distribution of two consecutive packet sizes, to make the traffic appear as if from some other website. The problem of finding the morphing matrix to transform packet size distributions is formulated as an optimization problem, in which the goal is to minimize the bandwidth overhead, and the constraints are the source and target packet size distributions.

An important objective of traffic morphing is bandwidth efficiency. However, if traffic morphing is extended to n-gram ($n \geq 2$) to consider more packet ordering information, it becomes increasingly bandwidth inefficient. The reason is that higher-gram morphing has to concurrently satisfy all the lower-gram distributions, so there are very limited choices for the size of the n-th packet once the previous $n-1$ packet sizes are fixed. Source packet sizes have to map to the target distribution even though the size differences are large. For some value of n, its bandwidth efficiency drops below packet padding schemes.

4.1 Fingerprint Differentiation

Our scheme can differentiate website fingerprints under traffic morphing [1]. For the ease of discussion, let us call the source website W_S, and its morphed traffic S'. Website W_S mimics a target website W_T. The traffic of W_T is T. From S', we want to identify W_S.

We differentiate W_S and W_T fingerprints by packet ordering. Assume that we know website W_S morphs its traffic, while website W_T does not. A large edit distance between T and test stream S' indicates that S' is not from W_T, because

the fingerprints of W_T should be fairly consistent without morphing. Hence, test stream S' is from website W_S.

If there are k websites imitating the same target W_T, source W_S is likely uniquely identifiable. The reason roots at the morphing constraint to minimize bandwidth overhead. A morphing matrix maps each source packet to some minimally different sizes in the target distribution. Because of the correlation in packet sizes before and after morphing, and the consistency in the ordering of unmorphed packets, morphed traffic also have reasonably consistent order. It leaves us a loophole to differentiate among the k websites.

In evaluations, we examine the identifiability of k morphed websites amid other websites not related by morphing. We preprocess to narrow down the candidate websites by packet size distributions, e.g. using $L1$ distance measurement.[2] On one hand, the preprocessing safeguards that the morphing websites are not filtered prematurely. Morphed streams tend to have larger variations in packet ordering, because a source packet can map to a few target sizes and the choice is probabilistic for each morphing instance. On the other hand, preprocessing eliminates noise websites that the test stream may incidentally share similar size sequence with, but are dissimilar in size distributions.

Since we evaluate the similarity of fingerprints based on the whole sequence of feature values, our scheme can identify a morphed traffic, as long as morphing does not handle the distribution of n-gram for n close to the sequence length.

5 Evaluation

We present the performance analysis of our scheme in this section. Experiment settings and data collection are described in Sect. 5.1. We evaluate the fingerprinting scheme in three aspects: identification accuracies (Sect. 5.2), robustness to traffic morphing (Sect. 5.3) and consistency of fingerprints (Sect. 5.4).

5.1 Experiment Setup and Data Collection

We prepared three datasets on OpenVPN for evaluation, namely, *static50, dynamic150*, and *OpenVPN1000*. The dataset *static50* contains trace data of 50 websites that are rarely updated (less than once in a few months); the dataset *dynamic150* contains traces of 150 websites that are frequently update (daily); and the dataset *OpenVPN1000* contains traces of 1000 popular websites.

To demonstrate that our scheme also works on SSH, we use the publicly available dataset *OpenSSH2000* [2], which contains traces of 2000 most visited websites accessed through OpenSSH, captured once every 6 hours for 2 months. The trace captures only packet headers in the first 6 seconds of a webpage download.

[2] $L1$ distance between two website fingerprints is computed as the sum of absolute differences in their packet size distributions.

Table 1. Fingerprint identification accuracy for various datasets

Dataset	Static50	Dynamic150	OpenVPN1000
Accuracy	99%	97%	97%

We prepare the OpenVPN datasets by visiting the websites[3] and capturing the packet traces daily for 3 months. The traces are captured using TCPDump [19] between the client and the VPN server. We use Firefox as the browser, HTTP pipelining is enabled and its default value is 4 (at most 4 HTTP requests can be served concurrently). Cache is cleared after each access. Flash player is installed. Each web access is monitored for 30 seconds, to allow operations of object requests and responses to fully complete.

The topology, size and content of high level cache is not within our control for evaluation. In our experiments, the profiled websites are accessed regularly, so we expect most objects are consistently cached close to the VPN or SSH server.

We use Liberatore and Levine's scheme [2] for performance comparison, and refer to it as the *reference scheme*. Our scheme that performs preprocessing is referred to as the *improved scheme*, otherwise it is called the *basic scheme* or simply our scheme. For each website, 15 traces are used to generate the fingerprint profile and 10 other traces are used for testing. Analyzing from the trace data and protocol specifications, we use 300 and 1450 as thresholds on packet sizes to upper bound control packets and to lower bound MTU packets respectively. Thus the fingerprint sequence of object requests contains packet sizes that are larger than 300 bytes in the uploading direction, and the fingerprint sequence of non-MTU responses consists of packet sizes between 300 to 1450 bytes in the downloading direction. These settings are used in our experiments, unless otherwise specified.

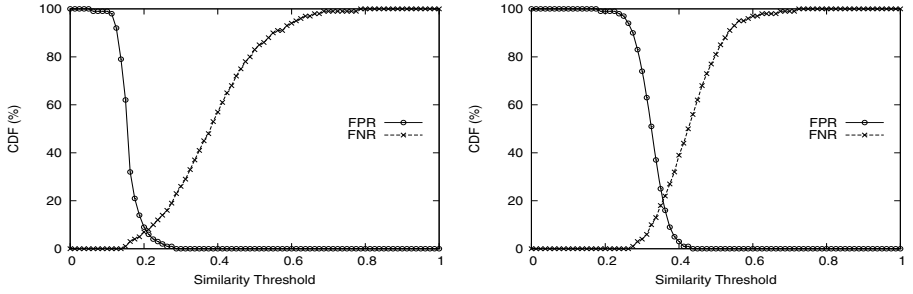
5.2 Fingerprint Identification Accuracy

Accuracy of classification scenario. The fingerprint identification accuracies on the OpenVPN datasets *static50*, *dynamic150* and *OpenVPN1000* are shown in Table 1. Note that for all three datasets, the accuracy is close to 100%.

To demonstrate that our scheme works well for identifying websites accessed in other tunnel, we tested it on the dataset *OpenSSH2000*, with parameters fully complying with the settings in [2]. The accuracy of our scheme is 81%, outperforming the reference scheme by 11%. The improvement is attained as our scheme differentiates websites that have similar sets of packet sizes but different packet ordering.

Accuracy of detection scenario. It is unknown in this scenario whether a test stream is from any profiled websites. Thus a test stream not from the profiled websites may be identified wrongly as from one of the websites in the profile

[3] We are fingerprinting a website by its homepage. If the internal pages of a website are also profiled, we can use the fingerprints of individual pages and their linkage information to identify more accurately and confidently the website being surfed.



(a) Our scheme (b) Reference scheme

Fig. 2. False positive and false negative rates with respect to similarity thresholds

(false positive); and a test from the profiled websites may be identified wrongly as not from the profiled websites (false negative). We evaluate the fingerprint identification accuracy in terms of false positive rate (FPR) and false negative rate (FNR) for the detection scenario.

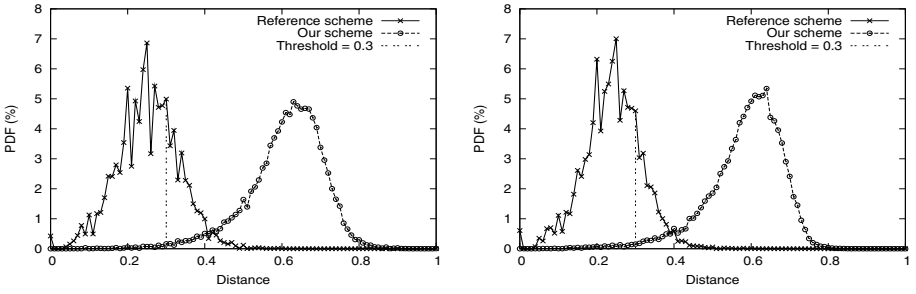
From the dataset *OpenVPN1000*, we randomly pick 200 websites to profile. Traces from both these 200 websites and the remaining 800 are used to test the fingerprint identification. The accuracy of our scheme with respect to different similarity thresholds is shown in Fig. 2a, and the performance of the reference scheme is shown in Fig. 2b. The equal error rate (EER) of our scheme occurs when the similarity threshold is 0.21, at which both FPR and FNR are 7%; while for the reference scheme, at the similarity threshold of 0.36, EER occurs to be 20%, almost three times of ours. If we minimize the total error rate, i.e. sum of FPR and FNR, the optimal similarity threshold for our method is 0.22, with only 6% FPR and 8% FNR; whereas for the reference scheme, the optimal similarity threshold is 0.37, at which its FPR and FNR are 9% and 27% respectively. The results show that our scheme is much stronger at differentiating websites.

5.3 Accuracy with Traffic Morphing

In this subsection, we empirically show that our scheme can differentiate fingerprints of websites morphed by packet sizes, and we verify that there is significant bandwidth overhead for n-gram ($n \geq 2$) morphing.

5.3.1 Fingerprint Differentiation under Traffic Morphing

Differentiating morphed traffic. We take 2000 websites as the mimicked targets, and generate the morphed traffic such that the packet size distributions are within $L1$ distance of 0.3 between a morphed traffic and its target. The same $L1$ distance threshold is used in the traffic morphing scheme [1]. For each target website, we generate 4 variants of the morphed distributions, and draw 5 instances from each variant. The distance in our scheme is measured using edit distance; while that in the reference scheme [2] is $1 - S$, where S is Jaccard's



(a) Unigram morphing (b) Bigram morphing

Fig. 3. Distance distribution of the morphed traffic and the mimicked target

coefficient. The distribution of distances between morphed traffic and their targets is shown in Fig. 3.

For unigram morphing as shown in Fig. 3a, compared to the reference scheme, our scheme shifts rightwards the range of distances from 95% in the interval [0.1, 0.4] to 95% in [0.4, 0.8]. The larger distance indicates that our scheme differentiates more clearly the morphed traffic and the target. The vertical bar at 0.3 indicates the distance threshold, and a test returning a value below the threshold means that the distance evaluation cannot separate the morphed instance and the mimicked target. For the reference scheme, 75% of the tests fail; while for our method, there is only 1.0% error cases, i.e., our scheme differentiates 99% of the morphing traffic. As shown in Fig. 3b, the experiment on bigram morphing yields similar results. For the reference scheme, 78% of the tests fail; while for our scheme, there is only 1.3% error cases. The experiment result shows that although traffic morphing is effective against the reference scheme, our scheme is highly resistant to it.

Identifying multiple websites morphed to the same target. We extend the evaluation to multiple websites morphing to the same target, and compare the distinguishability using our scheme, the reference scheme and random guess. We take 20 disjoint sets of k websites, where the first $k-1$ websites bigram morph towards the k-th website. Each morphing website generates 6 instances, 3 as learning samples and the rest 3 as test cases. We vary k from 2 to 6, and measure the identification accuracy of morphed traces in each set. The average identification accuracy of all sets at each k is presented in Fig. 4.

When our scheme differentiates between $k = 2$ websites, i.e. the source and target of morphing, it yields a high identification accuracy of 85%. As k increases, more websites share the same morphing target, the accuracy gradually decreases. When $k = 6$ candidates, which means five source websites morph to one target, our scheme has an accuracy of 50%. In contrast, accuracy of the reference scheme is 29% when $k = 6$, while random guess gives a probability of 17%. The reference scheme performs not much better than random guess. It

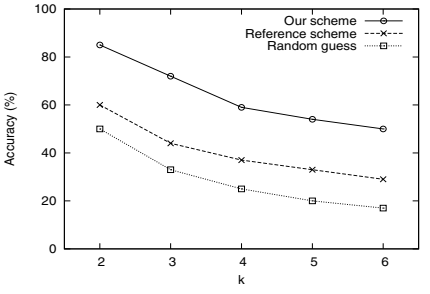


Fig. 4. Identification accuracy of k websites that morph to the same target

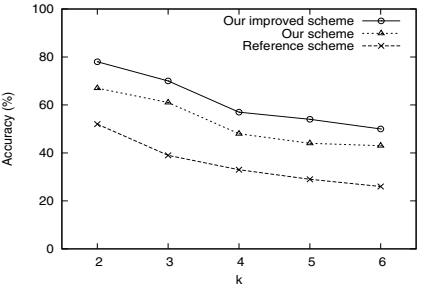


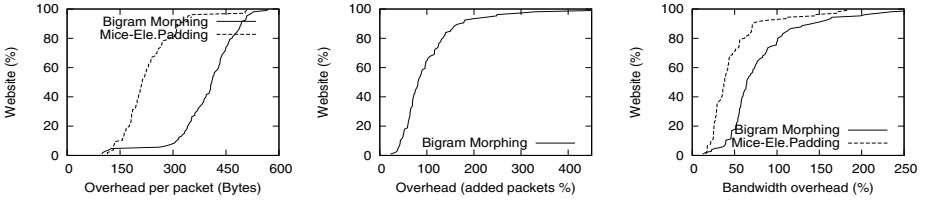
Fig. 5. Identification accuracy of k morphed websites among 2000 other websites

shows that the reference scheme cannot reliably distinguish websites that share similar packet size distributions. Our scheme is much stronger at differentiating among multiple morphing websites and their mimicked target.

Identifying morphed traffic mixed with other website profiles. Previous experiments examine the distinguishability within morphing sources and their target. Next we evaluate the identifiability of 20 sets of k morphing websites when they are mixed with 2000 other websites. Morphing websites are profiled based on their morphed traces. We set the $L1$ distance threshold to be 0.4 in screening websites by packet size distributions. The threshold is slightly larger than in morphing matrix computation to accommodate some incompliance between the computed and generated packet size distributions. The incompliance is caused by packet splitting which generates new packets that are not accounted for in the computation of morphing matrix. The identification accuracies are shown in Fig. 5.

Comparing Fig. 5 and Fig. 4, our improved scheme performs equally well when it identifies k morphing websites among a large set of unrelated websites and when it differentiates within the morphing websites. When k varies from 2 to 6, our improved scheme has an identification accuracy ranges from 78% to 50%, while the reference scheme has a corresponding range of 52% to 26%. The performance of our basic scheme in Fig. 5 degrades compared to in Fig. 4, because some morphed traces incidentally map to websites unrelated by morphing. However, it still outperforms the reference scheme. Our basic scheme correctly identifies 61% fingerprints of the morphing websites at $k = 3$, while the reference scheme identifies 39%.

We run our improved scheme on 2000 non-morphing websites. It gives the same identification accuracy of 81% as the basic scheme in this case. It is compatible to website fingerprinting at the absence of morphing, since non-morphing websites have HTTP streams that are consistent in both packet size distribution and ordering.



(a) Extra bytes per packet (b) Added packets (c) Bandwidth overhead

Fig. 6. Bandwidth overhead of bigram morphing and mice-elephant packet padding

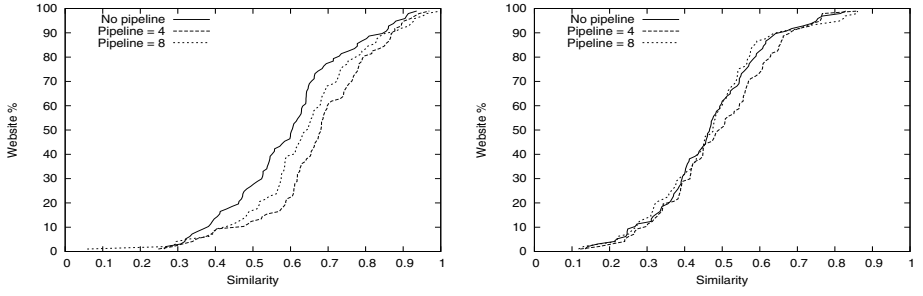
5.3.2 Bandwidth Overhead of n-Gram ($n \geq 2$) Morphing

N-gram ($n = 1, 2, 3$) morphing has been evaluated on several applications in [1]. Yet for website fingerprinting, its bandwidth overhead was presented only for unigram morphing ($n = 1$). The bandwidth overhead was compared to padding packets to the path MTU. In this paper, we evaluate the bandwidth overhead of bigram ($n = 2$) morphing. The overhead is compared to mice-elephant packet padding, which is another simple and effective padding scheme against website fingerprinting, but less constraining on the network bandwidth. We use traces of 100 websites from the *SSH2000* dataset. The i-th website is morphed to the $(i + 1)$-th website, where each HTTP stream is morphed 5 times. The average overhead is presented in Fig. 6.

Fig. 6a shows the average added bytes per packet by morphing and mice-elephant. Using mice-elephant, 80% websites have 150 to 300 bytes of overhead per packet; but using traffic morphing, about 90% websites need to send 300 to 500 extra bytes for each packet, which is much higher than mice-elephant. Fig. 6b shows the percentage of added packets caused by packet splitting during morphing. Transmission of extra packets incurs overhead in bandwidth and delay. For some website, more than 4 times of packets are added. Fig. 6c shows the overall bandwidth overhead. Using bigram morphing, 20% websites have an overhead that exceeds 100% and can be 300% of the total source packet sizes. While using mice-elephant packet padding, 75% websites have the bandwidth overhead less than 50% of the total source packet sizes, and only 5% websites have bandwidth overhead more than the total source packet sizes. As n increases, the bandwidth overhead of n-gram morphing also increases. In summary, our experiment shows that n-gram morphing is less bandwidth efficient than mice-elephant packet padding even when $n = 2$.

5.4 Consistency of Fingerprints

Fingerprint consistency can be influenced by browser pipeline configurations, web content updates and network dynamics which cause packet loss, reordering and retransmission. Note that our fingerprints are not affected by the randomness in the amount of control packets, as we have excluded control packets from website fingerprints. In practice, packet loss, reordering and retransmission are



(a) Sequence of HTTP requests (b) Sequence of non-MTU responses

Fig. 7. Effect of pipelining on the fingerprint sequences

not severe. In this subsection, we empirically show that the effect of pipelining is not serious on our fingerprints, and our fingerprints need not be updated frequently for most websites.

Effect of HTTP pipelining. To illustrate the effect of HTTP pipelining on the fingerprint sequences, we vary the pipeline degree from 1 (non-pipelined) to 8 (the maximum supported by Firefox). We capture OpenVPN traces of 100 websites. For each website, fingerprints from 15 traces with varying pipeline degrees are kept as profile, and 15 test streams of pipeline degrees 1, 4 and 8 are compared with their own website profile.

Fig. 7 shows the fingerprint sequences of a website have high similarities. 70% websites have similarity in the object request sequences higher than 0.5, and 50% of the non-MTU response sequences have similarities larger than 0.5. The figure also shows that the distributions of a website's fingerprint similarities at different pipeline degrees are similar. HTTP pipelining does not drastically affect our fingerprint sequences.

Effect of website update. To evaluate how well the profiles represent websites over time, we measure the fingerprint identification accuracy weeks after constructing the profiles. The evaluation helps to decide the update frequency of website profiles. We perform an experiment on the datasets *OpenVPN1000* and *dynamic150* to study the impact of (frequent) content updates on the fingerprint identification. For each website, we randomly pick 5 traces captured within week 0 to generate the website profile, then the traces of 4 randomly chosen days in each of week 1, 2, 3, 4 and 8, are tested against the website profiles.

The results are shown in Fig. 8. For dataset *OpenVPN1000*, the identification accuracy gradually decreases over time, which remains high at 94% a month later and becomes 88% after two months. Profiles of most websites need not be regenerated after 2 months. Only 6% most frequently updated websites need to update their fingerprint profiles in the first month, and another 6% in the second month. For the dataset *dynamic150*, the decrease in identification accuracy

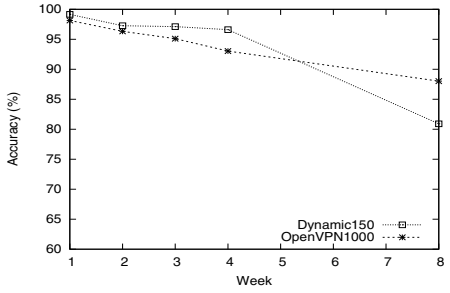


Fig. 8. Effect of time on accuracy

from week 4 to week 8 is obvious, because their website contents are frequently updated, and the changes accumulate over time. Nevertheless, its fingerprint identification accuracy is 96% one month later, and 81% two months later.

6 Countermeasures

We propose some countermeasures that randomizes the size and timing channels. These countermeasures incurs almost zero bandwidth overhead. (*i*) *Randomizing the object order.* We can use a browser plug-in to muddle the object request order. With the plug-in, browser buffers x HTTP requests, and sends them in a random order, where x can be a random number dynamically generated at each web access. (*ii*) *Randomizing the packet sizes.* To make the size and order of packets untraceable, browser and server can generate a sequence of random numbers for each access, and buffers the data to send in packets of the random sizes. Random delay can be added to conceal which packet sizes have been touched up in the HTTP stream. Although data buffering causes delay, the countermeasures significantly increase the difficulty of website fingerprinting.

7 Conclusion

In this paper, we developed a robust website fingerprinting scheme over low latency encrypted tunnels. We make use of the seemingly noisy packet ordering information rather than just the distribution of packet sizes as in previous work. The ordering of a selection of packets is consistent due to the behaviorial characteristics of protocols and browsers, and such information is preserved and exposed regardless of proxy and encryption. Our scheme identifies websites with high accuracy on both SSH and SSL tunnels, even for websites with dynamic contents. Furthermore, our scheme is designed to withstand traffic morphing [1]. Traffic morphing cannot handle packet ordering while satisfying low bandwidth overhead. Its bandwidth efficiency is worse than mice-elephant packet padding, even in bigram morphing. Our scheme is able to identify websites from morphed traffic with significantly higher accuracy than the previous scheme. In future work, we would analyze performances of the countermeasures in both their effectiveness and overheads.

References

1. Wright, C.V., Coull, S.E., Monrose, F.: Traffic morphing: An Efficient Defense against Statistical Traffic Analysis. In: 16th NDSS (2009)
2. Liberatore, M., Levine, B.N.: Inferring the Source of Encrypted HTTP connections. In: 13th ACM CCS, pp. 255–263 (2006)
3. Hintz, A.: Fingerprinting Websites using Traffic Analysis. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 229–233. Springer, Heidelberg (2003)
4. Sun, Q., Simon, D.R., Wang, Y.M., Russell, W., Padmanabhan, V.N., Qiu, L.: Statistical Identification of Encrypted Web Browsing Traffic. In: IEEE S&P 2002, pp. 19–30 (2002)
5. Song, D.X., Wagner, D., Tian, X.: Timing Analysis of Keystrokes and Timing Attacks on SSH. In: 10th USENIX Security Symposium (2001)
6. Saponas, T.S., Lester, J., Hartung, C., Agarwal, S.: Devices that Tell on You: Privacy Trends in Consumer Ubiquitous Computing. In: 16th USENIX Security Symposium, pp. 55–70 (2007)
7. Bonfiglio, D., Mellia, M., Meo, M., Rossi, D., Tofanelli, P.: Revealing Skype Traffic: When Randomness Plays with You. ACM SIGCOMM Computer Communication Review 37(4), 37–48 (2007)
8. Wright, C.V., Monrose, F., Masson, G.M.: On Inferring Application Protocol Behaviors in Encrypted Network Traffic. Journal of Machine Learning Research 7, 2745–2769 (2006)
9. Bissias, G.D., Liberatore, M., Jensen, D., Levine, B.N.: Privacy Vulnerabilities in Encrypted HTTP Streams. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 1–11. Springer, Heidelberg (2006)
10. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Journal of Soviet Physics Doklady 10(8), 707–710 (1966)
11. Norvig, P.: How to Write a Spelling Corrector, http://www.norvig.com
12. Wilson, C.: Who checks the spell-checkers, http://www.slate.com/id/2206973/pagenum/all/
13. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
14. Needleman, S.B., Wunsch, C.D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology 48, 443–453 (1970)
15. Navarro, G.: A Guided Tour to Approximate String Matching. Journal of ACM Computing Surveys 33(1), 31–88 (2001)
16. Gooskens, C., Heeringa, W.: Perceptive Evaluation of Levenshtein Dialect Distance Measurements using Norwegian Dialect Data. Journal of Language Variation and Change 16(3), 189–207 (2004)
17. Felten, E.W., Schneider, M.A.: Timing Attacks on Web Privacy. In: 7th ACM CCS (2000)
18. Wagner, R.A., Fischer, M.J.: The String-to-String Correction Problem. J. ACM 21(1), 168–173 (1974)
19. TCPDump, http://www.tcpdump.org

Web Browser History Detection as a Real-World Privacy Threat

Artur Janc[1] and Lukasz Olejnik[2]

artur@lingro.com,
lukasz.olejnik@man.poznan.pl

Abstract. Web browser history detection using CSS *visited* styles has long been dismissed as an issue of marginal impact. However, due to recent changes in Web usage patterns, coupled with browser performance improvements, the long-standing issue has now become a significant threat to the privacy of Internet users.

In this paper we analyze the impact of CSS-based history detection and demonstrate the feasibility of conducting practical attacks with minimal resources. We analyze Web browser behavior and detectability of content loaded via standard protocols and with various HTTP response codes. We develop an algorithm for efficient examination of large link sets and evaluate its performance in modern browsers. Compared to existing methods our approach is up to 6 times faster, and is able to detect up to 30,000 visited links per second.

We present a novel Web application capable of effectively detecting clients' browsing histories and discuss real-world results obtained from 271,576 Internet users. Our results indicate that at least 76% of Internet users are vulnerable to history detection, including over 94% of Google Chrome users; for a test of most popular Internet websites we were able to detect, on average, 62.6 (median 22) visited locations per client. We also demonstrate the potential to profile users based on social news stories they visited, and to detect private data such as zipcodes or search queries typed into online forms.

1 Introduction

Web browsers function as generic platforms for application delivery and provide various usability enhancements with implications for user privacy. One of the earliest such usability improvements was the ability to style links to Web pages visited by the user differently from unvisited links, introduced by the original version of the Cascading Style Sheets standard [1] and quickly adopted by all major Web browsers. This mechanism was soon demonstrated to allow malicious Web authors to detect links a client has visited and report them to the attacker [2].

Since then, a body of academic work has been published on this topic, describing history detection methods [3] and discussing the potential to detect visited websites to aid in phishing [4]. Several countermeasures against such attacks

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 215–231, 2010.

were proposed, including client-side approaches through browser extensions [5] and server-side solutions on a per-application basis [6], but such methods have not been adopted by browser vendors or Web application developers. Simultaneously, several demonstration sites have been created to show the ability to detect known popular websites, including Web 2.0 applications [7].

More recently, CSS-based history detection started to become applied as a powerful component of privacy research, including work to determine the amount of user-specific information obtainable by ad networks [8] and as part of a scheme for deanonymizing social network users [9]. However, there has been a notable lack of work examining several crucial aspects of history detection, including the types of browser-supported protocols and resource types which can be detected, performance considerations, and the number of users affected by such attacks.

In this paper, we provide a detailed examination of CSS-based history detection techniques and their impact on the privacy of Internet users. We provide an overview of existing work, and discuss basic cross-browser implementations of history detection using JavaScript as well as a CSS-only technique. We evaluate the detectability of resources based on the browser-supported protocols used to retrieve them, analyze the effect of loading content in frames and iframes, as well as review the impact of HTTP redirects and other codes.

We demonstrate an optimized algorithm for detecting visited links and its JavaScript implementation. We provide detailed performance measurements of our technique and compare it to existing approaches. Our approach is up to 6 times faster than known methods, and allows for the examination of up to 30,000 links per second on modern hardware. We also provide the first performance analysis of the CSS-only history detection technique, demonstrating its value as an efficient, though often neglected, alternative to the scripting approach.

Based on our work on a real-world testing system [10], we provide an overview of the design of an efficient history detection application capable of providing categorized test of various website classes, and realizable with minimal resources. We discuss approaches for the selection of links to be detected, and outline our implemented solution based on *primary* links (as site entry points), *secondary* resources, and *enumeration elements*.

Finally, we analyze history detection results obtained from 271,576 users. We demonstrate that a large majority (76.1%) of Internet users are vulnerable to history detection attacks, including over 82% of Safari users and 94% of Google Chrome users. We analyze the average number of primary and secondary links found in a test of popular Internet locations; for vulnerable users our test found an average of 62.6 visited links (22 median). We also provide an overview of detected outgoing links from social news sites and discuss the potential of our system to gather especially privacy-sensitive data.

Our results indicate that properly prepared history detection attacks have significant malicious potential and can be directed against the vast majority of Internet users.

2 Background

The CSS *visited* pseudoclass has been applied to links visited by client browsers since the introduction of the CSS1 standard in 1996 [1]. The feature of applying different styles to "known" links quickly became accepted by users and was recommended by usability experts [11].

The ability to use the *visited* pseudoclass for detecting Web users' browsing history was first reported to browser vendors as early as the year 2000 [2,12]. Since then, the technique has been independently rediscovered and disclosed several times [4], and has become widely known among Web browser developers and the security and Web standards communities. In fact, Section 5.11.2 of the CSS 2.1 standard [13], a W3C recommendation since 1998, discusses the potential for history detection using the *visited* pseudoclass, and explicitly allows conforming User Agents to omit this functionality for privacy reasons, without jeopardizing their compliance with the standard.

While initial discussions of CSS-based history detection were mostly conducted in on-line forums, the issue was also disclosed to the academic community and discussed in the work of Felten et al. in conjuction with cache-based history sniffing [3].

As a response, Jakobsson and Stamm discussed potential methods for implementing server-side per-application protection measures [14]; such techniques would have to be implemented by every Web-based application and are thus an extremely unlikely solution to the problem. A viable client-side solution was a proposed modification to the algorithm for deciding which links are to be considered visited as described in [5] and implemented in the SafeHistory extension [15] for Mozilla Firefox. Unfortunately, no such protection measures were implemented for other Web browsers[1], and the SafeHistory plugin is not available for more recent Firefox versions.

Other academic work in the area included a scheme for introducing voluntary privacy-oriented restrictions to the application of history detection [17]. Two more recent directions were applications of history detection techniques to determine the amount of user-specific information obtainable by ad networks [8] and as part of a scheme for deanonymizing social network users [9].

CSS-based history detection was also discussed as a potential threat to Web users' privacy in several analyses of Web browser security [18,19].

Outside of the academic community, several demonstration sites were created to demonstrate specific aspects of browser history detection. Existing applications include a script to guess a visitor's gender by combining the list of detected domains with demographic information from [20], a visual collage of visited Web 2.0 websites [7], and an entertaining detector of adult websites [21]. However, all known proof of concept sites focus on a single application, and do not explore the full potential of history detection as a tool to determine user-specific private information.

[1] Since writing the original draft of this work, we have become aware of ongoing efforts to mitigate history detection attacks in the Gecko and WebKit rendering engines [16].

3 Analysis

In order to fully evaluate the implications of CSS-based history detection, it is necessary to understand how and when Web browsers apply *visited* styles to links. In this section we analyze various browser behaviors related to visited links, describe an efficient algorithm for link detection and evaluate its performance in several major browsers[2].

3.1 Basic Implementation

CSS-based history detection works by allowing an attacker to determine if a particular URL has been visited by a client's browser through applying CSS styles distinguishing between visited and unvisited links. The entire state of the client's history cannot be directly retrieved; to glean history information, an attacker must supply the client with a list of URLs to check and infer which links exist in the client's history by examining the *computed* CSS values on the client-side. As noted in [12], there are two basic techniques for performing such detection.

The CSS-only method shown in Figure 1 allows an attacker's server to learn which URLs victim's browser considers to be visited by issuing HTTP requests for background images on elements linking to visited URLs. A similar, but less known technique is to use the *link* CSS pseudoclass, which only applies if the link specified as the element's href attribute has not been visited; the techniques are complementary.

```
<style>
#foo:visited {background: url(/?yes-foo);}
#bar:link {background: url(/?no-bar);}
</style>
<a id="foo" href="http://foo.org"></a>
<a id="bar" href="http://bar.biz"></a>
```

Fig. 1. Basic CSS Implementation

A similar technique can be performed on the client side with JavaScript, by dynamically querying the style of a link (<a>) element to detect if a particular CSS style has been applied, shown in Figure 2. Any valid CSS property can be used to differentiate between visited and unvisited links. The scripting approach allows for more flexibility on part of the attacker, as it enables fine-grained control over the execution of the hijacking code (e.g. allows resource-intensive

[2] Browser behavior and performance results were gathered with Internet Explorer 8.0, Mozilla Firefox 3.6, Safari 4, Chrome 4, and Opera 10.5 on Windows 7 using an Intel Core 2 Quad Q8200 CPU with 6GB of RAM.

tests to be run after a period of user inactivity) and can be easily obfuscated to avoid detection by inspecting the HTML source. It can also be modified to utilize less network resources than the CSS-only method, as discussed in Section 3.3. Both techniques can be executed transparently to the user and do not require any interaction other than navigating to a Web page.

```
<script>
var r1 = 'a {color:green;}';
var r2 = 'a:visited {color:red;}';

document.styleSheets[0].insertRule(r1, 0);
document.styleSheets[0].insertRule(r2, 1);

var a_el = document.createElement('a');
a_el.href = "http://foo.org";

var a_style = document.defaultView.getComputedStyle(a_el, "");

if (a_style.getPropertyValue("color") == 'red')
        // link was visited
</script>
```

Fig. 2. Basic JavaScript Implementation

3.2 Resource Detectability

The CSS history detection technique has historically been applied almost exclusively to detect domain-level resources (such as `http://example.org`), retrieved using the HTTP protocol. However, Web browsers apply the visited style to other kinds of links, including sub-domain resources such as images, stylesheets, scripts and URLs with local anchors, if they were visited directly by the user. In general, and with few exceptions, there exists a close correspondence between the URLs which appeared in the browser's *address bar* and those the browser considers to be visited. Thus, visited URLs within protocols other than HTTP, including `https`, `ftp`, and `file` can also be queried in all tested browsers, with the exception of Chrome which does not apply *visited* styles to `file://` links.

Because of the address bar rule outlined above, parameters in forms submitted with the HTTP POST request method cannot be detected, whereas parameters from forms submitted using HTTP GET are susceptible to detection. The URLs for resources downloaded indirectly, such as images embedded within an HTML document, are usually not marked as visited. However, one exception is the handling of frames and iframes in some browsers. A URL opened in a frame or iframe does not appear in the address bar, but the Firefox and Chrome browsers still consider it to be visited.

While all major browsers apply *visited* styles to valid resources (ones returning HTTP 200 status codes), variations exist for other response types. When encountering an HTTP redirect code (status 301 or 302) Firefox, Chrome and Opera mark both the redirecting URL and the new URL specified in the Location HTTP header as visited, whereas Safari saves only the original URL, and Internet Explorer exhibits seemingly random behavior. When performing a *meta redirect*, all browser except Internet Explorer consider both URLs to be visited; newer versions of Internet Explorer do not allow such redirects in the default configuration. When retrieving an invalid URL with a client or server error status (codes 4xx and 5xx), all browsers except Internet Explorer mark the link as visited. The handling of various types of HTTP responses is summarized in Table 1.

The ability to detect links visited by any user depends on the existence of those links in the browser's history store and is affected by history expiration policies. This default value for history preservation varies between browsers, with Firefox storing history for 90 days, Safari - 20 days, and IE - 20 days. Opera stores 1000 most recently visited URLs, whereas Chrome does not expire browsing history.

It is important to note that a potential adversary whose website is periodically visited by the user (or whose script is linked from such a site) can query the history state repeatedly on each visit, maintaining a server-side list of the user's detected links; such an approach would allow the attacker to aggregate browsing information, bypassing history expiration policies.

Table 1. Detectability for HTTP status codes and redirects

	IE	**Firefox**	**Safari**	**Chrome**	**Opera**
200	yes	yes	yes	yes	yes
301	random	both	original	both	both
302	random	both	original	both	both
meta redirect	n/a	both	both	both	both
404	no	yes	yes	yes	yes
500	no	yes	yes	yes	yes

3.3 Performance

CSS-based history detection is a viable technique for various privacy-related attacks because of its simplicity and the ability to quickly check for a large number of visited resources. In order to fully understand the implications of CSS-based history detection attacks, it is thus crucial to learn about its performance characteristics using optimized scripts for each browsing environment.

Optimizing JavaScript Detection. To date, little effort has been put into the analysis of efficient implementations of JavaScript-based history detection. Several existing implementations use DOM `a` elements in a static HTML document to hold URLs which are later inspected to determine if the CSS visited rule

applied to the corresponding URL, an approach significantly slower than a fully-dynamic technique. Additionally, due to browser inconsistencies in their internal representations of computed CSS values (e.g. the color red can be internally represented as "red", "#ff0000", "#f00", or "rgb(255, 0, 0)") most detection scripts try to achieve interoperability by checking for a match among multiple of the listed values, even if the client's browser consistently uses one representation. Another difference affecting only certain browsers is that an `a` element must be appended to an existing descendant of the document node in order for the style to be recomputed, increasing script execution time.

For our detection code, we took the approach of creating an optimized technique for each major browser and falling back to a slower general detection method for all other browsers. We then compared the execution time of the optimized test with the general method for each major browser. The differences in execution times are shown in Figure 3.

For each browser the implementation varies slightly, depending on the way CSS properties are represented internally and the available DOM mechanisms to detect element styles. The general detection algorithm for lists of links is as follows:

1. Initialize CSS styles and store URLs to check in a JavaScript array.
2. Detect browser version and choose appropriate detection function.
3. Invoke chosen detection function on URL array.
   - Create <**a**> element and other required elements.
   - For each URL in array:
     - Set <**a**> element href attribute to URL.
     - (for some browsers) Append element to DOM or recompute styles.
     - If computed style matches visited style, add URL to "visited" array.
4. Send contents of visited array to server or store on the client-side.

Our approach has the advantage of avoiding a function call for each check, reusing DOM elements where possible, and is more amenable to optimization by JavaScript engines due to a tight inner loop. Compared to a naive detection approach using static <**a**> elements in the HTML source and less-optimized style matching, our technique is between 1.8 and 6 times faster depending on the browser.

CSS Performance. The CSS-only detection technique is a valuable alternative to the scripting approach, as it allows to test clients with JavaScript disabled or ones with security-enhancing plug-ins such as NoScript. Our results, provided in Figure 4, show that CSS-based detection can perform on par with the scripting approach, allowing an attacker to test for over 25,000 visited links per second for small data sets of 50,000 links and fewer. An important drawback, however, is that CSS-based detection requires <**a**> elements with appropriate *href* attributes to be included in the static HTML source, increasing the page size and required bandwidth. Additionally, for larger link sets (HTML pages with over

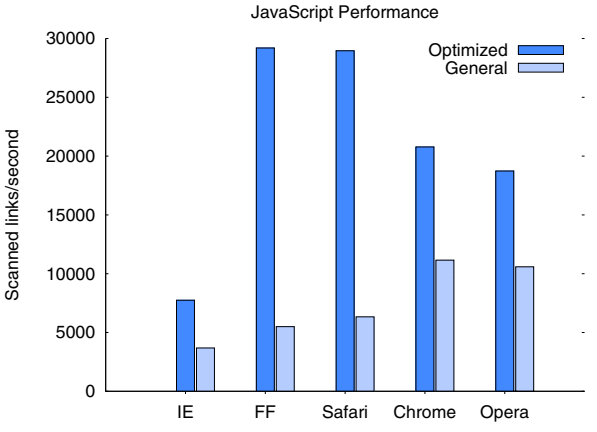


Fig. 3. JavaScript detection performance for different browsers. The general approach can be clearly seen as much slower.

50,000 elements), detection performance (and overall browser performance) decreases quickly with the increasing number of DOM elements included in the page[3].

Network Considerations. While client-side detection efficiency is of the most importance, we observe that the overall goal of detecting visited URLs in the client's browsing history can require significant network resources. Since many browsers on modern hardware are able to check tens of thousands of links per second, the bandwidth necessary to sustain constant checking speed becomes non-trivial.

In our test data set, the median URL lengths are 24 bytes for primary links (hostnames), and 60 bytes for secondary links (resources within each site). The overhead of including a URL in a JavaScript script in our implementation was 3 bytes (for quoting and separating array elements). For CSS, the average size overhead was 80 bytes due to the necessity of adding HTML markup and static CSS styles. In our tests, transmitting 30,000 thousand URLs required approximately 1650 kB (170 kB with gzip compression) for JavaScript, and 3552 kB (337kB with gzip compression) for CSS tests.

For an average broadband user, available bandwidth could potentially be a limiting factor, especially for owners of modern systems which can execute the detection code faster. To decrease the required bandwidth, transmitted links can omit common patterns (e.g. `http://` or `http://www.`); enumerating resources within a single domain can also significantly reduce the required network bandwidth by only transmitting the variable URL component.

[3] Test pages with more than 50 thousand elements caused errors and did not load in Internet Explorer.

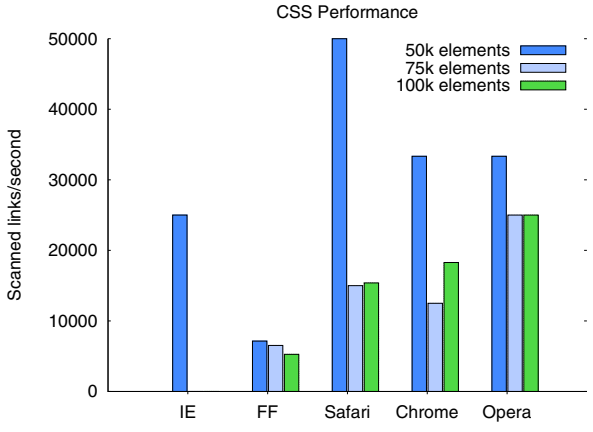


Fig. 4. CSS detection performance. Due to the limitations of Internet Explorer, only data for 50 thousand links is shown.

4 Methodology

A core goal of our work was to build a functional system to demonstrate the possible privacy risks associated with browser history detection, including the development of categorized tests detecting various classes of online resources. Our testing system was designed to maximize the number of URLs retrieved from each visitor's history and to present visitors with a visual representation of what can be inferred about their browsing habits.

4.1 System Overview

Our testing application was divided into multiple test categories, each of which contained several history detection tests. Test categories included:

- General tests of popular websites selected from Web rankings [22],
- On-line news and social news sites along with posted story links,
- A final category of miscellaneous tests (including a zipcode detection test and a check of performed search engine queries).

The default test which executed when a user visited the site homepage was the "top5k" test, checking for 6,417 most popular Internet locations. Selected tests are listed in Table 2.

When a user visited a test page, she was presented with a short test description, along with a list of *primary* links to check. When the page loaded, the browser automatically performed checks of all links in the list, continuously updating a progress bar to inform the user about the test status. When all links were checked, the browser submitted the results to the server using an AJAX request, and received in response the thumbnail images and descriptions for all

websites for which primary links were found, as well as a list of *secondary* links for each such website. The browser then checked all links in the secondary list and submitted the results to the server. The server's final reply contained an overview of the data found in the user's history, along with a detailed list of all primary and secondary links found.

For some tests, the set of secondary links was accompanied by a list of *enumeration elements* such as usernames on a visited social news site (Digg, Reddit or Slashdot), popular search engine queries for the search query test, or US zipcodes for the zip code detector test. Enumeration elements were appended to one or more base URLs supplied by the server (of the form `http://reddit.com/user/username`, with *username* as an enumeration element) and were checked similarly to primary and secondary links. This mechanism added a semantic component to the test by informing the server about the type of the link found in the user's history (e.g. username or search term), as contrasted with a "generic" link. It also helped the system conserve network bandwidth, by omitting common URL prefixes for similar resources.

If a user visited any test page with JavaScript disabled, the server automatically recognized that fact and redirected the client to a separate test page which utilized the CSS-only method described in Section 3.1. The CSS-only test required more network resources, but tested for the same primary and secondary links as the regular test and presented results in the same manner. An overview of differences between results gathered from clients with and without JavaScript is provided in Table 3.

4.2 Link Selection

The selection of URLs to check for in each client's history is of paramount importance in any project utilizing CSS-based history detection, as it determines how much browsing data can be gathered. However, if too much data is transferred to the user, both the page load and test run times might increase to the point that the user will leave the page without completing the test. Large data sets also limit the number of concurrent client a testing server can support due to server-side network and computational limitations. In our system we tackled this problem by both splitting tests into domain-specific categories, and dividing our tests into two phases for checking *primary* and *secondary* links.

Primary Links. For each test we gathered primary links representing domains of websites which contained resources of interest for the particular test. For the general test category we used popular Web analytics services including Alexa [22], Quantcast [20] and Bloglines [23] to identify the most popular Internet locations.

We retrieved the HTML source for each primary link and if any HTTP redirects occured, we kept track of the new URLs and added them as alternate links for each URL (for example if `http://example.org` redirected to `http://example.org/home.asp` both URLs would be stored). We also performed basic

unifications if two primary links pointed to slightly different domains but appeared to be the same entity (such as `http://example.org` and `http://www.example.org`).

A total of 72,134 primary links were added to our system as shown in Table 2. To each primary link we added metadata, including the website title and a human-readable comment describing the nature of the site, if available. Primary links served as a starting point for resource detection in each test—if a primary link (or one of its alternate forms) was detected in the client's history, secondary links associated with that primary link were sent to the client and checked.

Table 2. Number of links to be scanned per test is shown

	Primary links	Secondary links
top5k	6417	1416709
top20k	23797	4054165
All	72134	8598055

Secondary Links. Browser history detection has the potential for detecting a variety of Web-based resources in addition to just the hostname or domain name of a service. In our tests, for each primary link we gathered a large number of *secondary* links for resources (subpages, forms, directly accessible images, etc.) within the domain represented by the primary link. The resources were gathered using several techniques to maximize the coverage of the most popular resources within each site:

1. Search engine results. We utilized the Yahoo! BOSS [24] search engine API and queried for resources within the domain of the primary link, taking advantage of the fact that search engine results are sorted by relevance so that the top results returned correspond to the most often visited pages. For most primary links, we requested 100 results, but for the most popular Internet locations (sites in the Alexa 500 list) we retrieved 500 results.
2. HTML inspection. We retrieved the HTML source for each primary link and made a list of absolute links to resources within the domain of the primary link. The number of secondary links gathered using this method varied depending on the structure of each site.
3. Automatic generation. For some websites with known URL schemes we generated secondary links from list pages containing article or website section names; this behavior allowed us to quickly generate links for websites such as Craigslist and Wikileaks.

We then aggregated the secondary links retrieved with each method, removing duplicates or dubious URLs (including ones with unique identifiers which would be unlikely to be found in any user's history) and added metadata such as link descriptions where available.

For news site tests we also gathered links from the RSS feeds of 80 most popular news sites, updated every two hours[4]. Each RSS feed was tied to a primary link (e.g. the `http://rss.cnn.com/rss/cnn_topstories.rss` was associated with the `http://cnn.com` primary link). Due to the high volume of links in some RSS feeds, several news sites had tens of thousands of secondary links.

Resource Enumeration. In addition to secondary links, some primary links were also associated with *enumeration elements*, corresponding to site-specific resources which might exist in the browser's cache, such as usernames on social news sites, popular search engine queries, or zipcodes typed into online forms. To demonstrate the possibility of deanonymizing users of social news sites we gathered lists of active users on those sites by screen scraping for usernames of link submitters and comment authors. Enumeration elements were also useful for tests where similar resources might be visited by the user on several sites – in our search engine query test, the URLs corresponding to searches for popular phrases were checked on several major search engines without needing to transmit individual links multiple times.

4.3 Processing Results

For each visiting client, our testing system recorded information about the links found in the client's history, as well as metadata including test type, time of execution, the User Agent header, and whether the client had JavaScript enabled. Detected primary links were logged immediately after the client submitted first stage results and queried the server for secondary links. After all secondary links were checked, the second stage of detection data was submitted to the test server. All detected information was immediately displayed to the user.

For large-scale history detection systems, the amount of gathered data might be affected by server-side resource limits such as bandwidth and processing power. Our application was deployed on a single virtual server in a shared VM environment [25] using a basic $20/month plan, which affected the amount of information we could gather and process. Organizations with more resources would be able to perform more extensive history detection tests, posing a more serious threat to user privacy.

5 Results

The testing application based on this work was put into operation in early September 2009 and is currently available at [10]. Results analyzed here span the period of September 2009 to February 2010 and encompass data gathered from 271,576 users who executed a total of 703,895 tests. The default top5k

[4] Due to high interest in our testing application and associated resource constraints, we were forced to disable automatic updating of RSS feeds for parts of the duration of our experiment.

test, checking for 6,417 most popular Internet locations and 1,416,709 secondary URLs within those properties was executed by 243,068 users[5].

5.1 General Results

To assess the overall impact of CSS-based history detection, it is important to determine the number of users whose browser configuration makes them vulnerable to the attack. Table 3 summarizes the number of users for whom the top5k test found at least one link, and who are therefore vulnerable. We found that we could inspect browsing history in the vast majority of cases (76.1% connecting clients), indicating that millions of Internet users are at risk. A somewhat smaller number of users with found results for the All test might be attributed to the fact that users who recently cleared their browsing history or used private browsing modes executed the most extensive test to determine if they are at any risk.

Table 3. Aggregate results for popular tests for JavaScript and CSS-only techniques

Test	Tests Ran		Found Primary		Primary/user (avg)		Secondary/user (avg)	
	JS	CSS	JS	CSS	JS	CSS	JS	CSS
top5k	206437	8165	76.1%	76.9%	12.7	9.8	49.9	34.6
top20k	31151	1263	75.4%	87.3%	13.6	15.1	48.1	51.0
All	32158	1325	69.7%	80.6%	15.3	20.0	49.1	61.2

An analysis of relative differences in susceptibility to history detection based on the user agent is shown in Table 4. For all browsers, the number of clients who were found vulnerable was above 70%. Browsers such as Safari and Chrome reported higher rates of susceptible clients (82% and 94% average), indicating that history detection can affect a significant number of Internet *power users*.

For users with at least one detected link tested with the JavaScript technique we detected an average of 12.7 websites (8 median) in the top5k list, as well as 49.9 (17 median) secondary resources. Users who executed the more-extensive JavaScript top20k test, were detected to have visited an average of 13.6 (7) pages with 48.2 (15) secondary resources. Similar results were returned for clients who executed the most elaborate `all` test, with 15.3 (7) primary links and 49.1 (14) secondary links. The distribution of top5k results for JavaScript-enabled browsers is shown in Figure 5. An important observation is that for a significant number of users (9.5%) our tests found more than 30 visited primary links; such clients are more vulnerable to deanonymization attacks and enumeration of user-specific preferences.

[5] Our testing system was featured on several social news sites and high-readership blogs, which increased the number of users who visited our website and helped in the overall data acquisition.

Table 4. Percentage of clients with detected links by User Agent

Test	IE		Firefox		Safari		Chrome		Opera	
	JS	CSS	JS	CSS	JS	CSS	JS	CSS	JS	CSS
top5k	73	92	75	77	83	79	93	100	70	82
top20k	81	95	69	86	89	97	90	100	88	95
All	78	97	62	79	85	89	87	98	85	83

Due to the fact that our testing site transparently reverted to CSS-only tests for clients with scripting disabled, we are also able to measure the relative differences in data gathered from clients with JavaScript disabled or unavailable. A total of 8,165 such clients executed the top5k test; results were found for 76.9% of clients, with a median of 5 visited primary URLs and 9 secondary URLs. Results for the top20k test executed in CSS yielded results similar to JavaScript clients, with 15.1 (8) websites and 51.0 (13) secondary links.

Interestingly, it seems that for certain tests, users without JavaScript appear more vulnerable due to a higher number of clients with at least one found link, and more detected links per client. This result should be an important consideration for organizations which decide to disable scripting for their employees for security reasons, as it demonstrates that such an approach does not make them any more secure against history detection attacks and associated privacy loss.

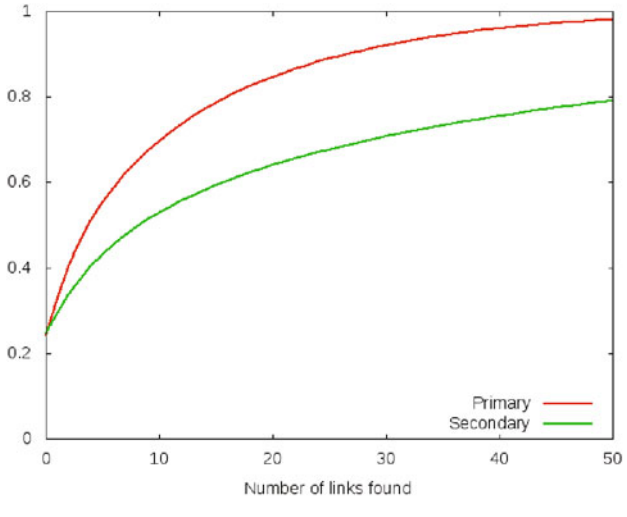


Fig. 5. Cumulative distribution of top5k primary and secondary links

5.2 Social News Site Links

An important overall part of our test system were tests of visited links from social news sites. We investigated three popular social news sites: Digg, Reddit and Slashdot. For each site, in addition to secondary links representing popular pages within that website, we also gathered all links to external destinations from the site's main RSS feed. We also checked for visits to the profile pages of active users within each site using the enumeration strategy outlined in Section 4.2.

We found that for users whose browsing history contained the link of the tested social news site, we could, in a significant majority of cases, detect resources linked from that site. History detection techniques could be used to measure user engagement on social news sites by comparing the average number of visited news stories; such analysis can be done both for individual users, and on aggregate, as a tool to compare social news site popularity. Additionally, for 2.4% of Reddit users we found that they visited the profile of at least one user of their social news site. Such data demonstrates that it is possible to perform large-scale information gathering about the existence of relationships between social news site users, and potentially deanonymize users who visit their own profile pages.

Table 5. Average and median numbers of found secondary links from social news sites

	Average secondary	Median secondary
Digg	51.8	7
Reddit	163.3	26
Slashdot	15.2	3

It is important to note that the specified architecture can potentially be used to determine user-specific preferences. Inspecting detected secondary links can allow a determined attacker to not only evaluate the relationship of a user with a particular news site, but also make guesses about the type of content of interest to the particular user.

5.3 Uncovering Private Information

For most history tests our approach was to show users the breadth of information about websites they visit which can be gleaned for their browsing history. However, we also created several tests which used the resource enumeration approach to detect common user inputs to popular web forms.

The zipcode test detected if the user typed in a valid US zipcode into a form on sites requiring zipcode information (there are several sites which ask the user to provide a zipcode to get information about local weather or movie showtimes). Our analysis shows that using this technique we could detect the US zipcode for as many as 9.2% users executing this test. As our test only covered several hand-picked websites, it is conceivable that with a larger selection of websites

requiring zip codes, the attack could be easily improved to yield a higher success rate.

In a similar test of queries typed into the Web forms of two popular search engines (Google and Bing) we found that it is feasible to detect some user inputs. While the number of users for whom search terms were detected was small (about 0.2% of users), the set of terms our test queried for was small (less than 10,000 phrases); we believe that in certain targeted attack scenarios it is possible to perform more comprehensive search term detection.

While limited in scope due to resource limitations, our results indicate that history detection can be practically used to uncover private, user-supplied information from certain Web forms for a considerable number of Internet users and can lead to targeted attacks against the users of particular websites.

6 Conclusions

This paper describes novel work on analyzing CSS-based history detection techniques and their impact on Internet users. History detection is a consequence of an established and ubiquitous W3C standard and has become a common tool employed in privacy research; as such, it has important implications for the privacy of Internet users. Full understanding of the implementation, performance, and browser handling of history detection methods is thus of high importance to the security community.

We described a basic cross-browser implementation of history detection in both CSS and JavaScript and analyzed Web browser behavior for content returned with various HTTP response codes and as frames or iframes. We provided an algorithm for efficient examination of large link sets and evaluated its performance in modern browsers. Compared to existing methods our approach is up to 6 times faster, and is able to detect up to 30,000 links per second in recent browsers on modern consumer-grade hardware. We also provided and analyzed results from our existing testing system, gathered from total number 271,576 of users. Our results indicate that at least 76% of Internet users are vulnerable to history detection; for a simple test of the most popular websites, we found, on average 62.6 visited URLs.

Our final contribution is the pioneering the data acquisition of history-based user preferences. Our analysis not only shows that it is feasible to recover such data, but, provided that it's large-scale enough, enables enumeration of privacy-relevant resources from users' browsing history. To our knowledge, this was the first such attempt. Our results prove that CSS-based history detection does work in practice on a large scale, can be realized with minimal resources, and is of great practical significance.

Acknowledgements. L.O. gratefully acknowledges financial support for this work from the European Organization for Nuclear Research (CERN) and, in particular, N. Neufeld for the help, support and fruitful discussions.

References

1. W3C: Cascading style sheets, level 1, http://www.w3.org/TR/REC-CSS1/
2. Bugzilla: Bug 57351 - css on a: visited can load an image and/or reveal if visitor been to a site (2000), https://bugzilla.mozilla.org/show_bug.cgi?id=57531
3. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: CCS 2000: Proceedings of the 7th ACM Conference on Computer and Communications Security, pp. 25–32. ACM, New York (2000)
4. Jagatic, T.N., Johnson, N.A., Jakobsson, M., Menczer, F.: Social phishing. ACM Commun. 50(10), 94–100 (2007)
5. Jackson, C., Bortz, A., Boneh, D., Mitchell, J.C.: Protecting browser state from web privacy attacks. In: WWW 2006: Proceedings of the 15th International Conference on World Wide Web, pp. 737–744. ACM, New York (2006)
6. Jakobsson, M., Stamm, S.: Web camouage: Protecting your clients from browsersning attacks. IEEE Security and Privacy 5, 16–24 (2007)
7. Webcollage: Web 2.0 collage, http://www.webcollage.com/
8. Wills, C.E., Zeljkovic, M.: A personalized approach to web privacy–awareness, attitudes and actions. Technical Report WPI-CS-TR-10-07, Computer Science Department, Worcester Polytechnic Institute (2010), http://www.cs.wpi.edu/~cew/papers/whattheyknow.pdf
9. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users, IEEE security and privacy. In: IEEE Security and Privacy, Oakland, CA, USA (2010)
10. Janc, A., Olejnik, L.: What the internet knows about you, http://www.wtikay.com/
11. Nielsen, J.: Change the color of visited links, http://www.useit.com/alertbox/20040503.html
12. Bugzilla: Bug 147777 - :visited support allows queries into global history (2002), https://bugzilla.mozilla.org/show_bug.cgi?id=147777
13. W3C: Cascading style sheets level 2 revision 1 (css 2.1) speci_cation, selectors, http://www.w3.org/TR/CSS2/selector.html#link-pseudo-classes
14. Jakobsson, M., Stamm, S.: Invasive browser sniffing and countermeasures. In: WWW 2006: Proceedings of the 15th International Conference on World Wide Web, pp. 523–532. ACM, New York (2006)
15. Jackson, C., Andrew Bortz, D.B.J.M.: Stanford safehistory, http://safehistory.com/
16. Baron, L.D.: Preventing attacks on a user's history through css : visited selectors (2010), http://dbaron.org/mozilla/visited-privacy
17. Jakobsson, M., Juels, A., Ratkiewicz, J.: Privacy-preserving history mining for web browsers. In: Web 2.0 Security and Privacy (2008)
18. Zalewski, M.: Browser security handbook, part 2 (2009), http://code.google.com/p/browsersec/wiki/Part2
19. König, F.: The art of wwwar: Web browsers as universal platforms for attacks on privacy, network security and rbitrary targets. Technical report (2008)
20. Quantcast: Quantcast, http://www.quantcast.com/
21. Anonymous: Did you watch porn, http://didyouwatchporn.com
22. Alexa: Alexa 500, http://alexa.com
23. Bloglines: Bloglines top feeds, http://www.bloglines.com/topblogs
24. Yahoo!: Yahoo! boss, http://developer.yahoo.com/search/boss/
25. Linode: Linode vps hosting, http://linode.com

On the Secrecy of Spread-Spectrum Flow Watermarks

Xiapu Luo, Junjie Zhang, Roberto Perdisci, and Wenke Lee

College of Computing, Georgia Institute of Technology
{csxpluo,jjzhang,wenke}@cc.gatech.edu, perdisci@gtisc.gatech.edu

Abstract. Spread-spectrum flow watermarks offer an invisible and ready-to-use flow watermarking scheme that can be employed to stealthily correlate the two ends of a network communication. Such technique has wide applications in network security and privacy. Although several methods have been proposed to detect various flow watermarks, few can effectively detect spread-spectrum flow watermarks. Moreover, there is currently no solution that allows end users to eliminate spread-spectrum flow watermarks from their flows *without* the support of a separate network element. In this paper, we propose a novel approach to detect spread-spectrum flow watermarks by leveraging their intrinsic features. Contrary to the common belief that Pseudo-Noise (PN) codes can render flow watermarks invisible, we prove that PN codes actually facilitate their detection. Furthermore, we propose a novel method based on TCP's flow-control mechanism that provides end users with the ability to autonomously remove spread-spectrum flow watermarks. We conducted extensive experiments on traffic flowing both through one-hop proxies in the PlanetLab network, and through Tor. The experimental results show that the proposed detection system can achieve up to 100% detection rate with zero false positives, and confirm that our elimination system can effectively remove spread-spectrum flow watermarks.

1 Introduction

Flow watermarks can be employed to trace end-to-end communications, even when they flow through stepping stones or anonymity networks [27]. By secretly embedding a (sequence of) watermark(s) into network flows at a location close to one end, it is possible to identify the other end of the communication by detecting the presence of the watermark in the traffic without being noticed by either end (see Figure 1). For example, flow watermarks may be used by law enforcement agencies to detect stepping stones used by attackers [20], to determine whether a certain user is accessing a specific (e.g., terrorism-related) web site [27,28], to trace communications among *bot*-compromised machines [21], to correlate anonymous Peer-to-Peer VoIP calls [26], etc.

If an adversary detects that her flows have been watermarked, she may be able to remove the watermarks, or deliberately cause false alarms by embedding the detected watermarks into legitimate flows [19,16]. Therefore, it is important

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 232–248, 2010.

to evaluate the *secrecy* of a flow watermarking scheme before deploying it in a real network. In this paper, we investigate the secrecy of spread-spectrum flow watermarks (SSFW) [28] from the following two aspects: (1) can SSFW be accurately detected? (2) can SSFW be effectively removed from network flows?

Recently, a few methods have been proposed that aim to detect SSFW [16,15]. Kiyavash et al. proposed a *multi-flow* attack, base on the assumption that SSFW is used to simultaneously embed the same watermark sequence into *multiple* flows. The detection approach leverages the length of low-throughput period as a metric to detect SSFW. The authors assume that that normal traffic follows the Markov-Modulated Poisson Process model, while watermarked flows would not fit this model [16]. However, it has been shown that the multi-flow attack can be evaded, if the encoder uses different Pseudo-Noise (PN) codes or if the watermark sequence changes for different flows [14]. The detection approach proposed by Jia et al. [15] leverages the fact that SSFW employs a single m-sequence, a specific PN code with good autocorrelation features, to spread the bits of a watermark sequence along a flow. Using only one m-sequence leads to obvious self-similarity in watermarked traffic. However, Jia et al. indicated that this detection approach may be evaded by using different m-sequences or orthogonal PN codes to spread individual bits of a watermark sequence [15].

To the best of our knowledge, there currently exists no solution that allows end users to remove SSFW from their flows without the support of a middlebox (e.g. a router, proxy, or a relay host within an anonymity network). In addition, although a middlebox may remove SSFW by altering the throughput of each individual network flow crossing it, few middleboxes actually deploy such watermark elimination strategy because of the consequent heavy overhead.

In this paper, we propose a novel detection system that is able to identify the existence of SSFW within a given network flow. In addition, we propose a novel elimination system that enables end users to autonomously remove SSFW from their flows. Our detection system leverages SSFW's intrinsic features. Unlike existing detection approaches (e.g., [16,28]), our approach does not assume that the same watermarks are simultaneously embedded in multiple flows, and does not assume an ideal traffic model. Moreover, we do not assume that SSFW uses only a single PN code. Instead, we assume that any other kind of valid PN codes (e.g. orthogonal PN codes) [12] could be employed.

Our detection approach is based on the following key observations: (1) similar to amplitude modulation in signal processing, SSFW causes alternate low-throughput and high-throughput periods in a watermarked flow; (2) PN codes make the detection of spread-spectrum flow watermarks easier, because they increase the number of low-throughput periods; (3) unlike spread-spectrum radio communications, which spread a radio signal over a wide frequency range, SSFW embeds watermarks separately in each *one* flow, instead of spreading them over a wide set of flows. Therefore, the detection system simply needs to examine individual flows (see Section 3).

Our elimination system leverages TCP's basic flow control mechanism to regulate the throughput of *incoming* traffic. More precisely, our system modifies the

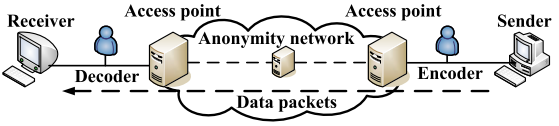


Fig. 1. Watermarks are embedded into the traffic by the Encoder, and identified by the Decoder

advertising window in TCP packets sent by either an end user or a middlebox to its upstream node to modulate the throughput. It is worth noting that our approach is independent from application-layer flow control and congestion control mechanisms. In fact, our approach works even in those cases when application layer flow control and congestion control mechanisms cannot remove SSFW.

In summary, this paper makes the following main contributions:

1. We propose a novel watermark detection system that is able to identify whether a flow has been watermarked using SSFW. Our approach removes many of the assumptions required by existing SSFW detection methods.
2. We propose a novel receiver-based system to remove SSFW. Our system can be deployed at either the end-user or middlebox level. To the best of our knowledge, ours is the first practical system that allows end-users to autonomously remove SSFW from their flows.
3. We performed extensive experiments to evaluate the proposed detection and elimination systems. The experimental results show that our system is able to successfully detect SSFW and remove the watermarks from TCP flows.

The rest of the paper is organized as follows. We describe the threat model and introduce related work in the next section. Section 3 and section 4 present the detection scheme and the elimination scheme respectively. We describe the experiment results in Section 5 and conclude the paper in Section 6.

2 Background

2.1 Threat Model

Figure 1 shows the threat model used in this paper. Assume an entity (e.g., law enforcement) intends to find out whether there exists an end-to-end network communication between a sender S and a receiver R. To this end, a watermark encoder E is placed between S and its neighbor network nodes, and a watermark decoder D is placed at the other end of the communication, between R and its neighbor nodes. E manipulates the throughput of all flows originating from S to embed a sequence of watermarks. On the other hand, all flows received by R are investigated by D to determine whether they carry watermarks previously embedded by E. If that is the case, this means that a communication between S and R is in place. Along the path between S and R there are n ($n \geq 1$) middleboxes. Each middlebox behaves as a proxy or relay host, therefore separating the

logical connection between S and R into multiple loosely coupled TCP connections. This scenario is typical of stepping stones [29,8], anonymity networks (e.g. Anonymizer (www.anonymizer.com) or Tor [6]), and HTTP/SOCKS proxies.

Our detection system (see Section 3) can be located at a middlebox, between the encoder and the decoder, to determine whether or not the flows going through the middlebox have been watermarked. If so, the middlebox can use a traffic shaper to remove the watermarks from the *outgoing* traffic sent to the next hop.

We also consider the case of non-cooperative middleboxes, and we assume the end user (the receiver) wants to make sure that her flows cannot be traced back. In this case, R can apply our elimination system (see Section 4) to blindly remove the watermarks from all her *incoming* flows before they can be identified by the decoder.

2.2 Spread-Spectrum Flow Watermarks

A target flow's throughput is the carrier of the spread-spectrum flow watermark. A watermark comprises of a sequence of bits denoted as $W = \{w_1, \ldots, w_M\}$, where M is the length of a watermark. Instead of using w_i $(i = 1, \ldots, M)$ to directly modulate a flow's throughput, the encoder first maps w_i to a PN code according to: $w_i \rightarrow \begin{cases} Z \text{ if } & w_i = 1, \\ \overline{Z} \text{ if } & w_i = -1, \end{cases}$ where $Z = \{z_1, \ldots, z_V\}$ is a V-bit PN code and $\overline{Z}$ is the complement of Z. After obtaining the new sequence of MV bits indicated as $W_{DSSS} = \{Z_1, \ldots, Z_M\}$, the encoder uses each bit in W_{DSSS} to modulate a flow's throughput. More precisely, if a bit is -1, the encoder will cause a low-throughput period of T_c (called *chip* duration) by causing many packet losses in the target flow. Otherwise, the encoder will maintain a high-throughput period of T_c by doing nothing or causing less packet loss in the target flow [7].

A PN code is a special binary sequence. Before introducing general features of a PN code, we give the definition of *run* in a binary sequence.

Definition 1. *Given a binary sequence $B = \{b_1, \ldots, b_L\}$, a run is defined as a sequence of $\{b_j, \ldots, b_k\}$ where $b_j = b_{j+1} = \ldots = b_k$ and $b_{j-1} \neq b_j$ and $b_{k+1} \neq b_k$. Its length is equal to $k-j+1$. Note that if $j = 1$, b_1 is the start of a run. Similarly, if $k = L$, b_L is the end of a run.*

Golomb indicated that a PN code may have one or many following properties [12]: (1) the number of 1 is approximately equal to the number of -1. (2) runs of 1 or -1 occur with probability that is inversely proportional to the length of runs. (3) its autocorrelation has the maximal value in the middle and declines quickly at the ends. Yu et. al. employed m-sequence, which has all above properties [12], to implement the spread-spectrum flow watermarks [28].

2.3 Countermeasures

Kiyavash et al. proposed the multi-flow attack to detect SSFW [16]. They assume that the same watermark is embedded into *multiple* flows simultaneously

and normal traffic follows the Markov modulated Poisson process [16]. Our detection system does not need such assumptions. The multi-flow attack exploits the observation that SSFW may cause a long low-throughput period on several flows comparing with a trained model. However, they also showed that the multi-flow attack can be evaded when the encoder applies different PN codes or flow watermarks to different flows [14]. Moreover, changing the position of a watermark in a flow may disable the multi-flow attack because the number of flow combinations that need investigation increases exponentially [16,14]. When facing multiple flows, our detection system only needs to investigate them one by one because we exploit the fundamental difference between network flows and radio signals, which is detailed in section 3.1.

The watermarked flow may show self-similarity because Yu et. al. used one m-sequence code to spread every bit in a watermark [28] and all m-sequence codes have excellent autocorrelation feature [7]. Exploiting this observation, Jia et al. proposed a detection approach that employs mean-square autocorrelation to measure the similarity between a modulated traffic segment and the same segment shifted by certain period [15]. However, it is easy to evade this method by using different m-sequence codes or orthogonal PN codes to spread individual bits of a watermark [15]. Our detection system can handle both cases.

3 Detection System

We first explain the traffic anomalies caused by SSFW in Section 3.1 and 3.2, and then elaborate on our detection scheme in section 3.3 and 3.4.

3.1 Basic Idea

Our detection scheme leverages the anomalies steming from SSFW's intrinsic features and takes advantage of fundamental differences between network flows and radio signals. We can first notice that when applied to a network flow SSFW causes an abnormal sequence of low-throughput periods in the flow. This happens because the encoder needs to throttle the flow's throughput to a low value for

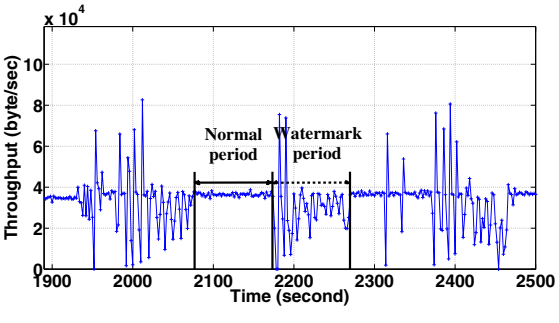


Fig. 2. Throughput of a watermarked flow that went through Tor network

a given period T_c when the bit to be embedded is -1 (notice that SSFW uses a binary encoding with values equal to either -1 or +1). The low-throughput periods caused by -1 bits are noticeably different from throughput degradations caused by network congestion in terms of the throughput level, duration and frequency. The reason is that network congestion is out of the control of the encoder and throughput degradations caused by network congestion may mislead the decoder. Therefore, to correctly decode a watermark bit and distinguish it from noise due to network congestion, Yu et al. have specified in [28] (Equation 12 and Figure 10) that the encoder needs to implement the following strategies:

- Increase the difference between the high-throughput and low-throughput levels. Since the maximum high-throughput is determined by the network, this strategy can only be achieved by decreasing the value of low-throughput.
- Increase the duration of low-throughput periods (i.e. T_c).
- Increase the length of the PN code, thus causing a higher number of low-throughput periods.

Based on this observations, the goal of our detection system is to detect SSFW by identifying the presence of anomalous sequences of low-throughput periods in a network flow. Figure 2, which is based on the data from [28], illustrates the throughput of a watermarked flow crossing the Tor network. Its throughput is computed in every *chip* duration (i.e. 2 seconds) [28]. We highlight two periods: the watermark period in which a watermark was embedded into the flow, and the normal period when the encoder is idle. It is easy to notice the higher number of low-throughput points during the watermark period, compared to the normal period. Since each low-throughput point in Figure 2 indicates the aggregated throughput during a chip duration, it represents a low-throughput period when the throughput is aggregated within a small time unit. We describe the selection of the basic time unit in section 3.3.

Second, we prove in section 3.2 that using PN codes to spread watermarks increases the number of low-throughput periods significantly. This feature allows our detection system to quickly identify SSFW.

Third, spread spectrum was originally designed to spread radio signal from a small frequency range to a wider frequency range [7]. A fundamental difference between radio signals and network flows is that although the spread-spectrum technique can spread the radio signal's energy to a wide range of frequencies and recover the original signal from those frequencies, applying SSFW to a network flow only affects that one flow and not a set of flows. Therefore, just like a decoder that only needs to inspect one flow to identify the embedded watermarks, our detection system only needs to investigate individual flows.

Based on the above observations, our detection scheme consists of two steps:

1. Locate low-throughput periods in a flow (section 3.3).
2. Detect abnormal sequences of low-throughput periods (section 3.4).

3.2 Low-Throughput Periods Resulted from PN Codes

We use R_1 and R_{-1} to denote the number of runs (see Section 2.2) of 1 and the number of runs of -1 in a binary sequence. R_{-1} is equal to the number of low-throughput periods. Without loss of generality, we assume that the flow's throughput is high before and after the watermark period. Since runs of 1 and runs of -1 alternate, the relationship between R_1 and R_{-1} falls into one of the following scenarios: (1) $R_{-1} = R_1$; (2) If $R_{-1} \neq R_1$ and $b_1 = -1$, $R_{-1} = R_1 + 1$; (3) If $R_{-1} \neq R_1$ and $b_1 = 1$, $R_{-1} = R_1 - 1$ [11].

Lemma 1 shows that the expected number of runs has the maximal value $1+\frac{L}{2}$ when the number of 1 is equal to the number of -1. For the ease of explanation, we assume that L is an even number. According to the relationship between R_{-1} and R_1 listed above, we know that the expected number of R_{-1} reaches its maximal value. Since PN codes have similar number of 1 and -1, they possess a large R_{-1}.

Lemma 1. *In a L-bit binary sequence, the expected number of runs reaches the maximal value $1 + \frac{L}{2}$ when the number of 1 is equal to that of -1.*

Proof. The expected number of runs (i.e. $R_{-1} + R_1$) in a L-bit binary sequence is equal to $1 + \frac{2L_{-1}(1-L_{-1})}{L}$ where L_{-1} is the number of -1 [11]. Since $2L_{-1}(L - L_{-1}) \leq \frac{L^2}{2}$ and the inequality becomes equality when $L_{-1} = \frac{L}{2}$, we get the maximal value $1 + \frac{L}{2}$ when the number of 1 is equal to that of -1.

Since SSFW turns an M-bit watermark into an MV-bit binary sequence using a V-bit PN code, R_{-1} is increased significantly. Without loss of generality, we assume that both the original watermark and the spread watermark are random sequences. In this case, we use the Corollary 2.1 in [11] to compute the probability of R_{-1}. Figure 3 illustrates R_{-1}'s PDF in 16-bit watermarks and that in the corresponding watermarks spread by 7-bit PN codes. Obviously, the spread watermarks have much larger R_{-1} than the original watermark. The average R_{-1} has been increased by around 7.

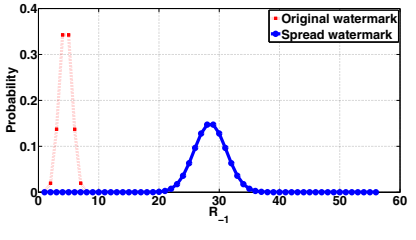


Fig. 3. The PDF of number of runs of -1 (i.e. R_{-1}) in 16-bit watermarks and that in the corresponding watermarks spread by 7-bit PN codes

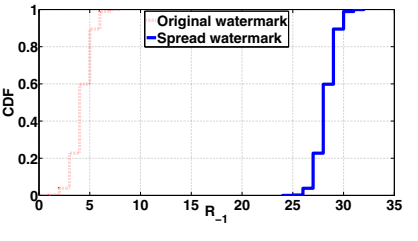


Fig. 4. The CDF of the number of runs of -1 (i.e. R_{-1}) in 16-bit original watermarks and that in the corresponding watermarks spread by a m-sequence PN code $\{1, -1, -1, 1, 1, 1, -1\}$

Lemma 2 calculates the exact number of R_{-1} in a m-sequence that is used as the PN code in [28]. Lemma 2 indicates that when using a L-bit m-sequence to spread one bit, the number of low-throughput periods will increase by around $\frac{L+1}{4}$. Figure 4 illustrates the CDF of R_{-1} in all possible 16-bit watermarks and that in the corresponding watermarks spread by a m-sequence PN code $\{1, -1, -1, 1, 1, 1, -1\}$. Obviously, the spread watermarks have much larger R_{-1} than the original watermarks.

Lemma 2. *In a L-bit m-sequence,* $R_{-1} = \frac{L+1}{4}$.

Proof. The number of k-bit runs of -1 is equal to 2^{J-2-k} $(k = 1, \ldots, J-2)$, where $J = \log_2(L+1)$ [7]. Since the maximal length of runs of -1 is $J-1$ and there is only one $(J-1)$-bit run of -1 [7], $R_{-1} = \sum_{k=1}^{J-2} 2^{J-2-k} + 1 = \frac{L+1}{4}$.

3.3 Locating Low-Throughput Periods

Our detection system computes a target flow's throughput in each basic time unit. We call these values as throughput samples. The system is independent of the transport layer protocol. For TCP flows, we let the basic time unit be the round-trip time (RTT), denoted as T_{rtt}, between the host where our detection system is located and its upstream host. The rational is that TCP packets are usually sent in burst within each RTT duration because of TCP's ACK-based self-clocking. Using a smaller period to compute throughput samples may lead to many useless zero values because the time for sending a burst of TCP packets is a small portion of RTT. Using a period larger than the chip duration to calculate throughput samples may blur low-throughput periods caused by the watermark. For UDP flows, the basic time unit could be set to the average inter-packet delay. Since the original SSFW targets on TCP flows [28] and many public proxies and anonymity networks (e.g. Tor) only support TCP connections, we evaluate our detection system using only TCP flows.

Given a sequence of throughput samples $\Pi = \{\pi_1, \pi_2, \ldots\}$, we construct a new binary sequence ($\hat{\Pi} = \{\hat{\pi}_1, \hat{\pi}_2, \ldots\}$) according to Equation (1):

$$\hat{\pi}_i = \begin{cases} 1 & \text{if } \pi_i - (1-\rho)\mu_\Pi > 0, \\ -1 & \text{if } \pi_i - (1-\rho)\mu_\Pi \leq 0, \end{cases} \tag{1}$$

where μ_Π is the average value of Π and ρ $(0 < \rho < 1)$ is a parameter. We define a low-throughput period as a sequence of throughput samples whose values are not larger than $(1-\rho)\mu_\Pi$ and the duration of such sequence is longer than T_r.

We found that it is proper to let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$, where μ_Π and σ_Π are the average value and the standard deviation of Π respectively. According to the one-side Chebyshev inequality [24], we have $Pr(\pi_i \leq \mu_\Pi - K\sigma_\Pi) \leq \frac{1}{1+K^2}$. When $K = 1$, $Pr(\pi_i \leq \mu_\Pi - \sigma_\Pi) = Pr(\hat{\pi}_i = -1) \leq \frac{1}{2}$. Since PN code $Z = \{z_1, \ldots, z_V\}$ has similar number of -1 and 1, $Pr(z_i = -1) \approx \frac{1}{2}$. For a V-bit m-sequence, $Pr(z_i = -1) \equiv \frac{V+1}{2V}$. Since SSFW degrades a flow's throughput when the bit to be embedded is -1, the probability of observing a low-throughput sample

approximates $\frac{1}{2}$. Therefore, by letting $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$, we have high probability to observe all low throughput values.

Throughput degradations caused by network congestion are noise to both SSFW's decoder and our detection algorithm. We exploit TCP's congestion control mechanism to filter out low-throughput periods caused by network congestions that occur on the path where the detection system is located. Detailed information can be found in [17].

3.4 Detection Algorithm

After locating a sequence of low-throughput periods, we employ the sequential probability ratio testing (SPRT) to carry out the detection. More precisely, as many recent Internet measurement studies have shown that the packet loss events in the Internet could be modeled as a Poisson process [2,18], we use SPRT to detect the abnormal increment in the rate of such events [9,13].

Let $x(t)$ be a poisson process modeling low-throughput periods. Its probability function is $Pr(x, \lambda) = \exp(-\lambda t)\frac{(\lambda t)^x}{x!}$, where λ is the rate of low-throughput periods. We define two hypotheses H_0 and H_1 as follows:

- H_0, the rate of low-throughput periods is within normal range.
- H_1, the rate of low-throughput periods is abnormal.

The log-likelihood ratio is defined as: $\Theta(t) = \ln \frac{Pr(x(t)|H_1)}{Pr(x(t)|H_0)} = x(t)\ln(\gamma) + \lambda_0(1-\gamma)t$, $\gamma = \frac{\lambda_1}{\lambda_0}$, where λ_0 and λ_1 indicate the normal rate of low-throughput periods and the abnormal one individually.

We choose H_0 if $\Theta(n) \leq B$ or select H_1 if $\Theta(n) \geq A$. Otherwise, the detection system continues monitoring. A and B are determined according to two user-defined parameters: α is the probability of false positive (i.e. select H_1 but H_0 is correct.) β is the probability of false negative (i.e. select H_0 but H_1 is correct.). Dvoretzky et. al. proved that $B = \ln \frac{\beta}{1-\alpha}$ and $A \leq \ln \frac{1-\beta}{\alpha} \leq A + \ln(\gamma)$ [9]. Since computing the exact value of A is time-consuming, Haggstrom suggested that $A \approx \ln(\frac{1-\beta}{\alpha}) - \frac{\ln(\gamma)}{3}$ [13].

Note that waiting periods, denoted as y, between consecutive events in a Poisson process follow the exponential distribution, whose probability function is $\lambda e^{-y\lambda}$. Haggstrom constructed an alternative SPRT of H_0 versus H_1 based on the waiting times after N observations. He proved that these two SPRTs will lead to the same decision and showed that the expected number of events when the SPRT stops is $E(N|\lambda_1) = E[x(t)|\lambda_1] + L(\lambda_1)$, where $L(\lambda_1)$ is the operating characteristic function of the test H_1 [13]. As Haggstrom has detailed every step of computing $E(N|\lambda_1)$, interested readers please refer to that report [13].

4 Elimination System

An effective approach to remove SSFW is to shape a flow's throughput. Although it is easy to regulate the *outgoing* traffic using mechanisms like Linux traffic

control or Tor's bandwidth limit on relayed traffic, they can not regulate the *incoming* traffic. Therefore, if an upstream middlebox does not shape outgoing traffic, a downstream host will receive watermarked traffic.

Public proxies and one-hop anonymity network like Anonymizer usually do not apply traffic shaping to avoid performance degradation. Although Tor uses a windowing scheme for each circuit to prevent congestion [6,22], it could not eliminate SSFW because it only limits the high throughput instead of removing the low-through periods. Though long delays introduced by the Tor network may affect SSFW's decoding rate, new mechanisms for increasing the performance of Tor network [23] will mitigate the noise to SSFW's encoding/decoding procedure. Therefore, end users need solutions to remove SSFW by themselves.

Being a complement to existing traffic control mechanism, our elimination system allows a middlebox or an end user to shape the throughput of *incoming* traffic. More precisely, our system modifies the advertising window in outgoing packets and adds additional delays if necessary. We employ the *leaky bucket* algorithm to determine the throughput of incoming traffic [25].

Let S_{pkt} and N_{pkt} denote the packet size and the number of packets received in a T_{rtt}. N_{pkt} is controlled by the available data in the TCP sender, its congestion window (i.e. `cwnd`) and the advertising window (i.e. `rwnd`) announced by the receiver. The instantaneous throughput is equal to $\frac{S_{pkt} \times N_{pkt}}{T_{rtt}}$. To scrub SSFW, we manipulate N_{pkt} and introduce additional delay, named T_{dly}. The throughput becomes $\frac{S_{pkt} \times \widetilde{N}_{pkt}}{T_{rtt}+T_{dly}}$, where $\widetilde{N}_{pkt}$ is the number of packets received during the period of $T_{rtt} + T_{dly}$. More precisely, whenever a packet is going to be sent, the elimination system delays it for T_{dly} and checks whether there is enough quota, denoted as `BukCap`, to *receive* a packet of S_{pkt} bytes from the upstream host. If so, our system changes the advertising window in that packet to S_{pkt}. Otherwise, the packet's advertising window will be set to 0 (or 1 for unpatched windows Vista/2003/2008 that do not handle zero window correctly [5]). If a packet of size S_{pkt} is received, `BukCap` is decreased by S_{pkt}. The quota will be recovered to a pre-defined value every second. Since sometimes the TCP receiver does not has outgoing packets, our system will generate an ACK packet every 200 ms to trigger packets from the TCP sender. These ACK packets' advertised windows are set according to the above leaky bucket algorithm.

5 Evaluation

We implemented SSFW's encoder, decoder and our detection and elimination system on Linux with the help of iptables 1.4.0, the libnetfilter_queue 0.0.16 library and Linux raw socket. We realized the advanced encoding approach suggested in [28,15]. Given a dropping probability, this approach discards packets probabilistically during a T_c period if the bit to be embedded is -1. Detailed information can be found in [17] due to limited space.

We evaluated our detection system using both the traces in [28] and the traces collected by ourselves. The traces in [28] include watermarked flows going through the Tor network. Our traces include watermarked flows going through

12 PlanetLab nodes that are located in different countries [17]. It represents the scenario of using public proxies or one-hop anonymity network like Anonymizer. Since Anonymizer does not provide free trail service now, we run a light-weight HTTP proxy, `Tiny HTTP Proxy` [1], on those PlanetLab nodes. In this case, the sender becomes a web server and the receiver downloads files from it.

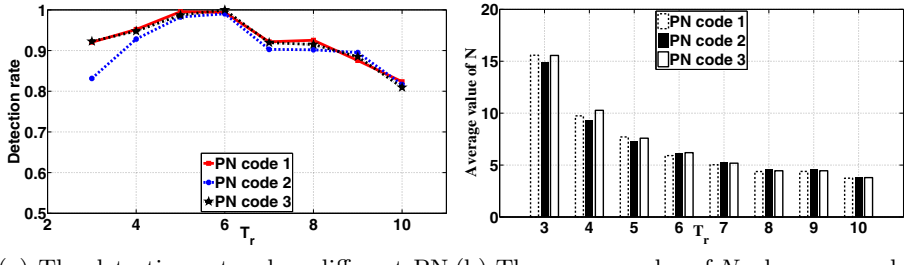
Recall that there are six parameters used in our detection system: α and β are user-defined false positive rate and false negative rate; T_r determines the minimal duration of a low-throughput period; ρ is related to the deepness of a low-throughput period; λ_0 and λ_1 are the expected normal rate and abnormal rate of low-throughput periods. In our evaluation, we fix two parameters' value (i.e. α and β), formulate the computation of another two parameters (i.e. λ_0 and λ_1) and examine the effect of the remaining two (i.e. T_r and ρ). The major reason is that only T_r and ρ affect the number of low-throughput periods, whose abnormal behavior is the basis of our detection system. We set $\alpha = 0.001$ and $\beta = 0.01$. Let Λ be the set of rates of low-throughput periods calculated from the training data. In all of our experiments, we let λ_0 be the mean value of Λ and λ_1 be the maximal value in Λ.

5.1 Evaluation of the Detection System Using Planetlab Traces

In the PlanetLab experiments, we used the same watermark (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$) in [28] and three chip durations T_c (i.e. 1s, 2s and 3s). We evaluated the detection system using three different PN codes:

1. *PN code 1*: the PN code mentioned in [28] (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$).
2. *PN code 2*: a m-sequence $\{1, -1, -1, 1, 1, 1, -1\}$ generated by [4].
3. *PN code 3*: a pair of Walsh-Hadamard code generated by [4] that spreads 1 and -1 using $\{1, -1, 1, -1, 1, -1, 1\}$ and $\{1, 1, -1, -1, 1, 1, -1\}$ respectively. They are orthogonal codes [10].

We downloaded a large file from the web server through each proxy 100 times. Half of the traces were used to compute the RTT between the client and the proxy, λ_0 and λ_1 used in the SPRT. The remaining traces were used to evaluate



(a) The detection rate when different PN codes were used

(b) The average value of N when a spread-spectrum flow watermark was detected

Fig. 5. The detection rate and the average value of N v.s. T_r

our system's false positive. Then we downloaded the same file 150 times through each proxy and the encoder started embedding the watermark to the flows.

We first examined the effect of different PN codes and that of T_r on the detection rate and false positive. In these experiments, we let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$. Figure 5(a) illustrates the impact of T_r on the detection rate. It is worth noting that when T_r changes the number of low-throughput periods in all flows may also vary. Consequently, Λ and λ_1 may also change. We found that when T_r increases from a small value 3 to a large value 10 the false positive remains *zero* while the detection rate first raises and then decreases.

The low false positive rate may result from the fact that we let $\lambda_1 = \max(\Lambda)$. The reason for the trend of detection rate is two-fold. On the one hand, using a small T_r may include many short low-throughput periods in both normal flows and watermarked flows. While the difference between a normal flow and a watermarked flow is that the latter has more *long* low-throughput periods, a large number of short low-throughput periods may obscure this feature and cause more false alarms. On the other hand, when T_r is too large (e.g. much larger than T_c) the low-throughput periods caused by SSFW with small chip duration may be ignored and therefore the detection rate decreases. It is rational to let T_r be 4 or 5 because from a decoder's point of view T_c should be larger than several RTTs to mitigate the noise from ordinary network congestion, where a TCP sender needs a few RTTs to recover its throughput from a mild network congestion. Such recovery process may lead to low-throughput periods similar to the effect of embedding flow watermark. Therefore if T_c is shorter than such recovery process, ordinary network congestion may give rise to decoding errors.

Figure 5(b) illustrates the average number of low-throughput periods needed by our system to raise an alarm. As T_r increases, less number of low-throughput periods is needed because the number of low-throughput periods in both normal flows and watermarked flows decreases. Since each flow carried only one watermark in our experiments, the decoder needs to observe the whole flow before recovering the watermark. Therefore, if a watermarked flow is detected before the end of the flow, the watermark must be identified before the decoder has the chance to recover the watermark. All detected flows in our experiments were identified before the end of each flow. A conservative approach is to let the middlebox shape the outgoing traffic when it uses our detection system to determine the existence of SSFW in incoming traffic.

When examining the effect of different PN codes, we observed from Figure 5(a) that the result of the *PN code 1* and that of the *PN code 3* are similar. It may be due to the similarity in their run properties. To encode the watermark (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$) , *PN code 1* has 18 runs of -1 including twelve 1-bit runs and six 2-bit runs. *PN code 3* also has 18 runs of -1 including fifteen 1-bit runs and three 2-bit runs. All these runs of -1 will cause low-throughput periods. In comparison with them, *PN code 2* has only 11 runs of -1 including one 1-bit run, seven 2-bit runs and three 3-bit runs. However, Figure 5(b) illustrates that to detect these watermarks the number of required low-throughput period for different PN codes is similar.

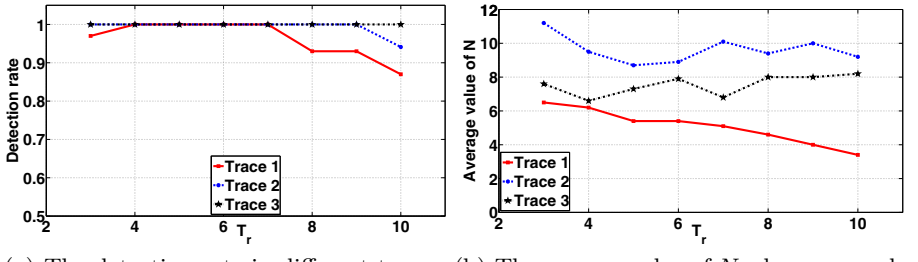
Table 1. Average detection rate (R_d) and false positive rate v.s. K ($\rho = K\frac{\sigma_\Pi}{\mu_\Pi}$)

	PN Code 1		PN Code 2		PN Code 3		
K	R_d	N	R_d	N	R_d	N	False Positive
1	0.996	7.71	0.983	7.26	0.987	7.59	0
2	0.939	5.53	0.908	5.53	0.935	5.76	0
3	0.94	4.08	0.91	4.26	0.934	4.3	0.003
4	0.781	4.1	0.751	4.36	0.795	4.28	0.006
5	0.74	3.82	0.655	3.99	0.711	4.01	0.006

To evaluate the effect of ρ, we let $T_r = 5$ and $\rho = K\frac{\sigma_\Pi}{\mu_\Pi}$ and then increase K from 1 to 5. The experiment result is listed in Table 1, which includes the detection rate (R_d) and the average number of N for different PN codes and the false positive rate. We can see that the best detection rate is achieved when $K = 1$. This observation is in accordance with the analysis in section 3.3. When K increases, the detection rate decreases. The reason is that a large ρ may filter out many low-throughput periods caused by the flow watermarks, for example, those generated by small dropping probability (e.g. 0.1). However, when K is not larger than 3, the detection rate is still larger than 90%. Moreover, a very large ρ may also increase the false positive rate because it may lead to a very small λ_0 (i.e. most normal flows do not have such kind of low-throughput periods) and flows with a few throughput outliers might cause false positive.

5.2 Evaluation of the Detection System Using Tor Traces from [28]

In [28]'s Tor experiments, the encoder used 98 seconds to embed a watermark because the chip duration is 2 seconds and both the m-sequence and the watermark have 7 bits (i.e. 2*7*7=98). After that, their encoder will wait for 98 seconds before embeding another watermark to the same flow [28].



(a) The detection rate in different traces (b) The average value of N when a spread-spectrum flow watermark was detected

Fig. 6. The detection rate and the average value of N v.s. T_r

The author of [28] provided us three traces that contain throughput aggregated in 0.1s. We let the basic unit time be 0.1s because we could not know the exact RTT from the traces. For each trace, we first identified normal periods and

watermark periods and then divided the normal periods into two groups. Based on the data in the first group of normal periods, we computed λ_0, λ_1, σ_Π and μ_Π. Then, we applied the detection algorithms to the data in the second group of normal periods to calculate the false positive rate and applied the detection algorithms to data in watermark periods to compute the detection rate.

Figure 6(a) shows the detection rate in different traces when T_r varies. In most cases our detection scheme can identify all watermarks. We observed that when T_r increases from a small value 3 to a large value 10 the false positive remains *zero*. Figure 6(b) illustrates the average number of low-throughput periods needed by our detection system to raise an alarm. Compared to the PlanetLab experiment results shown in Figure 5(b), the number of steps needed to detect watermarked flows going through Tor is relatively stable. It is because λ_0 and λ_1 computed from normal periods in those Tor traces are less sensitive to T_r. In these experiments, we let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$. When fixing T_r to 5, we still get *zero* false positive when $\rho = 2\frac{\sigma_\Pi}{\mu_\Pi}$.

5.3 Evaluation of the Elimination System

In [28], a flow is deemed as being watermarked if and only if the decoder can recover the 7-bit watermark. The experiment results showed that our elimination system can successfully remove SSFW from flows going through both one-hop proxies and Tor network. To explain the result clearly, we define the bit decoding rate as the ratio of the number of bits decoded correctly to the length of a watermark. A watermark is removed if the bit decoding rate is less than 1.

Table 2. Bit decoding rate and throughput ratio after a watermarked flow is processed by our elimination system

Dropping probability	0.1	0.2	0.4	0.6
Average bit decoding rate	0.651	0.51	0.5	0.571
Throughput ratio	0.771	0.672	0.536	0.424

Since a SSFW encoder degrades the throughput of the target TCP flow, the major goal of our elimination system is to remove SSFW and at the same time minimize the negative effect on its throughput. Table 2 lists the average bit decoding rate and throughput ratio obtained from the flows between a PlanetLab node in UK (194.36.10.154) and the client when the encoder adopted different dropping probability. The throughput ratio indicates the ratio of the average throughput of watermarked flows that were scrubbed by our elimination system to the average throughput of watermarked flows. In these experiments, we set the regulated throughput to the median of the throughput of watermarked flows. The experiment results showed that *all* watermarks were removed. When the dropping probability is small, the throughput degradation is around 23%. When the dropping probability increases, the throughput degradation becomes severe. The reason is that under high dropping probability the modulated TCP's throughput is already very low and consequently its median value (i.e. the regulated throughput) is very small.

We also applied our elimination system to flows going through the Tor network. Since Tor changed the paths during the experiments, we set the regulated throughput to fix values. Figure 7 shows a box plot of the bit decoding rates after our elimination system regulated the incoming traffic from Tor's entry node. The dot within each box indicates the median value of bit decoding rate. We can see that *all* watermarks were removed.

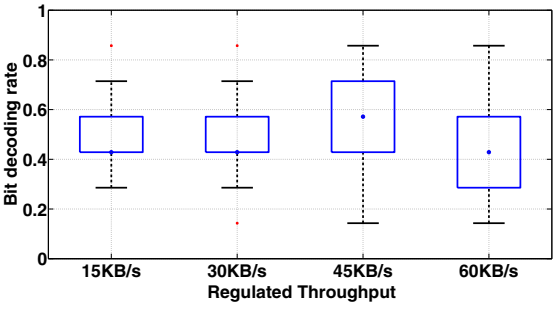


Fig. 7. Bit decoding rate v.s. regulated throughput

6 Conclusion

In this paper we proposed a novel method to detect spread-spectrum flow watermarks (SSFW). Our method leverages the intrinsic features of SSFW, and is mainly based on detecting anomalous sequences of low-throughput periods in network flows. Furthermore, we introduced a novel receiver-based approach to remove SSFW by leveraging TCP's flow control mechanism. This approach is complementary to router-based traffic shaping methods, and allows end users to autonomously remove SSFW even in case of non-cooperative middleboxes. We conducted an extensive evaluation of our watermark detection and elimination systems. The experimental results confirm that our detection approach is able to identify SSFW quickly and with high accuracy, and that our elimination system can effectively remove SSFW from network flows. In the further work, we will investigate how to mitigate the throughput degradation caused by the elimination system. Another possible direction is to select more suitable carriers and watermarks for the design of flow watermarks [3].

Acknowledgments

We thank Xinwen Fu and Ling Zhen for giving us Tor traces and suggestions on Tor's experiments. This material is based upon work supported in part by the National Science Foundation under grants no. 0716570 and 0831300, the Department of Homeland Security under contract no. FA8750-08-2-0141, the Office of Naval Research under grant no. N000140911042. 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, the Department of Homeland Security, or the Office of Naval Research.

References

[1] hisao, S. (2009), http://www.okisoft.co.jp/esc/python/proxy

[2] Altman, E., Avrachenkov, K., Barakat, C.: A stochastic model for tcp with stationary random losses. In: ACM SIGCOMM (2000)

[3] Cayre, F., Fontaine, C., Furon, T.: Watermarking security: Theory and practice. IEEE Transactions on Signal Processing 53(10), 3976–3987 (2005)

[4] Choi, B.: PN code generator (2000), http://www-mobile.ecs.soton.ac.uk/bjc97r/pnseq-1.1/pnseq-1.1.tar.gz

[5] Microsoft Corporation. Microsoft security bulletin ms09-048 (2009), http://www.microsoft.com/technet/security/Bulletin/ms09-048.mspx

[6] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX SEC (2004)

[7] Dixon, R.: Spread Spectrum Systems, 2nd edn. John Wiley & Sons, Chichester (1984)

[8] Donoho, D., Flesia, A., Shankar, U., Paxson, V., Coit, J., Staniford, S.: Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516, p. 17. Springer, Heidelberg (2002)

[9] Dvoretzky, A., Kiefer, J., Wolfowitz, J.: Sequential decision problems for processes with continuous time parameter testing hypotheses. Annals of Mathematical Statistics 24 (1953)

[10] Fazel, K., Kaiser, S.: Multi-Carrier and Spread Spectrum Systems. Wiley, Chichester (2003)

[11] Gibbons, J., Chakraborti, S.: Nonparametric Statistical Inference, 4th edn. CRC, Boca Raton (2003)

[12] Golomb, S.: Shift Register Sequences (revised edition). Aegean Park Press, Laguna Hills (1982)

[13] Haggstrom, G.: Sequential tests for exponential populations and poisson processes. Technical report, RAND Corporation (1979)

[14] Houmansadr, A., Kiyavash, N., Borisov, N.: Multi-flow attack resistant watermarks for network flows. In: IEEE ICASSP (2009)

[15] Jia, W., Tso, F., Ling, Z., Fu, X., Xuan, D., Yu, W.: Blind detection of spread spectrum flow watermarks. In: IEEE INFOCOM (2009)

[16] Kiyavash, N., HoumanSadr, A., Borisov, N.: Multi-flow attacks against network flow watermarking schemes. In: USENIX Security (2008)

[17] Luo, X., Zhang, J., Perdisci, R., Lee, W.: On the secrecy of spread-spectrum flow watermarks (2010), http://roberto.perdisci.com/publications/publication-files/DSSSWM_Extended_TechReport.pdf

[18] Markopoulou, A., Tobagi, F., Karam, M.: Loss and delay measurements of internet backbones. Computer communications (June 2006)

[19] Peng, P., Ning, P., Reeves, D.: On the secrecy of timing-based active watermarking trace-back techniques. In: IEEE Symp. on Security and Privacy (2006)

[20] Pyun, Y., Park, Y., Wang, X., Reeves, D., Ning, P.: Tracing traffic through intermediate hosts that repacketize flows. In: IEEE INFOCOM (2007)

[21] Ramsbrock, D., Wang, X., Jiang, X.: A first step towards live botmaster traceback. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 59–77. Springer, Heidelberg (2008)

[22] Reardon, J., Goldberg, I.: Improving tor using a TCP-over-DTLS tunnel. In: USENIX Security (2009)
[23] Tang, C., Goldberg, I.: An improved algorithm for Tor circuit scheduling. Technical report, University of Waterloo (2010)
[24] Therrien, C., Tummala, M.: Probability for Electrical and Computer Engineers. CRC, Boca Raton (2004)
[25] Turner, J.: New directions in communications (or which way to the information age?). In: IEEE Commun. Magazine (1986)
[26] Wang, X., Chen, S., Jajodia, S.: Tracking anonymous peer-to-peer voip calls on the internet. In: ACM CCS (2005)
[27] Wang, X., Chen, S., Jajodia, S.: Network flow watermarking attack on low-latency anonymous communication systems. In: IEEE Symp. on Security and Privacy (2007)
[28] Yu, W., Fu, X., Graham, S., Xuan, D., Zhao, W.: DSSS-based flow marking technique for invisible traceback. In: IEEE Symp. on Security and Privacy (2007)
[29] Zhang, Y., Paxson, V.: Detecting stepping stones. In: USENIX Security (2000)

Traffic Analysis against Low-Latency Anonymity Networks Using Available Bandwidth Estimation*

Sambuddho Chakravarty[1], Angelos Stavrou[2], and Angelos D. Keromytis[1]

[1] Columbia University, NY, USA
{sc2516,angelos}@cs.columbia.edu
[2] George Mason University, VA, USA
astavrou@gmu.edu

Abstract. We introduce a novel remotely-mounted attack that can expose the network identity of an anonymous client, hidden service, and anonymizing proxies. To achieve this, we employ *single-end controlled* available bandwidth estimation tools and a colluding network entity that can modulate the traffic destined for the victim. To expose the circuit including the source, we inject a number of short or one large burst of traffic. Although timing attacks have been successful against anonymity networks, they require either a *Global Adversary* or the compromise of substantial number of anonymity nodes. Our technique does not require compromise of, or collaboration with, **any** such entity.

To validate our attack, we performed a series of experiments using different network conditions and locations for the adversaries on both controlled and real-world *Tor* circuits. Our results demonstrate that our attack is successful in controlled environments. In real-world scenarios, even an under-provisioned adversary with only a few network vantage points can, under certain conditions, successfully identify the IP address of both Tor users and *Hidden Servers*. However, Tor's inherent circuit scheduling results in limited quality of service for its users. This at times leads to increased false negatives and it can degrade the performance of our circuit detection. We believe that as high speed anonymity networks become readily available, a well-provisioned adversary, with a partial or inferred network "map", will be able to partially or fully expose anonymous users.

1 Introduction

Low-latency anonymity systems strive to protect the network identity of users that are in need of services and applications that are latency sensitive or interactive in nature. Tor [27], JAP [22] and I2P [4] are examples of popular low-latency

* This work was partly supported by the Office for Naval Research through Grant N0014-09-1-0757 and the National Science Foundation through Grant CNS-07-14277. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ONR or the NSF.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 249–266, 2010.

anonymity systems. The Achilles' heel of these systems lies in the fact that they do not modify the inter-packet delay to achieve low end-to-end latency. However, this makes them vulnerable to traffic pattern observation attacks [26]. Indeed, the adversary can correlate flow patterns in traffic flowing through one section of the network to that in another[1]. Timing attacks are feasible only by an adversary who can observe traffic in **all** the network links called a Global Adversary (GA). A key design assumption behind low-latency anonymizing systems it requires a tremendous amount of resources and effort to become a GA. Therefore, the goal is to provide adequate protection against all but a determined, and possibly targeted, attack by a GA.

Recently, however, a number of practical attacks against such anonymity systems have been proposed. These attacks typically leverage a small number of compromised network entities to partially or fully expose information about a user of these systems [7,10,20,25,32,16]. Nonetheless, it is generally assumed that consistent tracking of the users of such anonymity systems is impractical for the large majority of users and organizations, because of the lack of many omnipresent adversaries. *We evaluate this assumption for present and future high-speed anonymity networks.*

Furthermore, popular low-latency anonymizing systems including Tor cannot guarantee consistent quality of service. This is due to a combination of the cryptographic computation, queuing delays, and traffic scheduling and conditioning [28]. Interestingly, this also degrades the adversaries ability to successfully track the anonymous users. Having low volume of traffic causes traffic analysis techniques to be less accurate [16]. *In this paper, we study the validity of this belief in a next-generation network and host infrastructure where the speed or throughput of the anonymizing network is not limited by the computational or communication capacity of the participating entities.*

To that end, we present a novel and effective method for performing source "trace-back" . Our approach can be employed to perform targeted traffic analysis against most low-latency anonymizing systems[2]. Contrary to the work by Murdoch *et al.* on traffic analysis for Tor relays [25], our technique can track not only the anonymizing relays, but also the victim's network location without the need of malicious Tor clients. We assume that the adversary controls, or can create a traffic fluctuation, in one end of the communication channel. The adversary modulates the traffic destined for the victim using a colluding end-point. Thereby, he or she attempts to detect the artificially induced traffic variations as they propagate over the candidate anonymizing proxies and routers, eventually leading to the anonymous client.

To observe the induced traffic fluctuation, the attacker requires novel tools that can quickly and accurately characterize the available bandwidth of individual links and entire network paths. We previously introduced LinkWidth [14],

[1] High latency anonymity systems such as *Mixminion* [17] modify the inter-packet latencies to curtail traffic analysis.

[2] In our experiments, we used Tor but our approach is applicable to all proxy-based anonymity systems.

a *single-end controlled* available-bandwidth estimation tool that is based on the algorithm behind TCP Westwood [18]. LinkWidth does not require a TCP listening server. Furthermore, it offers bandwidth estimation of relays and routers connecting the victim clients (and Hidden Servers) to the anonymizing network. We focus on using such available bandwidth estimation for sensing deliberately induced TCP traffic fluctuations and confirming the probable identity of anonymous end points.

To verify the validity of our approach, we performed a series of experiments. We evaluated the predictive accuracy of our technique on controlled environments using different network topologies. We also tested our technique on Tor circuits created using Tor nodes that are part of the public Tor network. In our experiments, we achieved varying success in exposing the identity of the victims. On an average, we detected 49.3% of the Tor relays participating in our circuits. In addition, we were also successful several times in identifying the intermediate network routers connecting Tor clients and Hidden Services to their Entry Nodes.

We posit that a real adversary equipped with many appropriately located high-bandwidth hosts and a partial map of the network, could effectively attack timing-preserving anonymizing systems. Some prior efforts in generating such maps are the *Internet Mapping Project* [6], *AS Peering Analysis* [1], *iplane* [23], and similar efforts by CAIDA [2]. Searching a map depicting the ingress and egress routers of an Autonomous System (AS) involves little complexity. An adversary equipped with such information would probe for the induced available bandwidth variation, **only on** the inter-domain routers in search for the AS that hosts the victim anonymous client[3].

The objective of this paper is not to demonstrate this search process but to answer the following question: *Can a well-provisioned (Semi-)Global Adversary equipped with probing nodes scattered close to most inter-AS routers, and "sender-only" controlled probing tools like LinkWidth, measure bandwidth fluctuation on network routers and anonymizing relays associated to a communication session?*

Having a large number of distributed measurement nodes ("vantage points") allows for better coverage of the network links. However, there is no requirement for network affinity of the vantage points to the victim or the Tor relays used in the circuit. Moreover, we do not assume that the attacker has access to routers, network infrastructure nodes (*e.g.,* DNS or DHCP servers), or anonymizing relays; nor do we exploit software vulnerabilities that inadvertently expose the true network identity of the user. *The novelty of our technique lies in aiding an adversary with bandwidth resources to perform traffic analysis for determining anonymizing network's entry-points, anonymous clients and location hidden services in a communication session.*

[3] Further, resolving down to the end hosts might require ISP router maps through services such as as *Rocketfuel* [5].

2 Related Work

The majority of the attacks against low-latency anonymization systems, employ some form of traffic analysis. Many of the successful methods apply *active* techniques, employing practical *predecessor attack* [34] derivatives like [30,9,35,16], which involve inside network elements. Other suggest *passive* attacks that simply count the number of packets at ingress and egress links of the anonymizing network [8].

There are yet some other attacks which try to track changes in non-anonymous system parameters (*e.g.* CPU and memory usage) in network nodes due to change in anonymous traffic flowing through them. Such attacks are called *side channel* attacks. Some examples of *side channel* attacks are presented in references [36,24]. Here the adversary observes changes in CPU usage by observing skews in TCP timestamps, that are brought about by changes in CPU temperature. Other such attacks use network latency as a side-channel to reveal identity of anonymous network system parameters [20,25]. The authors of those papers demonstrate the use of inter-packet latency and round-trip times (RTT) to reveal identity of anonymous peers and anonymizing network elements.

Although significant, this class of attacks can be ineffective when the bottlenecks of the network paths connecting the anonymously communicating peers and the adversary to the candidate networking elements, do not coincide. In such situations, the adversary's measurement capabilities, constrained by the bottleneck link speed, might not accurately detect network link contentions due to the anonymous traffic. Such contentions are essential to perceive changes in the measured RTT. The poor end-to-end QoS of popular systems such as Tor, further degrades the attacker's ability to launch such attacks. Low volume traffic leads to low network congestion; thereby causing little variations in measured RTT between the adversary and the victim. Results by Evans *et al.* indicate that this intuition is likely correct [16].

Our approach stems out of RTT based traffic analysis [25,20], and partially from traffic pattern injection techniques [31,21]. Unlike predecessor attack derivatives, our attack assumes nominal control of networking elements. The adversary induces end-to-end network traffic fluctuation by colluding with a malicious server with which anonymous clients communicate. Using Linkwidth, the adversary tries to observe the induced bandwidth fluctuations on candidate anonymizing network elements and routers leading to the anonymous client.

Our approach seems similar to targeted denial of service based technique, presented by Burch and Cheswick to "trace-back" to the source of a DoS source that used IP spoofing [11]. Their aim was to cause interference in the remote routers and notice fluctuations in the attack-DoS traffic. Using a network map they would iteratively trace-back to eventually identify the source of a DoS, or at least its approximate location (*e.g.,* hosting ISP). Our technique is similar to theirs. We, however, aim to "trace" available bandwidth fluctuations on network routers and anonymizing proxies that carry traffic between the anonymous parties.

3 Attack Methodology

Before delving into the attack details, we discuss the threat model for which we claim that our attacks are effective.

Threat Model

Our primary focus is an adversary who can induce "traffic fluctuations" into a targeted anonymity preserving channel and observe it "trickle" towards the victim. The adversary may have hosts under his control on many ASes or could be at a network vantage point with respect to routers in various subnets. Each of these hosts may be running a copy of bandwidth estimation tools such as LinkWidth. They would also be at a network vantage point with respect to all candidate victim relays and network routers. This is feasible in toady's' Internet: the inter-router media connecting the routers of major tier-3 ISPs can support over 10 Gbps (let alone the tier-1 and tier-2 ISPs). Most have 30–40% under-utilized spare capacity. Therefore, while we are not in a position of having a large set of vantage points, this does not prevent others from being able to do so. Often, adversaries like us, might be sensing "under-utilized high-bandwidth" links. Popular anonymity preserving systems, like Tor relays, often dedicate a considerable fraction of the traffic to forwarding client traffic. However, such small fluctuations in high capacity links, are less than what most state-of-the-art available link bandwidth estimation techniques can detect accurately. Nevertheless, we demonstrate that for some common network topology configurations and parameters, an adversary can harness bandwidth estimation to trace the "unknown" path a client uses to connect to a server.

Traffic Analysis Methodology

We used available bandwidth estimation tools to detect induced traffic fluctuations on candidate anonymizing relays. We identify probable candidate network routers that could reach this relay through a single network hop. Since most anonymity preserving relays are at the network edges, the network routers that could directly reach the candidate relays would be their default gateways. This intuition was re-applied on default gateways to determine which network interface, and hence the network routers within one network hop, exhibited similar fluctuations. We repeated the tracking process recursively until the fluctuations were tracked down to the source network (and possibly the source itself).

To quantify our detection capabilities, we performed extensive experiments initially in a controlled emulation environments, viz. DETER [3]. Some of the experiments were further validated in controlled lab-environment. We also uncovered real-world Tor clients, and hidden servers which communicated using public Tor relays, with some success. Our approach requires a "map" presenting with information of inter-domain routers for the Tor Entry Node and the victim. In our experiments, we did not use elaborate maps. We only considered result obtained from running *traceroute* between the targeted victim and its Tor Entry

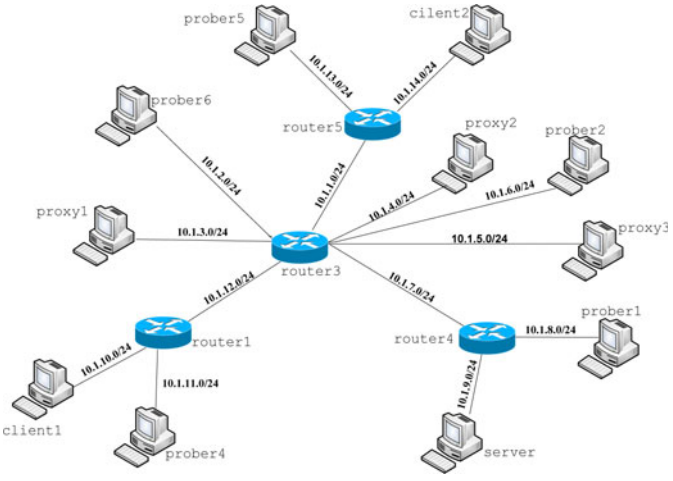


Fig. 1. DETER testbed network topology

Node. Moderate success rate is primarily due to a combination of our inadequate network vantage points and low end-to-end throughput offered by the Tor relays as compared to the available link capacities of routers and relays. Lastly, our accuracy was also affected by the presence of background cross traffic on regular network routers, resulting in higher false negatives when compared to the in-lab or DETERLAB experiments' results.

4 Experimental Evaluation

Our attack technique can be applied to low-latency anonymity systems that are based on *Onion Routing* [19]. Tor is a good example and chief among onion routing anonymizing based systems. In such systems, there seems to be a trade-off between anonymity guarantees and performance [28]. We show that in both controlled and real-world experiments, a well-provisioned adversary can expose the anonymity of Tor relays and end-points. This is performed by determining the available bandwidth fluctuations on a large number of relays and network routers.

4.1 Experiments Using the DETER Testbed

To determine the effectiveness of our attack, we used DETER [3] to emulate various network topologies. DETER is a network emulation system that offers commodity machines and switches connected with user specified topologies. Users can specify the operating systems, network connections, link speeds, routing protocols and other parameters.

Figure 1 depicts one of the topologies we used to validate our approach. In these experiments, we used Iperf [29] and LinkWidth to detect the fluctuation of the available link bandwidth on network routers along the path connecting the server to the actual client[5].

The host, marked as `server` in the topology figure (Figure 1), was the colluding web server. In addition, there were two client hosts, `client1` and `client2`. `client1` was the victim downloaded a large file from `server`, using HTTP; `client2` was idle.

With two clients on two separate branches, the available traffic fluctuation induced by `server` was observable only by one of the branches (the one leading to `client1`). The probing hosts on each of the subnets probe the intermediate links. To forward packets through `proxy1`, we used *squid* [33]. `client1` connected to `server` via `proxy1`. The `server` modulated the transmission rate to induce the necessary bandwidth fluctuation. The adversary probed for the bandwidth variation on the network elements in both branches using its probing hosts.

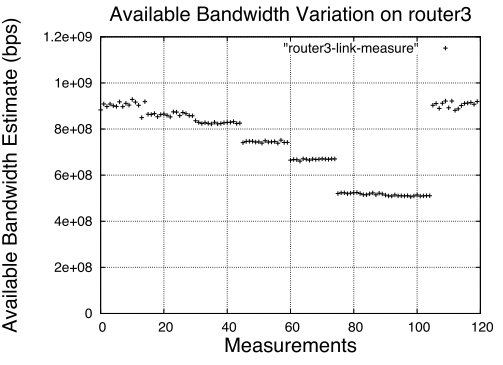


Fig. 2. The detected available bandwidth on the router connected to the victim `router3` drops uniformly as client traffic increases

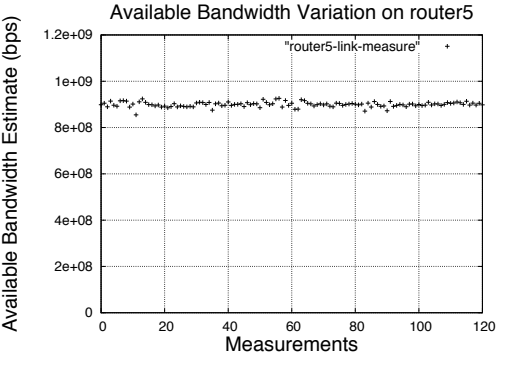


Fig. 3. We correctly measured the absence of consistent available bandwidth fluctuation on `router5` (not in the victim's path)

The reason we avoided installing an anonymizing system in this particular experiment is due to the poor QoS resulting from computational and network transport costs in systems like Tor. The motivation behind choosing *squid* was to demonstrate the effectiveness of our hypothesis; we assume that the relaying architecture of Onion Routing based systems would soon provide higher throughput rate like unencrypted *squid* proxy.

[5] Due to the way DETER emulates the requested link capacities and the use of traffic shaping, Iperf was more accurate than LinkWidth in estimating the bandwidth variations

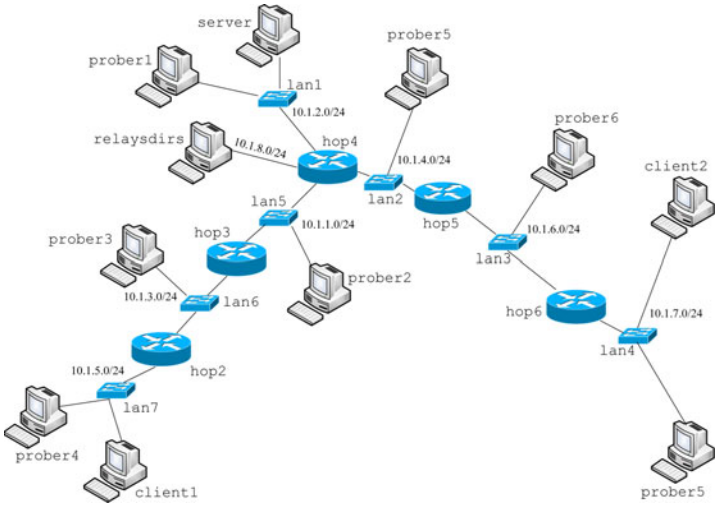


Fig. 4. In-Lab testbed network topology

Therefore, the goal of the experiment was to demonstrate that an adversary can observe these induced traffic fluctuations, provided the anonymizing service does not degrade client–server traffic throughput.

While the download was in progress, the server increased the transmission speed gradually by 50 Mbps, 100 Mbps, 200 Mbps, 300 Mbps and 500 Mbps, for every iteration of the experiment. The probing nodes (`probers`) measured the available bandwidth variation on both branches. The available link bandwidth fell steadily on all routers along the path carrying `client1`'s traffic. The probing of `router5` and `client2`, along the idle network branch, resulted in very high true negatives. For brevity, we only present the results obtained by probing `router3` and `router5`. The graphs representing these are presented in Figures 2 and 3. The graphs that show the fluctuations in available bandwidth for the rest of the hosts are presented in Appendix A.

The available bandwidth drops as the server increases its rate to the victim, and thus occupies greater share of the available bandwidth along the path via `router3`. Probing `router5` along the path connecting `client2` to `server` shows no significant fluctuation in available bandwidth. Overall, our experiments indicate that we can indeed detect small fluctuations by utilizing about 5–10% of the link bandwidth. Although this is a large value (approximately 50–100 Mbps), it is only a small fraction of the link bandwidth. This encouraged us to believe, that our technique will work well even in situations where only a small portion of the network link is being utilized by anonymous traffic.

We used a large file for our experiments. But we could have achieved the same fluctuation through multiple downloads of many small files. Through coordinated probing of the candidate links, momentary burst (due to small files being downloaded) can be easily detected. This is clearly evident from our results presented in Appendix A. The sudden fall in available link bandwidth from

approximately 100% (900 Mbps) to 90% (800 Mbps) within a short interval (few seconds) and and sudden rise later to 100%, in tandem to the induced traffic fluctuations, proves this. Further evidence of LinkWidth's effectiveness to detect small bandwidth fluctuations is presented in in our technical report [14].

4.2 In-Lab Experiments

To further support the DETER results, we performed the same experiments in an in-lab environment using commodity machines connected via a Gigabit switched network. Figure 4 depicts the in-lab network testbed topology used to demonstrate our technique. Again, the client `client1` is the victim and is connected to the `server` via a *squid* proxy installed on the host `relaysdirs`.

As before, the client downloaded a large file and the server varied the TCP throughput. The probing hosts measured the available bandwidth on the routers along both branches of the network - one leading to `client1`, which downloads the file, and the other leading to `client2`, which is not involved in the transfer.

Available bandwidth on all the routers along the path connecting `client1` to `server` fluctuated, as `server` varied the available TCP traffic to port 80, that it saw originating from the host `relaysdirs`. For brevity, we present graphically the results from probing `hop3` (Figure 5), along the path leading to the victim and `hop5` (Figure 6), leading to the idle node.

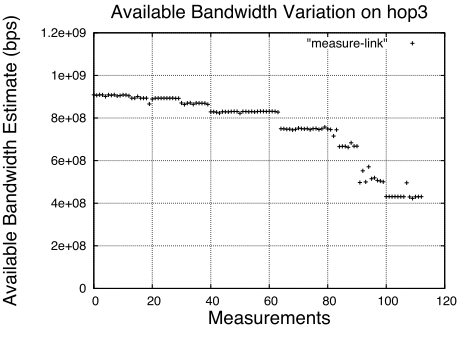


Fig. 5. Available bandwidth on `hop3` drops uniformly when we increase the traffic towards the victim

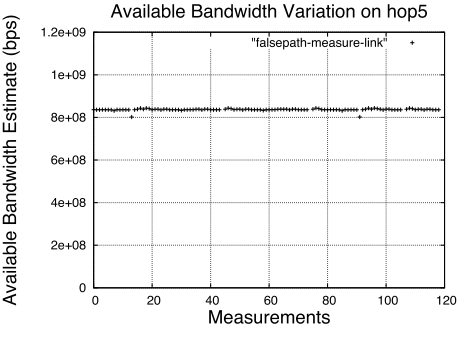


Fig. 6. There is no persistent available bandwidth fluctuation on `hop5` (unlike `hop3`, that is along path of the download traffic)

We observed reduction in available bandwidth as client-server traffic eventually occupies more bandwidth along the path via `hop3`. The available link bandwidth of `hop5` does not show any drastic change, as it was not along the actual download path. Probing router `hop3` and `client1` also showed similar bandwidth fluctuations while `hop6` and `client2` showed no definite fall in the available the link bandwidth; thus concurring with our idea of tracking link bandwidth fluctuation along the path leading up to the actual source of traffic. These results are presented in Appendix B.

4.3 Probing Tor Relays, Clients and Hidden Services

The validation of our technique in previous subsections, albeit on a controlled environment, encouraged us to believe that our technique might potentially be used to track induced available bandwidth on network routers connecting a Tor client to a Tor Entry Node. Therefore, we attempted to identify the Tor Onion Routers (ORs) participating in a real-world circuit. The server colluded with the adversary to induce fluctuations in the circuit throughput. This resulted in available bandwidth fluctuation on ORs and network routers connecting these ORs to Onion Proxies (OPs)[6]. *This experiment is elaborated in our previous paper* [13]. We concisely describe the experiment and results here. Figure 7 illustrates how the adversary probed the Tor relays involved in a circuit.

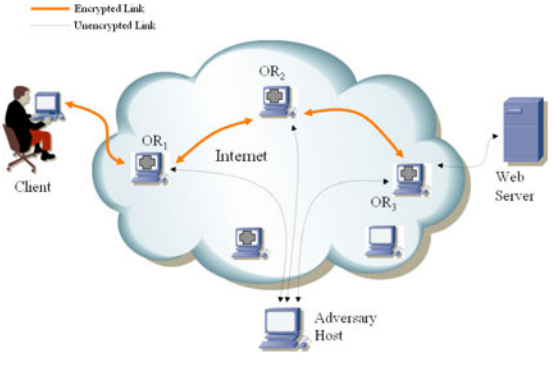


Fig. 7. Adversary probing the fluctuations in available bandwidth of ORs participating in a Tor circuit

The colluding web server transmitted a file which the client downloaded the file through a Tor circuit. During the download, the adversary used our traffic analysis technique to identify the victim relays participating in the circuit. While the server shaped the circuit's throughput to various values, the adversary measured the available bandwidth using LinkWidth. This process was repeated several times. In every iteration, the adversary changed the client's bandwidth share, increasing it each time by approximately 100 Kbps. The adversary detected decrease in measured available bandwidth that was reflected through increase in congestion and packet losses.

In our experiments, we successfully identified **all** the relay nodes in 11 out of the 50 circuits that we probed. For 14 circuits, we were able to identify only two of the three participating Tor relays. There were 12 circuits in which only one of the relays was detected. There were also 13 circuits that we are unable to detect any of the participating ORs. Finally, among all the 150 ORs probed there were 22 which filtered all probe traffic. A similar experiment was performed for obtaining

[6] Onion Proxy is the term used for Tor clients [15].

an estimate of the false positives. Initial observations yielded approximately 10% false positives. *However, on repeated iterations of the same experiment, we detected no false positives.* These very likely result from noises and errors in our measurement techniques resulting from lack of adequate network vantage hosts, background network cross-traffic and asymmetric network links.

Probe Set-up and Technique for Identifying Tor Clients: To determine the network identities of Tor clients involved in Tor circuits, we used the same setup as in Figure 7. However, in this experiment, the adversary probed routers on the network path connecting the Tor Entry Nodes to the Tor Clients. The client fetched a relatively large file from a colluding server, which shaped the bandwidth of the connection.

Table 1. Available-Bandwidth Fluctuation Detection in Links Connecting a Tor Client and its Entry Node

Circuit Number	Hops from Client –Entry Node Node	Correctly Detected Client–Entry Node Hops	Unresponsive Routers	Routers Not Reporting Enough Fluctuations	Success Rate
1	10	6	4	0	60.00%
2	15	4	0	0	26.67%
3	18	4	7	7	22.23%
4	18	5	8	5	27.78%
5	14	6	2	6	42.86%
6	14	9	1	4	64.30%
7	15	7	2	6	46.67%
8	14	7	2	5	50.00%
9	14	4	2	8	28.57%
10	15	6	4	5	40.00%

Lacking a "network map" that has link-level connectivity information of the Internet, we relied on `traceroute` information to discover the hops between the client and its Entry Node. However, in practice, an adversary equipped with AS-to-AS and link-level network connectivity maps, needs to probe only a relatively small set of links to determine which of them might be carrying the victim's traffic. Entry and exit, to and from an AS, is through a small number of *inter-domain* routers. The adversary can look up the AS location of the Tor relays from the inter-domain routing information. This will enable him to track the induced available bandwidth variation **only** on inter-domains routers and search for the AS that hosts the victim anonymous client. Nodes in all the probable ASes, with link speeds comparable to that of the inter-domain routers', could be used for this task. To obtain higher fine-grained network position, the attacker might have to track down to the end hosts. This step will require ISP router maps through services such as as *Rocketfuel* [5]. *Though seemingly an exponential*

search problem, this would be reasonably tractable provided the fluctuations could be detected with high confidence. Moreover, optimizing the adversary's search of links to probe is a different problem and we do not consider that aspect of the attack in this paper.

The results from probing the available link bandwidth variations on network routers connecting the clients to their respective Entry Nodes, is presented in Table 1. The accurate detection of bandwidth fluctuation was performed through the detection of packet loss changes. This loss is indication of decrease in available bandwidth, whenever the routers and Tor relays were probed using LinkWidth. Our technique used an un-cooperative method of available bandwidth or throughput estimation. Sometimes the probe traffic, being aggressive, prevented a Tor client, using TCP (which is "elastic" and non-aggressive), to utilize all of the allowed bandwidth. This lead to an "on-off" pattern in the the client's download. This is particularly true when the probes and the probed traffic traverse the same victim router.

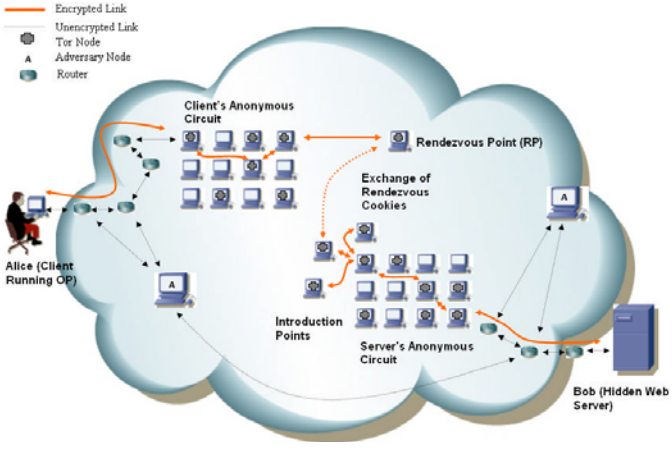


Fig. 8. The adversary measures the available bandwidth and thus detects any fluctuations in each link of the path leading to the victim

As a caveat, modifying the available throughput of the TCP connection through a certain repeating pattern (*e.g.,* 50 Kbps, 100 Kbps, 500 Kbps, 50 Kbps, 100 Kbps, 500 Kbps), would be akin to inducing a distinct "signal" or "watermark" in the client-server traffic. If very high true positives are assured, then this "signal" could be detected always and only on relays and routers through which the victim's traffic travels[7]. This can optimize the search for links to probe while determining the source of the anonymous traffic[8].

[7] Thereby also eliminating false positives and false negatives.

[8] Without such an optimization, the adversary might end up performing a depth-first search of large segments of the Internet rooted at the Tor entry node.

Probe Set-up and Technique for Identifying Tor Hidden Servers: To identify a Hidden Service, we used the setup depicted in Figure 8. Here the adversary probed the routers connecting Hidden Service to its Entry Node. Contrary to the previous cases, the available bandwidth fluctuation was induced by the client which is assumed to collude with the adversary. We relied solely on `traceroute` for determining which routers connect a Hidden Server to its corresponding Entry Node. Table 2 summarizes the results of this experiment:

Table 2. Available-Bandwidth Fluctuation Detection in Links Connecting a Hidden Server to Its Entry Nodes

Circuit Number	Hops from Hidden Server–Entry Node	Correctly Detected Hidden Server–Entry Node Hops	Unresponsive Routers	Routers Not Reporting Enough Fluctuations	Success Rate
1	13	4	2	7	30.70%
2	12	9	0	3	75.00%
3	11	7	1	3	63.64%
4	14	5	4	5	35.71%
5	12	9	0	3	75.00%
6	13	3	3	7	23.08%
7	16	5	4	7	31.25%
8	13	3	2	8	23.08%
9	17	4	1	12	23.53%
10	13	5	1	7	38.46%

In almost every circuit, there were some routers which filtered our probe packets. The rest of the routers were either detected correctly or not detected at all (*i.e.*, no false positives). This can be attributed to the lack of sufficient vantage points or to insufficient throughput achieved by the client in some cases (approximately 5–10 KBytes/sec). Despite of these practical problems, we were still able to trace the bandwidth fluctuations along the path (and hence the identity) of the Tor client and Hidden Server with high accuracy; over 60% and 75% in some of the circuits. The observed degradation in the client's performance whenever the adversary probed the candidate routers, are accepted as "available bandwidth fluctuations". Placing a large number of probing hosts at network vantage points would provide the adversary with better detection resolution and accuracy.

5 Issues, Discussion and Possible Counter-Measures

We initially tested our trace-back technique under various controlled environments. Our results indicate high true positives and almost zero false negatives. Small bandwidth variations, due to the introduction of a 50–100 Mbps TCP stream, were clearly discernible on a 1 Gbps link. This led us to believe that

small bandwidth fluctuations on *high-capacity links* can be detected provided there is low background cross traffic that may introduce false positives or false negatives in the measurements.

LinkWidth provides very accurate available link bandwidth estimation. As presented in our technical report [14], LinkWidth can accurately detect 1 Kbps fluctuation in available link bandwidth. Of course, this accuracy decreases when the variations decrease as a percentage of the overall link capacity. Small distortions, for instance 50 Kbps, on a 500 Kbps are easier to detect, than when they are on a 1 Gbps link.

Simple fluctuations on network links of the in-lab testbed could be detected within 15-20 seconds. The probing speeds were adjustable run-time parameters. Faster probing caused greater contention to the client-server traffic, thereby slightly decreasing the detection accuracy and granularity.

Having obtained high true positives under controlled environments, it seemed intuitive that an adversary could potentially detect available bandwidth fluctuation on an anonymizing proxy and its propagation to corresponding clients or servers via network routers. It is important that adversarial nodes are located at network vantage points where they can filter out traffic that causes unwanted distortion to the probes. It is also essential that a Tor client achieves high end-to-end throughput through Tor relays which is comparable to the installed link capacity of the network routers.

When applied to detect available link bandwidth variations on real Tor ORs, we were able to detect with some success, fluctuations on network routers connecting the client to its respective ORs. However, we restricted our selection of Tor relays within the US, to position our US-based probing host at a better network vantage point, when probing Tor relays. Probing nodes residing across trans-oceanic links seems infeasible and provided erratic results. Consequently, we were limited by the number of Exit Nodes within the US. Out of the approximately 150 exit relays at the time of our experiments less than 100 allow clients to setup their own circuits. Moreover, less than a fifth of these allow Hidden Servers to communicate with anonymous clients. This is mostly due to intermittent quality of service, node availability, and exit policies that restricted connectivity to a small set of permitted users. Probing Tor relays and network routers required considerably more measurements than the in-lab measurements (approximately 2-5 minutes per relay or router). High Internet cross-traffic and low Tor traffic necessitates longer probing and more measurements to avoid false positives and false negatives as much as possible.

In real world scenarios, there maybe various ways to entice a Tor client to connect to such malicious servers. Tempting commercials on a website, luring a Tor client to click on, could be one such tactic. This could download applications, like multiple Adobe Flash animations, on the client's host, resulting in a sudden change in his/her available network link bandwidth. An adversary running multiple co-ordinated probing hosts, probing suspected links, could detect such a sudden sharp change in available link bandwidths on all links connecting the anonymous party to its anonymizing service; thereby revealing its identity. The

adversary would require to own only a frame of a popular website, say a blog or an online forum, visited frequently by users who wish to stay anonymous.

Apart from the lack of accuracy in detecting small variations of available link bandwidth, Reardon and Goldberg have described why current Tor circuits offer low end-to-end throughput [28]. This is primarily because of multiplexing many TCP streams through a single Tor circuit connecting a Tor client to a relay or between the relays themselves, if such circuits already exist. Therefore, TCP congestion control and flow control mechanics affect the performance of all Tor circuits that are multiplexed over a single TCP connection. Packet losses and reordering affecting the *cells* of one Tor circuit reduced the overall performance of the TCP connection. Such losses cause the *cells* from unrelated Tor circuits to be delayed as well.

These inherent measurement limitations can be potentially leveraged to create countermeasures or even narrow the applicability of our attack. For instance, an anonymous client can utilize parallel connections in a round-robin fashion to access the same server. This would diffuse the ability of the server to generate detectable traffic variations: traffic spikes would be distributed across all the parallel connections. Likewise, traffic smoothing by anonymizing proxies is another potential countermeasure. Tor allows relay operators to use such techniques. Another option is to use shorter circuit lifetime. This would impose some time limitations on the duration of the communication path, making it harder for an adversary to completely trace the target through the anonymizing network. Anonymous connections using longer paths by employing more relays do not appear to make the attack appreciably more difficult. However, as discussed in [12], it can significantly affect the client's perception of the connection throughput.

6 Conclusions

We proposed a new traffic analysis technique that can expose the network identity of end-points (users) of low-latency network anonymity systems. Our technique involves an adversary who can probe links from many network vantage points using single-end controlled bandwidth estimation tools. In addition, the adversary employs a colluding server or is able to otherwise induce fluctuations in the victim's TCP connection. It is not essential to own such colluding servers: using carefully crafted online ads and pop-ups can be used judiciously to trick users to click on specific links and result in traffic spikes. Using the vantage points, the adversary measures the effects of this fluctuation as it "trickles" through the anonymizing relays and intermediate routers to the victim.

Our approach works well when the end-to-end throughput of the anonymizing network allows for bandwidth variations that can be detected by the vantage points. This motivated us to test our attack technique in real-world Tor circuits. Our experiments show that we were able to expose real-world Tor relays with a true positive rate of 49.3%. Furthermore, we can successfully traceback to the Tor clients and Hidden Servers by probing the network routers connecting them to their Entry Nodes. On average, we could correlate 49.5% of the network routers

to the victim Tor traffic that they carried. Further correlations were not always feasible due to bandwidth limitations imposed by relay operators and Tor's poor performance owing to circuit scheduling and management [28]. We believe that our work exposes a real weakness in proxy-based anonymity schemes. *This threat will become increasingly more apparent and accurate as future networks and hosts participate in higher end-to-end throughput circuits.*

References

1. AS Peering Analysis, http://www.netconfigs.com/general/anoverview.htm
2. CAIDA Router Measurements, http://www.caida.org/tools/taxonomy/routing.xml
3. DETER Network Security Testbed, https://www.isi.deterlab.net
4. I2P Anonymous Network, http://www.i2p2.de/
5. Rocketfuel: An ISP Topology Mapping Engine, http://www.cs.washington.edu/research/networking/rocketfuel/
6. The Internet Mapping Project, http://www.cheswick.com/ches/map/
7. Agrawal, D., Kesdogan, D.: Measuring Anonymity: The Disclosure Attack. IEEE Security & Privacy 1(6), 27–34 (2003)
8. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 245–257. Springer, Heidelberg (2001)
9. Bauer, K., McCoy, D., Grunwald, D., Kohno, T., Sicker, D.: Low-resource routing attacks against tor. In: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society (WPES), pp. 11–20 (2007)
10. Borders, K., Prakash, A.: Web Tap: Detecting Covert Web Traffic. In: Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS), October 2004, pp. 110–120 (2004)
11. Burch, H., Cheswick, B.: Tracing Anonymous Packets to Their Approximate Source. In: Proceedings of the 14th USENIX Conference on System Administration (LISA), December 2000, pp. 319–328 (2000)
12. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Approximating a Global Passive Adversary Against Tor. Computer Science Department Technical Report (CUCS Tech Report) CUCS-038-08, Columbia University (August 2008)
13. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Identifying Proxy Nodes in a Tor Anonymization Circuit. In: Proceedings of the 2nd Workshop on Security and Privacy in Telecommunications and Information Systems (SePTIS), December 2008, pp. 633–639 (2008)
14. Chakravarty, S., Stavrou, A., Keromytis, A.D.: LinkWidth: A Method to Measure Link Capacity and Available Bandwidth using Single-End Probes. Computer Science Department Technical Report (CUCS Tech Report) CUCS-002-08, Columbia University (January 2008)
15. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation Onion Router. In: Proceedings of the 13th USENIX Security Symposium (USENIX Security), August 2004, pp. 303–319 (2004)
16. Evans, N., Dingledine, R., Grothoff, C.: A practical congestion attack on tor using long paths. In: Proceedings of the 18th USENIX Security Symposium (USENIX Security), August 2009, pp. 33–50 (2009), http://freehaven.net/anonbib/papers/congestion-longpaths.pdf

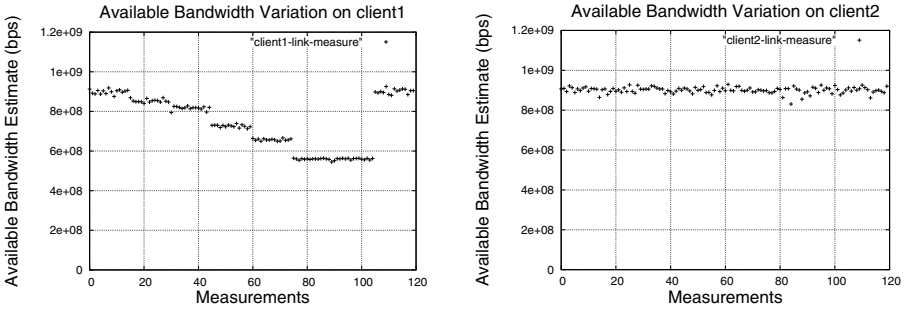
17. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: A type iii anonymous remailer, http://mixminion.net/
18. Gerla, M., Sanadidi, M.Y., Wang, R., Zanella, A.: TCP Westwood: Congestion Window Control Using Bandwidth Estimation. In: Proceedings of IEEE Global Communications Conference (Globecomm), November 2001, vol. 3, pp. 1698–1702 (2001)
19. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding Routing Information. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 137–150. Springer, Heidelberg (1996)
20. Hopper, N., Vasserman, E.Y., Chan-Tin, E.: How Much Anonymity does Network Latency Leak? In: Proceedings of ACM Conference on Computer and Communications Security (CCS), October 2007, pp. 82–91 (2007)
21. Huang, D., Agarwal, U.: Countering Repacketization Watermarking Attacks on Tor Network. In: Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS 2010), Beijing, China (June 2010)
22. JAP, http://anon.inf.tu-dresden.de/
23. Madhyastha, H.V., Isdal, T., Piatek, M., Dixon, C., Anderson, T.E., Krishnamurthy, A., Venkataramani, A.: iplane: An information plane for distributed services. In: Proceedings of 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), November 2006, pp. 367–380 (2006)
24. Murdoch, S.J.: Hot or not: Revealing hidden services by their clock skew. In: Proceedings of ACM Conference on Computer and Communications Security (CCS), October 2006, pp. 27–36 (2006)
25. Murdoch, S.J., Danezis, G.: Low-Cost Traffic Analysis of Tor. In: Proceedings of IEEE Symposium on Security and Privacy (IEEE S and P), May 2005, pp. 183–195 (2005)
26. Raymond, J.-F.: Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems. In: Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, pp. 10–29 (2001)
27. Dingeldine, R., Edman, M., Lewman, A., Mathewson, N., Murdoch, S., Palfrader, P., Perry, M., Syverson, P.: Tor: anonymity online, https://www.torproject.org/
28. Reardon, J., Goldberg, I.: Improving tor using a tcp-over-dtls tunnel. In: Proceedings of 18th USENIX Security Symposium 2009 USENIX Security (August 2009)
29. Tirumala, A., Qin, F., Dugan, J., Feguson, J., Gibbs, K.: IPERF (1997), http://dast.nlanr.net/projects/Iperf/
30. Pappas, V., Athanasopoulos, E., Ioannidis, S., Markatos, E.P.: Compromising Anonymity Using Packet Spinning. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 161–174. Springer, Heidelberg (2008)
31. Wang, X., Chen, S., Jajodia, S.: Network flow watermarking attack on low-latency anonymous communication systems. In: Proceedings of IEEE Symposium on Security and Privacy (IEEE S and P), pp. 116–130 (2007)
32. Wang, X., Reeves, D.S.: Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), October 2003, pp. 20–29 (2003)
33. Wessels, D., Rousskov, A., Nordstrom, H., Chadd, A., Jeffries, A.: Squid, http://www.squid-cache.org/
34. Wright, M.K., Adler, M., Levine, B.N., Shields, C.: An analysis of the degradation of anonymous protocols. In: Proceedings of the Network and Distributed Security Symposium, NDSS (2002)

35. Fu, X., Ling, Z.: One cell is enough to break tor's anonymity. In: Proceedings of Black Hat Technical Security Conference, February 2009, pp. 578–589 (2009)
36. Zander, S., Murdoch, S.: An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In: Proceedings of 17th USENIX Security Symposium (USENIX Security), San Jose, USA, July 2008, pp. 211–225 (2008)

Appendix

A Results from Probing Host/Routers on DETER Testbed

This section presents the results omitted in subsection 4.1. The graph in Figure A exhibits the results from probing the network interface of `client1`. As expected, we observe a very similar visible available bandwidth fluctuation pattern as we saw in Figure 5 for `router3`. Also evident from results in Figure A, we don't observe any obvious induced available bandwidth variations when the `client2` is probed from `prober5`.



(a) Available bandwidth on `client1` varies much like `router1` and `router3`, as the download rate changes

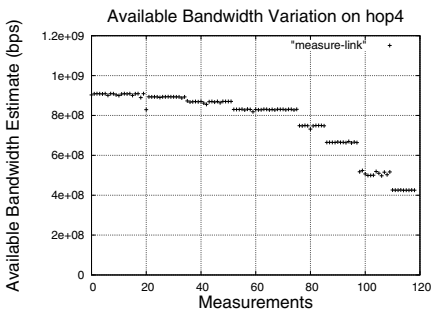
(b) Available bandwidth on `client2` doesn't vary like that on network entities along the actual download path

Fig. 9. Available Bandwidth Variation on `client1` and `client2` of DETERLAB testbed

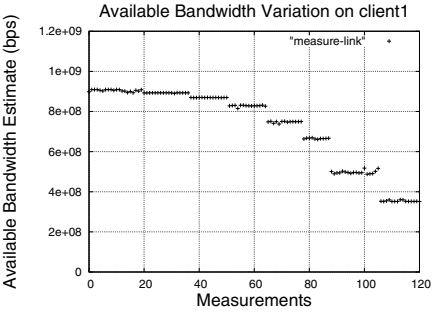
B Results from Probing Host/Routers on Lab Test-Bed

All of the results obtained from probing for available bandwidth variation of the network entities were not presented in subsection 4.2. For the sake of completeness, we present the remainder of the results here in Figures 10 and 11.

The results in Figures B and B are along the path which carries the download traffic. The plots in Figures B and B are for hosts that do not carry the download traffic (hence no observed variations in available link bandwidth).

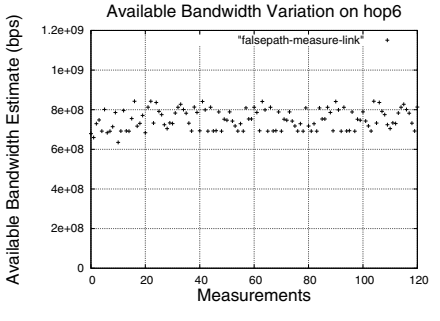


(a) Available bandwidth variation on `hop4` similar to that of `hop3`, along the actual download path

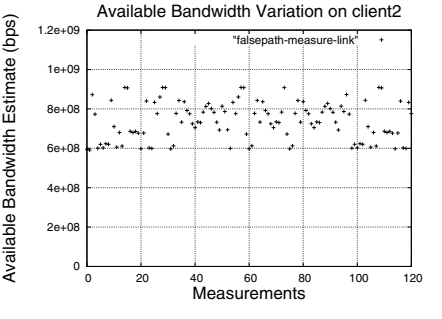


(b) Available bandwidth on `client1` varies much like `hop3` and `hop4`,as the download rate changes

Fig. 10. Available Bandwidth on Routers and End Hosts of the In-Lab Network Along the Download Path



(a) No uniform available bandwidth variation seen in `hop6`, similar to what we see in `hop3`



(b) Absense of the uniform available bandwidth variation that is observed on the actual source (`client1`)

Fig. 11. Available Bandwidth on Routers and End Hosts of the In-Lab Network Not Along the Download Path

A Hierarchical Adaptive Probabilistic Approach for Zero Hour Phish Detection

Guang Xiang, Bryan A. Pendleton, Jason Hong, and Carolyn P. Rose

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract. Phishing attacks are a significant threat to users of the Internet, causing tremendous economic loss every year. In combating phish, industry relies heavily on manual verification to achieve a low false positive rate, which, however, tends to be slow in responding to the huge volume of unique phishing URLs created by toolkits. Our goal here is to combine the best aspects of human verified blacklists and heuristic-based methods, i.e., the low false positive rate of the former and the broad and fast coverage of the latter. To this end, we present the design and evaluation of a hierarchical blacklist-enhanced phish detection framework. The key insight behind our detection algorithm is to leverage existing human-verified blacklists and apply the shingling technique, a popular near-duplicate detection algorithm used by search engines, to detect phish in a probabilistic fashion with very high accuracy. To achieve an extremely low false positive rate, we use a filtering module in our layered system, harnessing the power of search engines via information retrieval techniques to correct false positives. Comprehensive experiments over a diverse spectrum of data sources show that our method achieves 0% false positive rate (FP) with a true positive rate (TP) of 67.15% using search-oriented filtering, and 0.03% FP and 73.53% TP without the filtering module. With incremental model building capability via a sliding window mechanism, our approach is able to adapt quickly to new phishing variants, and is thus more responsive to the evolving attacks.

1 Introduction

Phishing is a social engineering attack, in which criminals build replicas of target websites and lure unsuspecting victims to disclose their sensitive information like passwords, personal identification numbers (PINs), etc. Exact numbers of direct damages done by phishing are hard to assess, in large part due to lack of data from organizations hit by phishing attacks. Estimates have ranged from a low of $61 million [15] to a high of $3.2 billion [1]. A significant proportion of those losses were caused by one particularly infamous group, known as the "rock phish gang", which uses phish toolkits to create a large number of unique phishing URLs, putting additional pressure on the timeliness and accuracy of blacklist-based anti-phishing techniques.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 268–285, 2010.

Generally, phish detection methods fall into two categories, i.e., those that perform URL matching via human verified blacklists and those that make use of heuristics via machine learning (ML) techniques. While the former has a very low false positive rate, human-verified blacklists do not generalize well to future unseen cases. For example, Sheng et al [21] showed that zero hour protection offered by major blacklist-based toolbars only has a true positive rate (TP) between 15% and 40%. Furthermore, human-verified blacklists can be slow to respond to new phishing attacks, and updating blacklists usually involves enormous human effort. For example, Phishtank [2] statistics in March 2009 show that it took on average 10 hours to verify that a URL was a phish. Finally, human-verified blacklists can be easily overwhelmed by automatically generated URLs. On the other hand, heuristic-based approaches enjoy the flexibility of being able to recognize new phish, but often lead to a relatively higher false positive rate. Concerns over liability for false positives have been a major barrier to deploying these technologies [20]. To underscore this point, Sheng et al [21] evaluated eight popular toolbars including Microsoft Internet Explorer, Firefox, Google Chrome, Norton 360, etc., all of which employ some level of human verification to achieve an extremely low FP in spite of the amount of human labor required, again primarily due to concerns over liability for false positives.

The goal of our work is to combine the best aspects of human verified blacklists and heuristics-based methods, and develop a reliable and robust method that is able to adaptively generalize to new attacks with reasonable TP while maintaining a close to zero FP. Our approach exploits the fact that a large number of current phishing attacks are created with toolkits, which tend to have a high similarity in terms of content. Our detection engine analyzes the content of phishing webpages on manually-verified URL blacklists via n-grams, and employs the shingling technique to identify near-duplicate phish in a probabilistic fashion. We also use a filtering module, which uses information retrieval (IR) techniques querying search engines to further scrutinize the legitimacy of a potential phish in an effort to control false positives. Our whole system is constantly updated by a sliding window upon the arrival of new phishing data, and is thus capable of adapting quickly to new phishing variants, while still maintaining a reasonable level of runtime performance. Under the optimal experimental setup, our method achieves a TP of 67.15% with 0% FP using search oriented filtering, and a TP of 73.53% and a FP of 0.03% without the filtering module, much better than blacklist-based methods in TP while comparable in FP. For applications like anti-phishing where FP is of paramount importance, a slightly lower TP is acceptable. Furthermore, we do not expect our approach to be used alone, but rather reside in the first part of a pipeline augmenting the existing system such as the commercial blacklists, thus fabricating a superior integrated solution.

We do not claim that our approach will solve the phishing problem. Rather, our specific claim is that we can augment existing blacklists in a very conservative manner using probabilistic techniques, with a very low FP, if not zero, and a reasonably good TP. Capable of identifying a fair amount of phishing attacks with no sacrifice on FP and considerably reducing the human effort involved

in manual verification, our approach significantly complements the prevalent blacklist-based methods, leveraging the manual labor that is already being used in verifying phishing sites. The major contributions of this paper are three fold.

1. We present the design of a novel hierarchical, content-based approach that leverages existing human-verified blacklists, by making use of shingling and information retrieval techniques to detect phish.
2. We demonstrate that with incremental model building via a sliding window mechanism, our approach is able to adapt quickly to the constantly evolving zero-hour phish attacks. Also, we only need the most recent 30 days' worth of data to achieve the same TP as using two months' worth of data, thus balancing accuracy with runtime efficiency.
3. By harnessing URL blacklists in a probabilistic fashion, we are able to leverage our approach to improve the coverage and timeliness of human-verified blacklists using considerably less human effort than existing techniques, without having to sacrifice the false positive rate. With only two weeks' worth of phish, our method achieves a TP of 65.02% with 0% FP using search oriented-filtering, and a TP of 71.23% and a FP of 0.03% without filtering.

2 Related Work

2.1 Methods for Automatic Phish Detection

A variety of techniques have been proposed for automatically detecting phishing web pages, and we will introduce some representative work in this section.

One camp exploits URL signatures to detect phish. Garera et al [13] identified a set of fine-grained heuristics from URLs, and combined them with other features to detect phish. Applying a logistic regression model on 18 features yielded an average TP of 95.8% and FP of 1.2% over a repository of 2508 URLs. Though interesting, this method has high variance in that URLs could be manipulated with little cost, causing the heuristics to fail.

Researchers have also devised a number of phish heuristics examining the content of web pages. Abu-Nimeh et al [8] adopted the bag-of-words strategy and used a list of words frequently found on phishing sites as features to detect phish, which is not expressive and easy to defeat by attackers. In [16], Ludl et al came up with a total of 18 properties solely based on the HTML and URL. The J48 decision tree algorithm was applied on these features and achieved a TP of 83.09% and a FP of 0.43% over a corpus with 4149 good pages and 680 phishing pages. However, heuristics purely based on DOM and URL are rather limited and may fail in capturing artfully designed phishing patterns. Zhang et al [23] proposed CANTINA, a content-based method using a linear classifier on top of eight features, achieving 89% TP and 1% FP on 100 phishing URLs and 100 legitimate URLs.

Another line of research focuses on discovering the intended phish brand to catch phish. Pan et al [18] proposed a method to extract the webpage identity from key parts of the HTML via the χ^2 test, and compiled a list of features based

upon the extracted identity. Trained with support vector machines (SVM), their features had an average FP of about 12%. However, its assumption that the distribution of the identity words usually deviates from that of the ordinary words is questionable, which is indicated by their high false positive rate. Even in DOM objects, the most frequent term often does not coincide with the web identity. More recently, Xiang et al [22] proposed a hybrid detection model that recognizes phish by discovering the inconsistency between a webpage's true identity and its claimed identity via search engine and information extraction techniques. Their full integrated system achieved a TP of 90.06% and a FP of 1.95%.

Though the work in [23] [22] also involve search engines, in this paper, we only resort to search engines in a postprocessing step to filter potential false positives in this paper while our core technique is the detection algorithm that exploits semantic similarity among the phishing attacks via the shingling technique. The only possible false positives generated in the detection phase are those well-known websites targeted by phishers, which guarantees the efficacy of our searching-based FP filtering method.

In addition to the past work above, anti-phishing toolbars are also available, many of which exploit human-verified blacklists to assure close-to-zero FP, such as NetCraft, Firefox 3, McAfee SiteAdvisor, etc.

Our goal in this paper is subtly different from the research above, in that we want to see how high our TP can be while maintaining close to 0% FP. As we noted in the introduction, industry has not adopted many of those heuristics above due to concerns about poor user experience for false positives as well as reasons of liability. Thus, our work here deliberately takes a conservative approach, though as we will show, we still get a reasonably good TP.

2.2 Toolkits for Creating Phishing Sites

In recent years, an increasingly large number of phishing webpages were automatically created by toolkits, which substantially increases the scale of attacks that criminals can attempt, while also countering current human-verified blacklists. For example, Cova et al [11] identified 584 phishing kits during a period of two months starting in April 2008, all of which were written in PHP. An analysis of rock-phishing sites by Moore et al [17] from February to April in 2007 reveals that 52.6% of all Phishtank reports were rock phish. One key observation of the rock phish is that their content is highly similar due to the way they are created, which is the property that our framework is based on. It is possible that criminals may modify their toolkits to include randomization to circumvent our detection mechanisms, and we discuss this issue towards the end of this paper.

3 A Multi-layered Phish Detection Algorithm

3.1 System Architecture

The overall architecture of our framework is shown in Fig.1. The first stage of processing involves filtering using domain whitelists, directly passing known

benign webpages. The detection engine employs a technique based on shingling to classify the remaining webpages, forwarding potential phish to the FP filter for further examination, which interacts with search engines to correct false positives. New phish from blacklists are added into our training set via a sliding window to update the detection model with the latest phishing patterns.

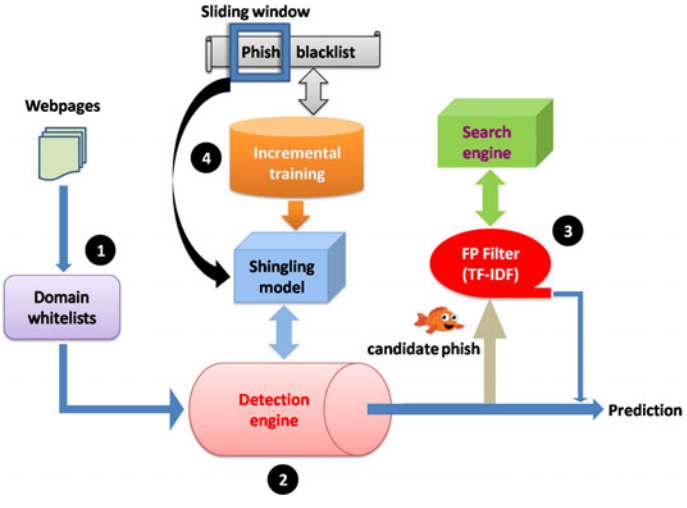


Fig. 1. System architecture. An incoming webpage is first checked against a small domain whitelist (1). If the page is not on the whitelist, our detection engine (2) compares the content of the webpage against the content of existing phish using the shingling algorithm. If a page is flagged as a potential phish, we check for false positives, resorting to search engines (3) if needed for additional verification. We use a sliding window (4) in the back-end to incrementally building the machine learning model as new phishing signatures arrive.

3.2 Shingling-Based Probabilistic Matching

The essence of our detection algorithm is to do "soft" matching of a given web page against known phishing pages. The intuition here is that many phishing web pages are created by toolkits, and thus have many semantic similarities in terms of page content. Our detection method manipulates this semantic uniformity via soft matching, which allows more flexibility than the rigid URL matching adopted by major blacklist-based methods. Our early evaluations using exact matching with hash codes of page content turned out to be reasonably effective, but also brittle and easy to defeat. As such, we want to make our system robust to simple changes, thus raising the bar for criminals.

Shingling [10], a technique for identifying duplicate documents, examines the webpage content on a finer-grained level via the notion of n-gram, and measures the inter-page similarity based on these basic units. N-grams are subsequences of

n contiguous tokens. For example, suppose we have sample text ***connect with the eBay community***. This text has 3-grams {***connect with the, with the eBay, the eBay community***}. Shingling employs a metric named *resemblance* to calculate the percent of common n-grams between two webpages. More formally, let q and d represent a webpage being examined and a phishing page in the blacklist respectively. Let D represent the set of all training phish, and $S(p)$ denote the set of unique n-grams in p. The similarity metric *resemblance* $r(q,d)$ is then defined as $r(q,d) = |S(q) \cap S(d)|/|S(q) \cup S(d)|$. Our soft matching approach first generates the set of n-grams for each $d \in D$. We then compute $r(q,d)$ $\forall d \in D$ for a query page q, and fire an alarm whenever $r(q,d)$ exceeds a threshold t. We choose the optimal t via cross validation.

3.3 Search Engine Enhanced Filtering

As we will show later on in the evaluation, shingling is effective in comparing a given web page against known phish. However, a potential problem is with false positives. More specifically, phishing web pages usually imitate legitimate web pages, which means that if there are no safeguards in place, shingling by itself is likely to label those target legitimate cases as phish as well. To solve this problem, we propose a filtering algorithm leveraging the power of search engines via information retrieval techniques. This module, based on one heuristic in CANTINA [23], compensates for the incompleteness of domain whitelists, and is able to minimize FP even for less popular phishing target sites.

Our filtering module is triggered when the detection engine recognizes a candidate phish, and works by executing in Google queries composed of K top keywords chosen from the page content plus the webpage domain keyword [1] and examining the presence of the page domain in the top N search results. The final prediction is restored to "legitimate" if the top N entries subsume the page domain, and thus we no longer incorrectly label such sites as phish. The validity of this filtering algorithm is partially attributed to the fact that legitimate websites are very likely to be indexed by major search engines, while phishing sites are not, due to their short-lived nature and few in-coming links.

We currently use $K = 5$, $N = 30$ according to the tuning result in [23] [22]. Candidate query terms on the page are ranked by the TF-IDF scoring function widely used in IR, which selects the terms that are most representative of the webpage content. The rationale is that search engines use TF-IDF when they match queries to documents in such a way that terms with high TF-IDF scores are the ones that have more influence over retrieval and ranking of documents.

3.4 Incremental Model Building via Sliding Window

To take the latest phishing signatures into our database and to improve the runtime performance of our whole system, we utilize a sliding window of the

[1] The domain keyword is the segment in the domain representing the brand name, which is usually the non-country code second-level domain or the third-level domain.

most recent phish from phish blacklists and incrementally build the detection model with those phishing web sites. In our evaluation, we show that discarding older data as the sliding window moves actually has little impact on accuracy.

Furthermore, a positive side effect of using a sliding window is that the time complexity of shingling is reduced from $O(|D|)$ to $O(|D_{win}|)$, where D_{win} represents all phishing data covered by the current sliding window win. Asymptotically, $|D_{win}|$ can be deemed as a large constant, and in light of this shrunk magnitude, we refrain from trading accuracy in exchange of speed via approximated algorithms as used in many applications [14]. For example, this sliding window could reduce a year's worth of phish to just a month's worth, achieving $\times 12$ runtime speedup without significantly sacrificing detection performance.

4 Experiment

4.1 Domain Whitelists

An enormous percentage of phishing frauds target well-known financial entities like eBay, Paypal, etc., by imitating their sites, and it is of practical value to pass those legitimate websites without feeding them to our detection engine. To reduce false positives and improve runtime performance, we quickly eliminate these known good sites through a whitelist. specifically, we collected known good domains from two sources. Google safe browsing provides a publicly-available database [3] with legitimate domains, and we obtained a total of 2758 unique domains from this whitelist after duplicate removal. Millersmiles [4] maintains an archive of the most common spam targets such as ebay, and we extracted 428 unique domains out of 732 entries after mapping organization names to domains and removing duplicates. In total, we had 3069 unique domains in our whitelist.

4.2 Webpage Corpus

Phishing sites are usually ephemeral, and most pages do not last more than a few days typically because they are taken down by the attackers themselves to avoid tracking, or taken down by legitimate authorities [21]. To study our approach over a larger corpus, we downloaded phishing pages when they were still alive and ran experiment offline. Our downloader employed Internet Explorer to render the webpages and execute Javascript, so that the DOM of the downloaded copy truly corresponds to the page content and thus gets around phishing obfuscations.

Our collection consists of phishing cases from PhishTank, and good webpages from seven sources. To eliminate the influence of language heterogeneity on our content-based methods, we only downloaded English webpages.

For phishing instances, we used the verified phishing URLs from the phish feed of Phishtank [5], a large community-based anti-phishing service with 38,324 active accounts and 527,930 verified phish [2] by the end of March 2010. We started downloading the feed in late February of 2009 and collected a total of 1175 phishing webpages from February 27, 2009 to April 2, 2009. All seven

legitimate corpus were downloaded after April 2, the details of which are given in Table 1. Note that the open directory project is the most comprehensive human-edited directory of the Web maintained by a vast community of volunteers, and by using this corpus, we want to verify that our algorithm achieves a very low FP on the low-profile and less popular sites.

Table 1. Legitimate collection with a total of 3336 web pages

Source	Size	Crawling Method
Top 100 English sites from Alexa.com	958	Crawling homepages to a limited depth
Misc login pages	831	Using Google's "inurl" operator and searching for keywords like "signin"
3Sharp [19]	87	Downloading good webpages that still existed at the time of downloading
Generic bank category [6] on Yahoo directory	878	Crawling the bank homepages for a varying number of steps within the same domains
Other categories of Yahoo directory	330	Same as the generic bank category
The most common phishing targets	69	Saving login pages of those sites
The open directory project [7]	183	Downloading "least popular" pages with zero pagerank

4.3 Test Methodology

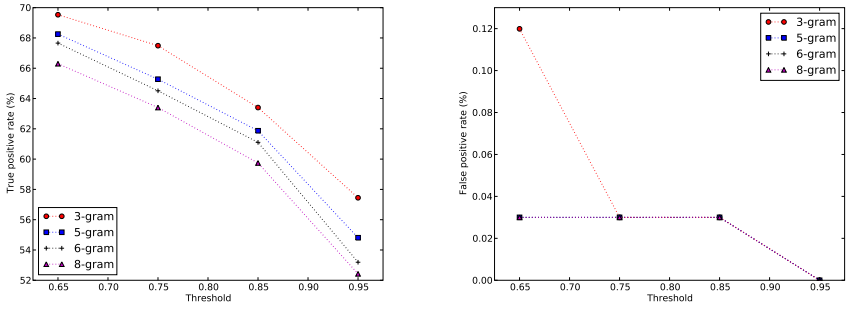
For notational convenience, we define in Table 2 the free variables in our context. Our experiment here focused on tuning these variables to optimize our results. To simulate a more realistic scenario, we processed data in chronological order in all of our experiments. In assessing TP, we move the sliding window of length L step by step along the time line and apply our detection algorithm to the webpages at each time point T_i using a shingling model built on the phishing data with time labels falling in window $[T_{i-L}, T_{i-1}]$. The FP is tested in a slightly different manner. In [12], Fetterly et al discovered through large-scale web crawling that webpage content was fairly stable over time, and based on that finding, we did not download the same set of legitimate pages at each time point but rather downloaded only once the whole set at a time later than all the phishing timestamps. Sliding windows of different sizes L are used similarly. Under all cases, four whitelist combinations are exercised with our detection algorithm, i.e., millersmiles, Google, none, and both whitelists.

Table 2. Definition of symbols

Variable	Explanation	Variable	Explanation
G	granularity of time	L	sliding window length
W	whitelist	n	n-gram
r	resemblance	t	resemblance threshold

4.4 Experimental Results

Shingling Parameter Tuning. Figure 2 shows the validation performance under different values for n and t. For all n-grams in the evaluation, the TP monotonically decreased as we raised the resemblance bar higher. With a resemblance of 65%, shingling achieved over 66% TP under all shingle lengths, manifesting the considerable similarity in content among phish due to rock phish. Although FP worsens as t and n decrease, we still stick to $n = 3, t = 0.65$ in the remaining evaluation of our experiment, hoping for the best TP performance and counting on the TF-IDF filtering algorithm to control false positives. The tuning results under all other configurations of G, L and W exhibit the same pattern, and we do not report them here.



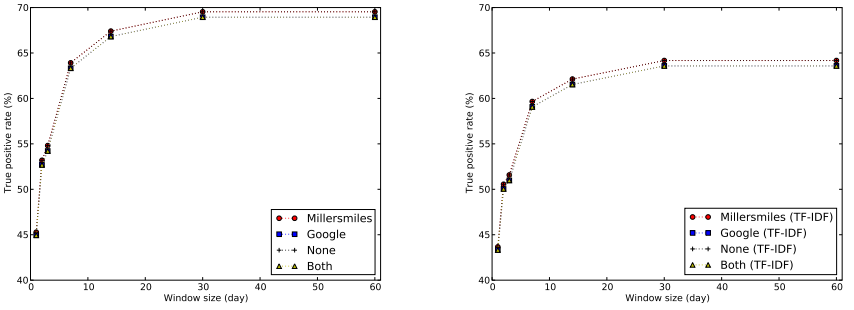
(a) TP under various r thresholds (b) FP under various r thresholds

Fig. 2. Shingling parameter tuning ($L = 60$, $G =$ day, $W =$ millersmiles, no TF-IDF FP filtering). A tradeoff between TF and FP is an important factor in choosing t in the final model. As t is increased, the rate of detection drops and FP picks up. TP tops at 69.53% and FP reaches a culmination of 0.1199% under $n = 3, t = 0.65$. The other three FP curves $n = 5, 6, 8$ perfectly coincide.

Evaluation of True Positive Rate. Figure 3 suggests that even with only one day's worth of training phish, our algorithm is able to detect around 45% phishing attacks, demonstrating the efficacy of our method and also proving the conjecture that mainstream phishing attacks are created by toolkits.

Another finding is that when using search engines to filter false positives (the right plot in Fig.3 and Fig.4), TP dropped as a side effect. An explanation is that some phishing URLs (2%/1% with a 1-day/1-hour sliding window) are actually returned among the top 30 entries when querying TF-IDF ranked terms plus the domain keyword on Google and are thus mistakenly filtered as legitimate.

Real-time application of our algorithm does not suffer from this false filtering problem as much as observed in our offline experiment. A semi-formal explanation for this finding has two main points. First, when a new phish just comes out of attackers' workshop, few, if any, links point to that phishing site. As such, search engines are unlikely to return its domain as a top result; second, search



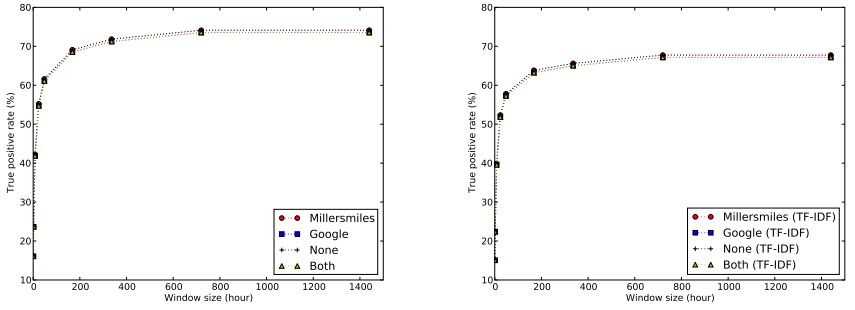
(a) TP without TF-IDF filter (b) TP with TF-IDF filter

Fig. 3. TP under various L (G = day) and W. Our approach achieves about 45% (no FP filtering) and 43% (with FP filtering) TP in all cases with only 1 day's worth of training phish, and around 69% (no FP filtering) and 64% TP (with FP filtering) with a 60-day window. FP filtering hurts TP, and a whitelist with only popular phishing targets beats a more comprehensive whitelist.

engines might index the phish as time progresses when more links out in the web begin referring to it, however, the phish may have already become unavailable due to the short-lived nature of phishing activity and no harm will be done to the users even if it is incorrectly passed as a good page. The usefulness of this FP filtering module will become more evident when we embark on the analysis of FP in the following section.

Figures 3 and 4 suggest that the TPs under millersmiles whitelist are universally better than those under Google whitelist. Examining both whitelists reveals that millersmiles only contains a core group of the most spammed domains while the Google whitelist has many more less popular domains. None of the phishing domains in our corpus appear in the millersmiles whitelist, however, some do show up in the Google whitelist, among which is "free.fr", occurring 6 times in our phishing set. Those phish were thus erroneously filtered, lowering the TP inadvertently. This observation delivers a message about the use of domain whitelists, i.e., the quality of whitelists does impact TP and the optimal usage is to adopt a small core whitelist covering a group of popular spam target sites. Our detection method performed convincingly better with respect to TP when the model is iteratively built on a hourly basis.

Evaluation of False Positive Rate. Figure 5 shows the FPs under different sliding window sizes and whitelists with no TF-IDF filtering. All four curves in both plots start with zero FPs, when L is minimum, and gradually escalate as more training phish are added to the model. Domain whitelists prove to be effective in suppressing false positives, with FPs of 0.1199%, 0.06%, 0.5396%, 0.03% for millersmiles, Google, none and both whitelists under both a 60-day window (left) and a 1440-hour window (right). With TF-IDF filtering, FPs are all zero under all circumstances, and we do not explicitly show the plots here.

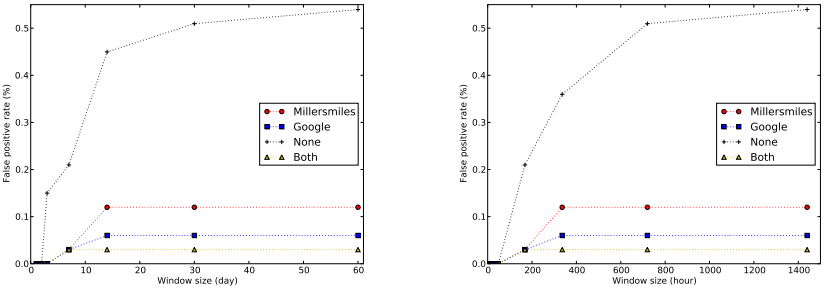


(a) TP without TF-IDF filter (b) TP with TF-IDF filter

Fig. 4. TP under various L (G = hour) and W. Under all whitelists, TP bottoms around 16% in all cases with a 1-hour window and peaks around 74% with a 1440-hour window without FP filtering; with FP filtering, TP bottoms around 15% with a 1-hour window and peaks around 67% with a 1440-hour window.

Granularity of Time Unit for Window Size. A comparison of TPs with day and hour based L (Table 4 in the appendix) shows that under sliding windows of identical time length, hour-level incremental model building outperformed day-level building, indicating the superior responsiveness of hourly updating. The largest gaps occurred at a window length of 1 day (24 hours), amounting to TPs of 9.95%, 9.78%, 9.95%, 9.78% with no FP filtering and 8.68%, 8.51%, 8.68%, 8.51% with FP filtering under four whitelist configurations. This disparity gradually diminished as L increased, which is reasonable in that as more and more phish are absorbed into the training set by the growing window, the tiny amount of shift in time relative to the window size no longer has as large of an impact as before. Surprisingly, simply with a 24-hour window, our algorithm was able to achieve over 50% TP under all whitelists and filtering setups.

As expected, the FPs under two time units in Table 5 in the appendix are identical except for one cell, since all legitimate pages in our web collection were downloaded after the phishing ones and regardless of time measurement (day or hour), the sliding window with the same length in terms of time actually covered roughly the same set of training phish. Interestingly, the FP filtering module successfully removed all the false positives, leading to zero FP under all experiment settings, at the cost of slight degradation on TP. Note that the evaluation of FP in our experiment is sound and thorough partially in that our legitimate corpus contains a diverse variety of data including those categories that are the worst case scenarios for phish detection. As a result, the experimental result offers conservative statistics that are more meaningful to the actual adoption and deployment of our system. As suggested by the statistics in Table 4 and Table 5, another feature of our system is that it offers an adjustable range of performance depending on a user or provider's willingness to accept false positives.



(a) FP vs window size (in number of days) (b) FP vs window size (in number of hours)

Fig. 5. FP under various L and W with no TF-IDF filtering. Under all whitelists, FP escalates with the growth of the sliding window size. FPs are zero when using TF-IDF to filter false positives under all settings and are not plotted here.

Evaluation Against Toolbars. In [23], Zhang et al proposed CANTINA, a content-based method, which performed competitively against two state-of-the-art toolbars, SpoofGuard and Netcraft. We implemented an offline version of CANTINA, and evaluated our algorithms with CANTINA on the same corpus.

Table 3 shows that our algorithm outperformed CANTINA significantly on FP, though its TP was inferior to CANTINA. Since the goal of our work is to achieve a high TP on the basis of maintaining a close-to-zero FP, we can accept this slight degradation on TP in exchange of a dramatic improvement on FP. All real-world anti-phishing applications call for an extremely low FP, and thus our solution is more effective and practical.

Table 3. Experiment reveals that our approach beats CANTINA significantly on FP with some degradation on TP, indicating that our method is more practical and effective in real-world scenarios. A combination of millersmiles and Google whitelists was used here.

	No FP filter	With FP filter	CANTINA
TP(%)	73.53	67.15	76.25
FP(%)	0.03	0.0	1.14

5 Discussion

In this section, we briefly discuss the merits and limitations of our current approach, and offer some ideas on how to address these problems.

5.1 Domain Whitelist and URL Blacklist

Domain whitelists have been shown in our experiment to be able to reduce FP, and yet using them has risks too. For example, phishing sites hosted on certain legal domains on our whitelists will be wrongly filtered as legitimate cases. However, we can always reduce this impact by only using a core list of most targeted legal domains such as bankofamerica.com, whose defence systems are usually superb, and therefore, it will be extremely difficult for attackers to evade being detected by planting phishing sites into such legitimate domains.

Though our approach makes use of whitelists in the first step of the pipeline, it does not rely solely on them to achieve an acceptable FP. As suggested in Sect. 4.4, by only utilizing a small whitelist and mostly relying on the search engine-based filtering to slash false positives, our approach does not suffer from the incompleteness of whitelists and thus is scalable and realistic.

Currently, URL blacklists cannot detect new attacks unless the phishing URLs remain the same, which is unlikely due to the phishing nature of constantly avoiding tracking. Our work in this paper demonstrates a way to augment existing blacklists with conservative probabilistic strategies, and therefore we did not conduct an experiment specifically using traditional blacklists only.

5.2 Blacklist-Based Soft Matching

Obtaining new phishing patterns in a timely fashion is critical to signature-based anti-phishing methods, and our approach addresses this problem by means of a sliding window that incrementally and constantly adds verified phish from blacklists into our database. Though the first few cases of new attacks are initially able to evade our detection, we only need to identify a few new phishing instances to update our model, subsequently being able to block the rest of the large volume of phishing attacks built by toolkits while maintaining a nearly zero FP.

This design philosophy emphasizes the adaptiveness and responsiveness of a usable phish detection system, and is a significant improvement over the traditional blacklist-based methods that are generally unable to cope with a high volume of unique phish URLs. To enlarge the range of phishing variants covered, our approach can be easily generalized to other phish feed like the APWG feed with the assistance of the whole anti-phishing community.

We could further exploit the dynamic aspects of phishing blacklists to improve our approach. For instance, we could prioritize our phish database by putting the phishing attacks with the most matches in a recent period of time in the top position for future comparison. In addition, although we currently focus on English websites, the general idea of our approach and the pattern of toolkit-based phishing attacks carry over to non-English sites, and our approach could be modified slightly to accommodate that change.

5.3 Effectiveness of TF-IDF Filtering

Our TF-IDF filter module has been shown to be effective by its extremely low number of FPs (see Sect. 4.4). Considering the fact that phishing activity always

targets well-known brands due to its lucrative nature, false positives tend to be raised almost entirely on those popular target sites, which are very likely, if not almost certainly, to be indexed by major search engines. Accordingly, querying TF-IDF ranked terms plus the domain keyword will return the page domain as a top result with a high probability, thus successfully removing the false positive.

On the other hand, true phishing attacks are not as likely to be filtered by this module, thanks to the very nature that phishing sites rarely last long enough to be included in a web index. This will be difficult for phishers to change, because creating indexable, long-lived sites implies either access to stable hosting infrastructure, or that hacked sites will be able to go undiscovered for significant lengths of time. We believe the former is going to be too risky or expensive for phishers to engage widely in, and the latter is already being addressed by existing take-down approaches.

5.4 Legitimate Corpus

Our legitimate webpage collection mainly focuses on popular sites, commonly spammed sites, webpages with characteristics similar to phish (such as login forms), etc., and by appraising our idea on these hard cases, we actually provide worst case performance statistics, which is more beneficial for the real-life application and deployment that follow.

Our data set is by no means representative of what users experience during their everyday browsing. In [9], Bennouas et al proposed a random web crawl model and found through large-scale experiments that the in-degree, out-degree and pagerank of webpages follow power laws. Given the profit-driven nature of phishing activity, it is unlikely that the gigantic number of low-profile and less popular sites resemble the phishing pages with respect to the content, and not using those data in our experiment has no impact on the experiment result.

5.5 Runtime Performance

On a machine with 1.73 GHz CPU and 2.00G RAM running Windows XP, our detection algorithm took about 229.11 milliseconds on average to check each web page with a standard deviation of 220.99 milliseconds. Filtering via whitelists took roughly 0.18 milliseconds per URL. The phish detection phase via the shingling algorithm in our pipeline is essentially an parallel problem, and it should scale well because our phish database can be easily distributed into multiple computers and the process of matching page content probabilistically via database scanning can be easily parallelized.

We have two points of discussion here. First, we have not done many optimizations to our system for performance. Second, our approach is an embarrassingly parallel problem, one that scales well without a lot of effort, simply by adding more computers. As such, existing blacklist services, ISPs, and takedown services could easily use our approach to improve their ability and timeliness in detecting phishing sites. The main limiting factor in terms of runtime performace is bandwidth. The TF-IDF filter in our system queries Google to verify

the legitimacy of a webpage, which involves a round-trip traffic overhead on the web. However, this filter is only triggered when a page is sufficiently similar to an existing phishing signature, and considering the fact that the vast majority of the pages will not in any way resemble phishing attacks, the query frequency should be low enough. Moreover, caching search results on the client side is of paramount importance to speed up the performance, and may to a certain extent alleviate the network traffic problem.

5.6 How Phishers May Respond

We do not claim that our approach will solve the phishing problem. Rather, our specific claim is that we can augment existing blacklists in a very conservative manner using probabilistic techniques, with good results in terms of TPs and FPs. Taking a wider perspective, criminals will inevitably come up with countermeasures to the specifics of any deployed technique. What we offer here is a new way of thinking about how to effectively combine human-verification with ML and IR techniques, to increase the effectiveness of existing blacklists.

Phishers could try to penetrate our method by HTML obfuscation tricks such as injecting garbage text to the DOM with tiny or invisible fonts, background color, 100% transparency, multiple i-frames. These are, however, more of an implementation issue than a design one, and we can easily peel off those special effects and extract intentionally separated text by manipulating the DOM in the preprocessing step. Our system also cannot detect Flash-based phishing, and would require other techniques to detect these. The use of other types of popular obfuscation techniques such as doorway pages, chains of redirections, URL cloaking and so on is also futile in front of our algorithm. The reason is that no matter how many irrelevant intermediate steps attackers try to involve, web users will land in the actual phishing webpage eventually, and our content-based detection idea still applies. As a matter of fact, such tricks in hope of obfuscating online customers and anti-phishing algorithms turn out to be beneficial to our method in that search engines are even less likely to crawl those phishing sites given such gimmicks, and our search-oriented filtering module is thus more unlikely to incorrectly filter the corresponding phishing attacks as legitimate cases. These kinds of tricks could also be new features for machine learning algorithms as well, again since few legitimate sites would use such techniques.

Another likely countermeasure that criminals would attempt is to have toolkits include some element of randomization, making it harder to do soft matching. This, however, is not hard to cope with. If the randomization is in the invisible part of the DOM, our argument in the beginning of the previous paragraph still applies, and we can easily extract the real content. Should the random elements be added to the web page content, we could tune the resemblance threshold t accordingly and still achieve a reasonable detection rate. On the other hand, there is a limit on how much random noise attackers could add before web users start feeling suspicious about the legitimacy of the web pages. Furthermore, restricting people's awareness while hindering probabilistic matching simultaneously by adding noise is not an easy task, which would make the process of designing

phish toolkits very difficult and thus significantly limits the vast production of phishing attacks, rendering the cost-benefit undesirable for the criminals.

It is very hard for attackers to elevate the FP of our approach, since the design of legitimate webpages and the crawling process of search engines are beyond their control. It is even harder to trick search engines to give their phishing sites higher rankings, due to the scrutiny of search engines, short-lived nature of phishing behavior and negligible popularity scores of the phishing sites.

6 Conclusion

In this paper, we presented a system that combined human-verified blacklists with information retrieval and machine learning techniques, yielding a probabilistic phish detection framework that can quickly adapt to new attacks with reasonably good true positive rates and close to zero false positive rates.

Our system exploits the high similarity among phishing web pages, a result of the wide use of toolkits by criminals. We applied shingling, a well-known technique used by search engines for web page duplication detection, to label a given web page as being similar (or dissimilar) from known phish taken from blacklists. To minimize false positives, we used two whitelists of legitimate domains, as well as a filtering module which uses the well-known TF-IDF algorithm and search engine queries, to further examine the legitimacy of potential phish.

We conducted extensive experiments using phish from Phishtank and legitimate web pages from seven different sources. These experiments showed that our proposed method had a TP of 73.53% and a FP of 0.03% without TF-IDF filtering, and a TP of 67.15% and zero FP with TF-IDF filtering under the optimal setting. Moreover, our approach is able to adapt quickly to zero-hour attacks by incrementally building the model via a sliding window with a few new phishing instances out of a huge magnitude of phishing attacks created by toolkits, thus providing a feasible framework for industry to vastly improve the existing limited blacklists without increasing their false positives. This sliding window mechanism also leads to good balance between accuracy and runtime efficiency: with only two weeks' worth of training phish, our method had a TP of 65.02% with 0% FP using search oriented-filtering, and a TP of 71.23% and a FP of 0.03% without FP filtering.

Acknowledgements. This work has been supported by NSF grants CCF-0524189 and DGE-0903659. Additional support has been provided ARO research grant DAAD19-02-1-0389 to Carnegie Mellon University's CyLab, and the CMU/Portugal Information and Communication Technologies Institute.

References

1. http://www.gartner.com/it/page.jsp?id=565125
2. http://www.phishtank.com/stats.php
3. http://sb.google.com/safebrowsing/update?version=goog-white-domain:1:1

4. http://www.millersmiles.co.uk/scams.php
5. http://data.phishtank.com/data/online-valid/
6. http://dir.yahoo.com/Business_and_Economy/Shopping_and_Services/Financial_Services/Banking/Banks/
7. http://rdf.dmoz.org/
8. Abu-Nimeh, S., Nappa, D., Wang, X., Nair, S.: A comparison of machine learning techniques for phishing detection. In: Proceedings of the Anti-Phishing Working Groups (APWG) 2nd Annual eCrime Researchers Summit, pp. 60–69 (2007)
9. Bennouas, T., de Montgolfier, F.: Random web crawls. In: Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pp. 451–460 (2007)
10. Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G.: Syntactic clustering of the web. In: Proceedings of the Sixth International Conference on World Wide Web, pp. 1157–1166 (1997)
11. Cova, M., Kruegel, C., Vigna, G.: There is no free phish: An analysis of 'free' and live phishing kits. In: Proceedings of the 2nd USENIX Workshop on Offensive Technologies, WOOT 2008 (2008)
12. Fetterly, D., Manasse, M., Najork, M.: On the evolution of clusters of near- duplicate web pages. In: Proceedings of the First Conference on Latin American Web Congress, pp. 37–45 (2003)
13. Garera, S., Provos, N., Chew, M., Rubin, A.D.: A framework for detection and measurement of phishing attacks. In: Proceedings of the 2007 ACM Workshop on Recurring Malcode, pp. 1–8 (2007)
14. Henzinger, M.: Combinatorial algorithms for web search engines: three success stories. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1022–1026 (2007)
15. Herley, C., Florencio, D.: A profitless endeavor: phishing as tragedy of the commons. In: Proceedings of the 2008 Workshop on New Security Paradigms, pp. 59–70 (2009)
16. Ludl, C., McAllister, S., Kirda, E., Kruegel, C.: On the effectiveness of techniques to detect phishing sites. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 20–39. Springer, Heidelberg (2007)
17. Moore, T., Clayton, R.: Examining the impact of website take-down on phishing. In: Proceedings of the Anti-phishing Working Groups (APWG) 2nd Annual eCrime Researchers Summit, pp. 1–13 (2007)
18. Pan, Y., Ding, X.: Anomaly based web phishing page detection. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186, pp. 381–392. Springer, Heidelberg (2006)
19. 3sharp report: Gone phishing: Evaluating anti-phishing tools for windows. Tech. rep. (September 2006), http://www.3sharp.com/projects/antiphishing/gone-phishing.pdf
20. Sheng, S., Kumaraguru, P., Acquisti, A., Cranor, L., Hong, J.: Improving phishing countermeasures: An analysis of expert interviews. In: Proceedings of the 4th APWG eCrime Researchers Summit (2009)
21. Sheng, S., Wardman, B., Warner, G., Cranor, L., Hong, J., Zhang, C.: An empirical analysis of phishing blacklists. In: Proceedings of the 6th Conference on Email and Anti-Spam (2009)
22. Xiang, G., Hong, J.: A hybrid phish detection approach by identity discovery and keywords retrieval. In: Proceedings of the 18th International Conference on World Wide Web (WWW 2009), pp. 571–580 (2009)
23. Zhang, Y., Hong, J., Cranor, L.: Cantina: a content-based approach to detecting phishing web sites. In: Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pp. 639–648 (2007)

Appendix: Detailed Results under Various Settings

Table 4. TP (%) under day/hour-measured sliding window. Under all settings, shingling with hour-level incremental model building is more responsive to phishing attacks, attaining higher TPs under all L values. Our approach achieved almost optimal TP with only 1 month's worth of training phish.

	No TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	45.28	53.19	63.91	67.4	**69.53**	**69.53**	55.23	61.62	69.11	71.83	**74.13**	**74.13**
Google	44.94	52.68	63.32	66.81	**68.94**	**68.94**	54.72	61.11	68.51	71.23	**73.53**	**73.53**
None	45.28	53.19	63.91	67.4	**69.53**	**69.53**	55.23	61.62	69.11	71.83	**74.13**	**74.13**
Both	44.94	52.68	63.32	66.81	**68.94**	**68.94**	54.72	61.11	68.51	71.23	**73.53**	**73.53**
	With TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	43.66	50.55	59.66	62.13	**64.17**	**64.17**	52.34	57.79	63.83	65.62	**67.74**	**67.74**
Google	43.32	50.04	59.06	61.53	**63.57**	**63.57**	51.83	57.28	63.23	65.02	**67.15**	**67.15**
None	43.66	50.55	59.66	62.13	**64.17**	**64.17**	52.34	57.79	63.83	65.62	**67.74**	**67.74**
Both	43.32	50.04	59.06	61.53	**63.57**	**63.57**	51.83	57.28	63.23	65.02	**67.15**	**67.15**

Table 5. FP (%) under day/hour-measured sliding window. Whitelists lessen the FPs, reaching 0.1199%, 0.06%, 0.5396%, 0.03% respectively with the millersmiles, Google, none and both whitelists at $L = 60$ days or $L = 1440$ hours. The search engine oriented filtering step significantly significantly improves the FPs , downsizing FP values in all settings to zero.

	No TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	0.00	0.00	0.03	0.1199	0.1199	0.1199	0.00	0.00	0.03	0.1199	0.1199	0.1199
Google	0.00	0.00	0.03	0.06	0.06	0.06	0.00	0.00	0.03	0.06	0.06	0.06
None	0.00	0.00	0.2098	0.4496	0.5096	0.5396	0.00	0.00	0.2098	0.3597	0.5096	0.5396
Both	0.00	0.00	0.03	0.03	0.03	0.03	0.00	0.00	0.03	0.03	0.03	0.03
	With TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Google	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
None	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Both	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Kamouflage: Loss-Resistant Password Management

Hristo Bojinov[1], Elie Bursztein[1], Xavier Boyen[2], and Dan Boneh[1]

[1] Stanford University
[2] Université de Liège, Belgium

Abstract. We introduce Kamouflage: a new architecture for building theft-resistant password managers. An attacker who steals a laptop or cell phone with a Kamouflage-based password manager is forced to carry out a considerable amount of online work before obtaining any user credentials. We implemented our proposal as a replacement for the built-in Firefox password manager, and provide performance measurements and the results from experiments with large real-world password sets to evaluate the feasibility and effectiveness of our approach. Kamouflage is well suited to become a standard architecture for password managers on mobile devices.

1 Introduction

All modern web browsers ship with a built-in password manager to help users manage the multitude of passwords needed for logging into online accounts. Most existing password managers store passwords encrypted using a master password. Firefox users, for example, can provide an optional master password to encrypt the password database. iPhone users can configure a PIN to unlock the iPhone before web passwords are available.

By stealing the user mobile device the attacker is able to obtain the password database encrypted under the master password. He or she can then run an offline dictionary attack using standard tools [17,14] to recover the master password and then decrypt the password database. We examined a long list of available password managers, both for laptops and smartphones, and found that all of them are vulnerable to offline attack. To address this threat, several potential defenses quickly come to mind:

The first one is to use salts and slow hash functions to slow down a dictionary attack on the master password. Unfortunately, these methods do not prevent dictionary attacks [5]; they merely slow them down. Moreover on mobile devices, we found that password managers tend to offer the use of a numerical PIN code to protect the database. While PIN codes are more easy to use on a smartphone, they also offer a smaller key space which makes the aforementioned attacks easier.

Another potential defense is to use a password generator [18] rather than a password manager. A password generator generates site-specific passwords from a master password. Users, however, want the ability to choose memorizable

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 286–302, 2010.

passwords so that they can easily move from one machine to another; this can be quite difficult with the strings typically created by a password generator. As a result, if a password generator is not ubiquitous, and currently none are, then the majority of users will never use one.

Finally, another defense is to store passwords in the cloud [21,7] and use a master password to authenticate to the cloud. This solution only shifts the problem to the cloud; any employee at the cloud service can run an offline dictionary attack to expose account passwords for multiple users. Other potential defenses and their drawbacks are discussed in Section 6.

Our contribution. We propose a new architecture for building theft-resistant password managers called *Kamouflage*. Our goal is to force the attacker to mount an *online* attack before he can learn any user passwords. Since online attacks are easier to block (e.g. by detecting multiple failed login attempts on the user's account) we make it harder for an attacker to exploit a stolen password manager. Major websites already implement internal security mechanisms that throttle after several failures and therefore forcing an attacker to perform online work is an effective defense.

Kamouflage works as follows. While standard password managers store a single set S of user passwords, Kamouflage stores the set $S_0 = S$ along with $N-1$ decoy sets $S_1, \ldots, S_{N-1}$. A typical value for N is $N = 10,000$, but larger or smaller values are acceptable. Based on our personal experience $M = |S|$, the size of the real stored password set, is on the order of 100 (a pessimistic estimate: in reality users on average have fewer than ten passwords [6]). The key challenge for Kamouflage is to generate decoy sets that are statistically indistinguishable from the real set. This is difficult because as shown by our user survey, users tend to pick memorizable passwords that are closely related. If Kamouflage simply picked random decoy sets, an attacker would be able to easily distinguish the real password set from the decoys.

With Kamouflage, an attacker who steals the device will be forced to perform, on average, $N/2$ online login attempts (or $N/2M$ at *each* of the M websites) before recovering the user's credential. Web sites can detect these failed login attempts and react accordingly. With web site participation, Kamouflage can be further strengthened by having the user's device registering a few decoy passwords (say 10) at web sites where the user has an account. If a web site ever sees a login attempt with a decoy password, the site can immediately block the user's account and notify the user. This is similar in spirit to the warning displayed by GMail when an account is accessed from two different countries in a brief period of time.

Using decoy password sets is a practical approach because if we assume that each user has about 100 passwords and each password takes about 10 bytes, then the decoy sets will take about 10MB of storage, which is roughly the size of three MP3 files, which a negligible storage requirement for modern laptops and smartphones.

Here are the main difficulties we had to overcome to make Kamouflage work:

- **Human-memorable passwords:** Since decoys must look like human memorizable passwords, we need a model for generating such passwords. To build such a model we performed a study of human passwords, as discussed in Section 2. We also took advantage of previous work on this topic [24,13,23,22]. While most previous work studied this problem for the purpose of speeding up dictionary attacks, here we give the first "positive" application for these password models, namely hiding a real password in a set of decoys.
- **Related passwords:** Since humans tend to pick related passwords, Kamouflage must pick decoy sets that are both chosen according to a password model and related to each other as real users tend to do. We develop a model for *password sets* that mimics human behavior.
- **Relation to master password:** In some cases it makes sense to encrypt the password database using a master password. Unfortunately, our experiments found that users tend to pick master passwords that are themselves related to the passwords being protected. Hence, we had to develop an encryption scheme that cannot be used to rule out decoy sets. We present our approach in Section 5.1.
- **Site restrictions:** Finally, different sites have different password requirements. When generating decoy sets we have to make sure that all passwords in the decoy set are consistent with the corresponding site policy.

We built a Kamouflage prototype as a drop-in replacement for the Firefox password manager. We give performance numbers in Section 4.2 where we show that Kamouflage has little impact on user experience. Our design is complementary to mechanisms that aim at preventing password theft in-transit, such as key-logging malware. Ideally both off-line and on-line protections should be deployed in order to ensure password safety.

2 How Users Choose Passwords

In this section we present the results of our experiments on user password behavior that we use to support our assumptions that users are uncomfortable with random independent passwords, and as a result tend to select predictable and related passwords across multiple sites. In order to test this and related hypotheses, we conducted two experiments: qualitative user interviews and quantitative analysis of large, real-world password databases (one of them containing over 30 million entries). The results complement existing work in the area, such as [6].

Empirical motivation from user interviews. Our first experiment was a survey conducted with undergraduate and post-doctoral students on our campus. The goal of the survey was not to learn people's passwords, but to elicit their approach to dealing with passwords. The interviews were performed via an in-person questionnaire, and completed by 87 individuals: 30 CS undergraduate students, and 57 post-doctoral researchers from various fields including biology,

Table 1. General properties of password databases. phpbb data is skewed towards shorter lengths (selection bias) because we had to crack it before analyzing it. We confirm the observations about password shape made in [24].

Name	Entries	Size: 1-4	Size: 5-8	Size: 9-12	Size: 12+
RockYou	32.6M	0.2%	69.3%	27.0%	3.5%
phpbb	343K	4.2%	82.3%	13.4%	0.1%

chemistry and psychology. 33% of the respondents were female, and over 61% were between 26 and 35 years old. It can be said that our subjects represented the worst case scenario for an attacker in the sense that they were highly educated (more than 60% having a PhD) and sophisticated in their use of the Internet.

In the survey we asked direct questions about users' password habits. We briefly summarize the results due to space constraints: 81% of our subjects admitted to reusing the same password on many websites, thereby supporting our hypothesis on password reuse. This hypothesis is also supported by the number of passwords used by our subjects: 65% of the sample reported using at most 5 passwords, and 31% reported using between 6 and 10 passwords. In a separate question, 68% of the sample admitted selecting related but not necessarily identical passwords across sites. We also found that 83% of our subjects reported that they did not password-protect their smartphone, despite the presence of private data on the phone.

Password database analysis. In our second experiment we analyzed two real-world password databases that were recently leaked to the public. One was from the RockYou service: it contained 32 *million* entries [20], and we had access to all the entries in plaintext. As a consequence, the results from its analysis can be considered a highly reliable predictor of user behavior. The other password database was from the developer web site `phpbb.com`. We include it here for comparison purposes, because `phpbb.com` does not enforce any password requirements, so users were free to use whatever they want. The `phpbb.com` database contained 343 000 passwords. In this database, passwords were not listed in the clear, but as hashes created by one of two schemes: a simple MD5 and 2048-fold MD5 with salt. Using a cluster of computers over several months, we where able to recover 241 584 passwords, or 71% of the full database. Since we did not have direct access to the passwords as plaintext, our recovery process induces a *selection bias* towards easier passwords, hence the `phpbb.com` numbers should be used only as a secondary reference point.

Table 1 shows some basic properties of the analyzed databases. In our work, the primary concern was the structure of passwords: understanding this structure is the key to being able to generate high-quality plausible decoys. Accordingly we tested the hypothesis that people use known words in their passwords by comparing the databases to the dictionaries created by *openwall* to work in conjunction with the famous cracker "John the ripper" [17]. When combined these dictionaries contain around 4 millions words (note that not all of these

Table 2. The effectiveness of simple word-based rules in parsing passwords. Percent matched increases with each rule. The high coverage makes it possible to use this approach for password set analysis (Section 4).

Rule Name	Format	RockYou	RockYou %	phpbb.com	phpbb.com %
Strict	W	6.6M	20.2%	80.0K	33.2%
Post	Wd+	6.9M	41.4%	37.7K	48.8%
Concat	WW	6.1M	60.1%	50.0K	69.6%
Digit	d+	5.2M	76.1%	32.4K	83.0%
Concat-Post	WWd+	2.4M	83.4%	9.3K	86.9%

Table 3. Examples of passwords that did not match our rules. A large portion of the remaining passwords can be classified based on additional rules like "hax0r" letter-digit substitution and "iluv***" three word and letter concatenation.

Password	Reason for not matching
`lordburnz`	Letter substituted ('s' → 'z')
`php4u`	Word-digit, word-letter, three tokens
`ilove$$$`	Non-alphanumeric, three tokens

are words from natural languages, but can rather be viewed broadly as "known password tokens").

Our analysis tool tried to match each password in the databases to one or two dictionary words according to several rules: direct match; direct match with a numerical suffix; match two words concatenated; etc. Just by using five rules we were able to match more than 80% of the passwords in both databases (Table 2). This result implies that for most users we can automatically produce the rules that were used in coming up with the passwords. Our password manager can then use the derived rules to generate new, plausible password sets that meet the same constraints. The remaining users appear to use more advanced password generation rules (Table 3), which can be emulated by building a simple N-gram model based on the extensive available data.

3 Threat Model

The basic threat model. We consider an attacker who obtains a device, such as a laptop or smartphone, that contains user data stored in a password manager. The password manager stores user passwords for online sites (banking, shopping, corporate VPN) and possibly personal data such as social security and credit card numbers. The attacker's goal is to extract the user's data from the password manager.

We assume that the password manager encrypts the data using a master password. On Windows, for example, password managers often call the Windows DPAPI function `CryptProtectData` to encrypt data using a key derived from the user's login credentials. In this case we treat the user's login password as the

master password. Other passwords managers, such as the one in Firefox, let the user specify a master password separate from the login password.

An attacker can defeat existing password managers by an offline dictionary attack on the master password. Hence, simply encrypting with a master password cannot result in a secure password manager (unless the master password is quite strong). We capture this intuition in our threat model by saying that the attacker has "infinite" computing power and can therefore break the encryption used by the password manager. This is just a convenience to capture the fact that encryption based on a human password is weak.

In our basic threat model we assume that the attacker has no side information about the user being attacked such as the user's hobbies, age, etc. The only information known to the attacker are the bits that the password manager stores. We relax this assumption in the extended threat model discussed below. We measure security of a password manager by the expected number of online login attempts the attacker must try before he obtains some or all of the data stored in the password manager. The attacker is allowed to attempts to log on different and unrelated sites. We count the expected total number attempts across all sites before some sensitive data is exposed. To deal with web sites that have no online attack protections, Kamouflage will compartmentalize passwords into groups to ensure that the most sensitive passwords are protected in all cases, even if the less important ones are cracked successfully.

Extended model: taking computing time into account. In the basic model we ignored the attacker's computing time needed to break the encryption of the password manager. In the extended model we measure security more accurately. That is, security is represented as a pair of numbers: (1) expected offline computing time; and (2) expected number of online login attempts until information stored in the password manager is exposed. This allow us to more accurately compare schemes. For this extended model we are using encryption to slow down the attacker. The key challenge here is to design an encryption scheme that does provide information to the attacker that he can use to reduce the amount of his online work. Using encryption allows us to relax our basic-model assumptions. We permit the attacker to have side information beyond the device data, such as the user's age, gender, hobbies, etc. As we will see, we can offer some security even if the attacker has intimate knowledge of the victim.

Non-threats. In this paper we primarily focus on extracting data from long-term storage. We do not discuss attacks against the device or its operator while in active use (such as shoulder snooping and key loggers), and similarly omit hardware-based side-channel attacks, which can come in many guises, based, e.g., on electromagnetic emanations ("van Eck phreaking") and capacitive data retention ("cold-boot attacks"), to name but a few possibilities. Similarly, we do not address social engineering attacks such as phishing. While phishing is an effective way to steal user passwords, addressing it is orthogonal to our goal of providing security in case of device theft or loss.

4 Architecture

At any point in time, the password management software maintains a large collection of plausible password sets; exactly one of these sets is the real one, and the rest are *decoys*. Figure 1 illustrates the storage format. Apart from passwords, all other site information is kept in the clear. That is, an attacker looking at the database knows which sites the user has passwords for: she just has no idea what the passwords are. When the real user launches his web browser, he is prompted for the master password (MP) which is then cryptographically transformed to obtain the index of the real password set in the collection. During a dictionary attack, any attempt to guess the master password results in a different (plausible and valid, but incorrect) set of passwords.

Database operations. The following are the core operations that a password database needs to support, along with a description of our implementation for them. Note that in Kamouflage these operations are well-defined for any choice of master password: that is, by guessing a master password and attempting database operations the attacker does not obtain any new information about the validity of his guess.

- **Add a new password to the database.** In our design, this amounts to adding a new password to each password set in the collection. The real password set gets the user-supplied value, while decoy sets get auto-generated entries influenced by the true one.
- **Remove.** Remove a password from the database. This is the reverse to adding a password. Only requires removing the web site's entry from each password set. No regeneration is needed.
- **Update.** Update a password entry, presumably when a password is changed. While the size of the password set collection will not change, the corresponding entry in each decoy password set must also be regenerated. On the one hand, this step ensures that the reuse patterns in the decoy sets continue to match those in the real password set (subject to random mutations, of course). On the other hand, we are also preventing an attacker from looking at the database image before and after an update, and finding the real password set based on the fact that it is the only one that has changed. Journaling file systems make this second attack vector very plausible.
- **Find.** Find the right password given some web page characteristics like URL and form field names. Given the master password, this is a straightforward database access in the real password set, ignoring all decoys. (Of course, the MP is needed to locate the correct set.)

4.1 Password Set Generation

Decoy password sets must be indistinguishable from the real password set. We extend the context-free grammar approach from [22] to work for the case where multiple passwords are being generated for the same user. In [22], passwords candidates are generated by assigning probabilities to password templates (the

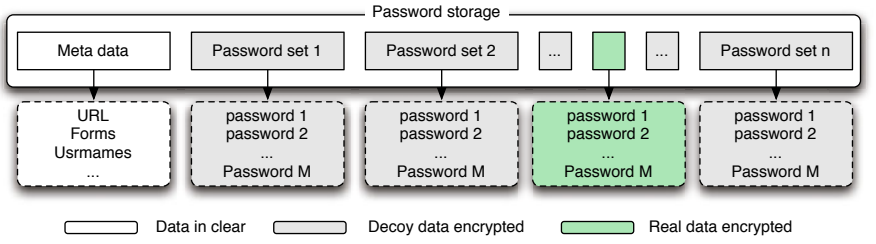


Fig. 1. Kamouflage database: cleartext metadata and encrypted real and decoy password sets. Encryption is discussed in Section 5.1.

templates look like "$l_4 d_2 s_1$", meaning "a four-letter word, followed by two digits, followed by a special character). The important insight there is that the likelihood of a password being used is determined by the likelihood of its template, in addition to the likelihood of its components. We point out that for the purpose of decoy password set generation, varying the templates is unnecessary, and even dangerous if done incorrectly: by performing asymmetric transformations on the template, the password manager might leak information about the correct password set.

In Kamouflage, decoy set generation proceeds in three steps:

1. **Tokenization:** The password manager converts the real user passwords into rules of the form $P_i \rightarrow T_{i_1} ... T_{i_i}$. The tokens typically stand for words or numbers of a certain size from a fixed dictionary[1]. In addition, tokens are reused across passwords, ensuring that portions that are shared by several passwords are correctly represented as equal by the rules. Tokenization is performed by attempting to partition each password according to the different rules from Section 2.
2. **Validation:** The system confirms that the tokenization is good: any tokens which are not dictionary words are flagged and reviewed by the user. This is the user's opportunity to remove from passwords any words that are specific to him, such as a last name or a birth date—such words can almost certainly be used to identify the correct password set among all decoys. In a perfect scenario, all of the users passwords will be readily tokenizable using standard dictionary words, possibly combined with apparently random characters. (As a consequence, decoys will be generated by using words of similar probability, along with similarly distributed random characters.)
3. **Generation:** Decoy sets are generated using the derived rules. We note that if the validation step completed without any tokens being flagged, then the fixed dictionary is as likely to generate the real password set as any of the decoys. The ease with which we can argue the statistical properties of generated decoys is the main reason we converged on this model.

[1] The dictionary used can be customizable to be able to accommodate different languages, or combinations of languages. However allowing arbitrary user customizations may lead to compromising security.

Table 4. Example decoy sets generated by the three-step algorithm

Set	Description	Site #1	Site #2
S_0	Real	jones34monkey	jones34chuck
S_1	Decoy #1	apple10laptop	apple11quest
S_2	Decoy #2	tired93braces	frame93braces
S_3	Decoy #3	hills28highly	hills48canny

A short example will clarify the password generation mechanism outlined above: let's assume the real set is of size two ($M = 2$), and we need to generate three different decoy sets ($N = 4$). Suppose the real password set consists of the passwords "jones34monkey" and "jones34chuck". The rules that are output by the tokenization step are: $P_1 \rightarrow ABC, P_2 \rightarrow ABD, A \rightarrow W_5, B \rightarrow D_2, C \rightarrow W_6, D \rightarrow W_5$. Depending on the dictionary used, the validation step could complete successfully, or maybe some of the words could be flagged as not present in the dictionary. For example the system could alert the user that "jones" is not a word from the dictionary, and as such does not blend in with generated decoy sets. The validation could also scan the contents of the user's device (e-mails, contacts, etc.) and specifically find words that are specific to the user even though they can be found in the dictionary—"chuck" is not likely a random 5-letter word if the user has a close relative called "Charles". After all issues are resolved at validation, the system generates decoy sets that might look like to ones in Table 4.

Optionally, the process of generating decoys can be customized by the user to better mimic the real password distribution. The customization choices should not be stored on disk as they can help the attacker.

4.2 Implementation

In order to prove the feasibility of our architecture, we built a proof-of-concept extension for the Firefox web browser, called Kamouflage. The extension implements the `nsILoginManagerStorage` interface and acts as an alternative storage for login credentials.

The main goal in developing the extension was to show that the overhead of maintaining decoy password sets is acceptable, particularly from a user's point of view. We intentionally used no optimizations in the handling of the password database, because we wanted to get a sense of the worst-case performance implied by our approach. Completing the extension to a point where it can be deployed for real-world use is straight forward.

When it is loaded, Kamouflage registers with Firefox as a login manager storage provider (`nsILoginManagerStorage`). Each of the implemented API methods (`addLogin`, `findLogin`, etc.) calls some internal methods that deal with reading and writing the password database file from and to persistent storage, as outlined earlier (see Figure 1). If the password storage file does not exist, it is assumed that the user's password set is the empty set.

Table 5. User-visible performance of the Kamouflage Firefox extension for three typical use cases. The estimate of 20 passwords per user is realistic, while 100 passwords are a worst-case scenario unlikely to occur in practice [6].

Collection size (number of decoy sets)	10^3	10^4	10^4
Password set size (number of user passwords)	100	100	20
Database size on disk	2MB	20MB	4MB
Measured performance (access and update time)	< 1 sec	5 sec	< 1 sec

Performance. We measured how individual API calls are impacted by various password set collection sizes. We show that performance in Table 5. In our implementation the password file is read in its entirety every time a password is accessed, and written out completely for every update. From a user's point of view, there is no impact when maintaining approximately 10^3 decoy password sets; at 10^4 decoy sets the performance drop becomes clearly noticeable.

In practice, the performance of our prototype could be further improved in a number of ways:

- **Caching.** Our measurements of user-visible latency often include several invocations of the `nsILoginManagerStorage` interface, each of which reads the whole file from scratch. The login manager could cache the database contents, reading the file only once, at launch time.
- **Read size.** Password storage does not need to be read in its entirety. Given a master password (input by the user), only one password set needs to be read from disk. In the context of a Firefox extension this would require writing a native implementation for the read function: the JavaScript file I/O API available does not allow random access inside a file.
- **Write size.** Password sets do not all have to be rewritten on every `addLogin` or `updateLogin` operation, if we can guarantee that older versions are overwritten and unavailable to an attacker.

5 Extensions

The system described in Section 4 does not encrypt the password database with a master password. The master password is only used to identify the location of the real password set. As we will see, encrypting the password database without exposing the system to an offline dictionary attack is not trivial. In this section we extend Kamouflage to address this issue and others.

5.1 Why and How to Encrypt

Side information is dangerous. An attacker armed with side information about the victim can be successful, being able to guess the correct password set in the collection by searching for victim-related keywords in the password storage, hoping that those keywords appear as part of a password. Alternatively, if the user elects to use a very weak password at a specific, unimportant web site, the attacker may be able to recover it in an online attack, and use that

information later on to crack the master password offline. Both of these attacks are reason enough to consider using encryption techniques similar to those used by current systems.

Password managers often encrypt the password database with a master password, denoted MP. In our settings this is non-trivial, and if done incorrectly, can cause more harm than good. To see why, suppose the password database is encrypted using an MP. Our user study from Section 2 shows that people tend to choose master passwords that are related to the passwords being protected. An attacker who obtains the encrypted password database can find the MP with an offline dictionary attack and then quickly identify the real password set by looking for a set containing passwords related to the MP.

Our approach to encryption. We use the following technique to avoid the preceding problem. Recall that each password set S_i contains a set of related decoy passwords generated as discussed in Section 4.1. We use the same approach to generate a master password MP_i for the set S_i, so that MP_i will likewise be related to the passwords in S_i. The master password for the real set is the user-selected MP. Now, for each set S_i do:

- generate a fresh random value IV_i to be stored in the clear with the set S_i, and
- use two key derivation functions (KDF) to generate two values K_i and L_i from MP_i as follows: $K_i \leftarrow \mathrm{KDF}_1(\mathrm{MP}_i,\ \mathrm{IV}_i)$; $L_i \leftarrow \mathrm{KDF}_2(\mathrm{MP}_i)$

The key K_i is used to encrypt the set S_i. The index L_i determines the position of the set S_i in the password database. In other words, the index of the set S_i in the database is determined by the master password for the set S_i. Collisions (i.e., two sets that have the same index L) are handled by simply discarding any new set that attempts to claim a busy slot, optionally regenerating it, and allowing some small fraction of decoy slots to remain unused in order to ensure short completion time. Once the whole database is generated, all master passwords MP_i are deleted. When the user enters the real master password MP the system can recompute the index L to locate the encrypted set and its IV and then recompute K to decrypt the set.

The best strategy for an attack on this system is to run through all dictionary words (candidate MPs) and for each one to compute the corresponding index L and candidate key K. Whenever the attacker eventually tries to decrypt a password set using the actual MP that was used to encrypt it, he can generally recognize this fact, causing that MP to become exposed along with the corresponding password set. However, in the end, even after decrypting all the sets in the password database with their respective correct master passwords, the attacker must still do substantial online work to determine the good set.

Table 6 shows how the master password strength affects the offline computation effort and the number of online login attempts that an attacker needs to perform, even if the attacker has perfect knowledge of the real master password distribution.

Note that when the user adds or updates a password, the system cannot add passwords to the decoy sets since it does not have the master passwords for the decoy sets. Instead, when the real set is updated, all the decoy sets and their master passwords are regenerated, and all sets are re-encrypted using new random values IV_i. The previous IVs must be securely purged from the database to thwart attacks that compare the current contents from past snapshots.

Our performance numbers from Table 5 show that the running time to perform this whole update operation remains acceptable.

By adding encryption to Kamouflage, we have neutralized the threat of attacks based on side information: depending on the MP strength, the attacker may need to spend considerable offline effort before he is in a position to mount the online attack.

Table 6. Comparison of attack difficulty in traditional and the new password management schemes, for different master password strengths (distribution known by the attacker)

	Master Password Strength		
Traditional	**Weak**	**Medium**	**Strong**
Offline (# of decryptions)	10^4	10^7	10^{10}
Online (# of login attempts)	1	1	1
Kamouflage	**Weak**	**Medium**	**Strong**
Offline (# of decryptions)	10^4	10^7	10^{10}
Online (# of login attempts)	10^4	10^4	10^4

5.2 Website Policy Compatibility

Restrictive password policies can be detrimental to the security of passwords stored using a mechanism like Kamouflage. Imagine that a web site requires that user passwords consist only of digits, while the decoy set generator randomly uses letters and digits when generating all passwords. In this scenario, an attacker looking at all the password sets in the collection can zero in on the real password set, because most likely it is the only one which contains a numeric password for that specific web site. We surveyed the top 10 web sites listed by Alexa, along with a small list of bank web sites, and tabulated their password requirements. The results are shown in Table 7, and clearly demonstrate that this danger is real.

It is evident that the major Internet web sites already allow arbitrary passwords, subject only to a minimum length requirement. Security-savvy companies such as Google, Yahoo, and Facebook realize that forcing specific password patterns on users results in a system that is more difficult to use, prone to human error, and ultimately less secure.

Kamouflage effectively deals with this challenge by mimicking the composition of a user's passwords, and ensuring that passwords containing specific classes of characters (lowercase letters, uppercase letters, digits, special characters) continue to contain those classes of characters in the generated decoy sets.

Table 7. Password strength requirements at top sites ranked by Alexa and a small group of finance-related web sites

Web Site	Password Requirement
Google	at least 8 characters
Yahoo!	at least 6 characters
YouTube	at least 8 characters
Facebook	at least 6 characters
Windows Live	at least 6 characters
MSN	at least 6 characters
MySpace	6 to 10 characters, at least 1 digit or punctuation
Fidelity	6 to 12 characters, *digits only*
Bank of America	8 to 20 characters, $\geq$ 1 digit and $\geq$ 1 letter, no `$ < > & ^ ! [ ]`
Wells Fargo	8 to 10 characters, $\geq$ 3 of: uppercase, digit, or special characters

It is conceivable that some web sites will implement weak security, or will not be significant for the user and as a result their passwords will be easy to guess. In order to prevent this weakness from helping the attacker find out the master password, and along with it the passwords for more secure and important to the user web sites, it is also preferable for the password manager to allow the grouping of web sites according to their importance. This grouping can be suggested by the user or inferred automatically, and will determine whether site passwords are kept in the protected database, or in a separate, unprotected area.

5.3 "Honeywords": Using Decoys as Attacker Traps

Some web sites are averse to blocking a user's account when they see a large number of failed login attempts. This is usually due to fear that a user's account will effectively suffer a denial-of-service attack. The reasoning behind this is sound: it is much more likely an attacker is trying to block a user's account, than trying to guess her password. It is attacks that are exceptions to this rule that have the greatest potential to cause damage however: an unauthorized user of an account could transfer money, attack other related accounts, or steal personal information to be used later for identity fraud.

We have seen that decoy password sets carry certain risks when deployed without care. At the same time, they provide an opportunity to cooperate with web sites in detecting and blocking targeted attacks on user accounts, alleviating concerns over potential DoS vulnerabilities of the lock-out logic [15].

Supplying web sites with some of their corresponding decoy passwords can provide them with an effective tool for identifying attacks that are based on compromised password files, and encourage them to take steps to block the user account in such scenarios. This presents little risk on the part of the web site, because the likelihood that a casual DoS attacker hits a decoy password, without having access to the user's device, should be very low. In other words, knowing

that an attack is not a random DoS but a genuine impersonation attempt will make web sites more willing to take immediate and decisive actions to stop the attack. This idea has been previously explored in the context of network security for identifying and rapidly blocking intrusions via honeypots [19,16,3].

5.4 Master Password Fingerprinting

The flip side of using decoy traps as a defense mechanism, is that it becomes vital to provide the user with positive feedback on the correctness of the master password being entered. With the honeyword mechanism, a mistake on the master password is indeed much more likely to result into a locked-out account than a mistake on the account's login password itself.

A simple technique similar to Dynamic Security Skins [4] can solve this problem. When the user selects his master password, he can be presented with an icon selected pseudo-randomly from several thousand possible ones, based on the master password. The user remembers the icon and uses its presence as a cue that he typed the correct master password when he logs in again later on. It is very unlikely that the user will mistype his password and at the same time get the same icon, believing the password he typed was correct. In particular, error-correction codes can be employed to ensure that single-character errors always result in different validation icons.

5.5 Kamouflage Summarized

It is instructive to take a step back and compare our extended Kamouflage architecture to a "traditional" password manager design.

The encryption step we added ensures that our password database is at least as hard to crack as a traditional one: an attacker that guesses the master password can test her guess offline, however even upon successful decryption of a password set, there will be no guarantee that the decrypted set is the real one and not a decoy. Successfully uncovering all password sets takes time proportional to the size of the master password space, which is just the same as with a traditional design. In other words, using decoy sets we are requiring the attacker to perform a significant amount of on-line work even when the whole space of master passwords has been explored offline.

Kamouflage also provides opportunities for additional security mechanisms to be deployed by web sites. We mentioned the use of honeywords, whereby a subset of the decoy password sets could be provided to web sites by the password manager, enabling them to identify and quickly respond to a targeted attack, without providing new opportunities for DoS.

Finally, the visual fingerprinting technique ensures that users have feedback on whether they entered the correct master password, without leaking any information to an attacker, and thus without weakening the strength of the master password.

6 Additional Related Work

RSA key camouflage. The idea of camouflaging a cryptographic key in a list of junk keys was previously used by Arcot systems [12] to protect RSA signing keys used for authentication. Arcot hid an RSA private key among ten thousand dummy private keys. The user's password was a 4 digit PIN identifying the correct private key. The public key was also kept secret. An attacker who obtained the list of ten thousand private keys could not determine which is the correct one. Camouflaging an RSA private key is much easier than camouflaging a password since the distribution of an RSA private key is uniform in the space of keys.

Remote password storage. Several password management systems store passwords on a remote third party server. As examples, we mention Verisign's PIP [21], the Ford-Kaliski system [7], and Boyen's Hidden Credential Retrieval [2], while noting that many other proposals exist. These systems, unlike ours, require additional network infrastructure for password management. Moreover, in some systems, like Verisign's, there is considerable trust in the third party since it holds all user passwords. An exception is Boyen's HCR scheme [2], which is designed to exploit the limited redundancy of stored passwords to prevent the third-party storage facility from validating the master password offline. Compared to [2], Kamouflage can also deal with (sets of) passwords with large amounts of redundancy.

Trusted Computing Chips. Another approach to protecting password storage is to rely on disk encryption, such as Windows BitLocker which uses special hardware (a TPM) to manage the disk encryption key. An attacker who steals the laptop will be unable to decrypt the disk, unless he or she can extract the key from the TPM. This solution, however, cannot be used on devices that have no TPM chip, such as smartphones and some laptops; it also suffers from portability problems. Clearly, we prefer a solution that does not rely on special hardware.

Intelligent dictionary attacks. Several recent papers propose models for how humans generate passwords [13,8,22]. These results apply their models to speeding dictionary attacks. Here we apply these models defensively for hiding passwords in a long list of dummy passwords.

Slow hash functions and halting functions. Many password management proposals discuss slow hash functions for slowing down dictionary attacks [1,5,10]. These methods are based on the assumption that the attacker has limited computing power. They can be used to protect the user's master password against dictionary attacks. Our approach, which is secure in the face of an attacker with significant computing power, is complementary and can be used in conjunction with slow hashing methods for additional security.

Graphical passwords. Graphical passwords [11,9] are an alternative to text passwords. While they appear to have less entropy than textual passwords, our methods can, in principle, also be used to protect graphical passwords. One would

need a model for generating dummy graphical passwords that look identical to human generated passwords. We did not explore this direction.

7 Conclusions

We presented a system to secure the password database on a mobile device from attacks that are often ignored by deployed password managers. The system leverages our knowledge of user password selection behavior to substantially increase the expected online work required to exploit a stolen password database. The mechanism can be further strengthened with the cooperation of web sites to detect decoy passwords. Using a prototype implementation we demonstrated that the system can be used as a drop-in replacement to existing systems with minimal impact on the user experience. Our experiments on real password databases suggest that the proposed decoy generation algorithm produces decoy sets that are indistinguishable from the real set.

References

1. Boyen, X.: Halting password puzzles: hard-to-break encryption from human-memorable keys. In: 16th USENIX Security Symposium—SECURITY 2007, pp. 119–134 (2007)
2. Boyen, X.: Hidden credential retrieval from a reusable password. In: ACM Symp. on Information, Computer & Communication Security—ASIACCS 2009, pp. 228–238 (2009)
3. Das, V.V.: Honeypot scheme for distributed denial-of-service. In: International Conference on Advanced Computer Control, pp. 497–501 (2009)
4. Dhamija, R., Tygar, J.D.: The battle against phishing: Dynamic security skins. In: SOUPS 2005: Proceedings of the 2005 Symposium on Usable Privacy and Security, pp. 77–88 (2005)
5. Feldmeier, D., Karn, P.: UNIX password security – 10 years later. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 44–63. Springer, Heidelberg (1990)
6. Florencio, D., Herley, C.: A large-scale study of web password habits. In: WWW 2007: Proceedings of the 16th International Conference on World Wide Web, pp. 657–666. ACM, New York (2007)
7. Ford, W., Kaliski, B.: Server-assisted generation of a strong secret from a password. In: Proc. 9th IEEE International Workshops on Enabling Technologies, pp. 176–180 (2000)
8. Glodek, W.: Using a specialized grammar to generate probable passwords. Master's thesis, Florida state university (2008)
9. Goldberg, J., Hagman, J., Sazawal, V.: Doodling our way to better authentication. In: Proceedings CHI 2002, pp. 868–869 (2002)
10. Halderman, J.A., Waters, B., Felten, E.W.: A convenient method for securely managing passwords. In: WWW 2005: Proceedings of the 14th International Conference on World Wide Web, pp. 471–479. ACM Press, New York (2005)
11. Jermyn, I., Mayer, A., Monrose, F., Reiter, M., Rubin, A.: The design and analysis of graphical passwords. In: Proc. 8th USENIX Security Symposium, pp. 135–150 (1999)

12. Kausik, B.: Method and apparatus for cryptographically camouflaged cryptographic key. US patent 6170058 (2001)
13. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space trade-off. In: Proc. of ACM CCS 2005, pp. 364–372 (2005)
14. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
15. Account lockout attack (2009), http://www.owasp.org/index.php/Account_lockout_attack
16. Portokalidis, G., Bos, H.: Sweetbait: Zero-hour worm detection and containment using honeypots. Technical report, Journal on Computer Networks, Special Issue on Security through Self-Protecting and Self-Healing Systems, TR IR-CS-015. Technical report, Vrije Universiteit (2005)
17. Project, O.: John the ripper password cracker (2005), http://www.openwall.com/john
18. Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J.: Stronger password authentication using browser extensions. In: Proceedings of USENIX security (2005)
19. Sardana, A., Joshi, R.: An auto-responsive honeypot architecture for dynamic resource allocation and qos adaptation in ddos attacked networks. Comput. Commun. 32(12), 1384–1399 (2009)
20. TechCrunch. One of the 32 million with a rockyou account? you may want to change all your passwords. like now (2009), http://techcrunch.com/2009/12/14/rockyou-hacked/
21. Verisign. Personal identity portal (2008), https://pip.verisignlabs.com/
22. Weir, M., Aggarwal, S., Glodek, B., de Medeiros, B.: Password cracking using probabilistic context-free grammars. In: Proceedings of IEEE Security and Privacy (2009)
23. Yampolskiy, R.: Analyzing user passwords selection behavior for reduction of password space. In: Proc. IEEE Int. Carnahan Conference on Security Technology, pp. 109–115 (2006)
24. Yan, J., Blackwell, A., Anderson, R., Grant, A.: Password memorability and security: Empirical results. IEEE Security and Privacy magazine 2(5), 25–31 (2004)

Sequential Protocol Composition in Maude-NPA*

Santiago Escobar[1], Catherine Meadows[2], José Meseguer[3], and Sonia Santiago[1]

[1] DSIC-ELP, Universidad Politécnica de Valencia, Spain
{sescobar,ssantiago}@dsic.upv.es
[2] Naval Research Laboratory, Washington, DC, USA
meadows@itd.nrl.navy.mil
[3] University of Illinois at Urbana-Champaign, USA
meseguer@cs.uiuc.edu

Abstract. Protocols do not work alone, but together, one protocol relying on another to provide needed services. Many of the problems in cryptographic protocols arise when such composition is done incorrectly or is not well understood. In this paper we discuss an extension to the Maude-NPA syntax and operational semantics to support dynamic sequential composition of protocols, so that protocols can be specified separately and composed when desired. This allows one to reason about many different compositions with minimal changes to the specification. Moreover, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics, and illustrate our results on some examples.

1 Introduction

It is well known that many problems in the security of cryptographic protocols arise when the protocols are composed. Protocols that work correctly in one environment may fail when they are composed with new protocols in new environments, either because the properties they guarantee are not quite appropriate for the new environment, or because the composition itself is mishandled.

The importance of understanding composition has long been acknowledged, and there are a number of logical systems that support it. The Protocol Composition Logic (PCL) begun in [9] is probably the first protocol logic to approach composition in a systematic way. Logics such as the Protocol Derivation Logic (PDL) [4], and tools such as the Protocol Derivation Assistant (PDA) [1] and the Cryptographic Protocol Shape Analyzer (CPSA) [8] also support reasoning about composition. All of these are logical systems and tools that support

* S. Escobar and S. Santiago have been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2007-68093-C02-02. J. Meseguer has been supported by NSF Grants CNS 07-16638 and CNS 09-04749.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 303–318, 2010.

reasoning about the properties guaranteed by the protocols. One uses the logic to determine whether the properties guaranteed by the protocols are adequate. This is a natural way to approach composition, since one can use these tools to determine whether the properties guaranteed by one protocol are adequate for the needs of another protocol that relies upon it. Thus in [6] PCL and in [15] the authentication tests methodology underlying CPSA are used to analyze key exchange standards and electronic commerce protocols, respectively, via composition out of simpler components.

Less attention has been given to handling composition when model checking protocols. However, model checking can provide considerable insight into the way composition succeeds or fails. Often the desired properties of a composed protocol can be clearly stated, while the properties of the components may be less well understood. Using a model checker to experiment with different compositions and their results helps us to get a better idea of what the requirements on both the subprotocols and the compositions actually are.

The problem is in providing a specification and verification environment that supports composition. In general, it is tedious to hand-code compositions. This is especially the case when one protocol is composed with other protocols in several different ways. In this paper we propose a syntax and operational semantics for sequential protocol composition in Maude-NPA [10,11], a protocol specification and analysis tool based on unification and narrowing-based backwards search. Sequential composition, in which one or more child protocols make use of information obtained from running a parent protocol, is the most common use of composition in cryptographic protocols. We show that it is possible to incorporate it via a natural extension of the operational semantics of Maude-NPA. We have implemented this protocol composition semantics via a simple program transformation without any change to the tool. We prove the soundness and completeness of the transformation with respect to the semantics.

The rest of the paper is organized as follows. In Section 2 we introduce two protocol compositions that we will use as running examples: one example of one-parent-one-child composition, and another of one-parent-many-children composition. After some preliminaries in Section 3, in Section 4 we give an overview of the Maude-NPA tool and its operational semantics. In Section 5 we describe the new composition syntax and semantics. In Section 6 we describe the operational semantics of composition and the protocol transformation and give soundness and completeness results. In Section 7 we conclude the paper and discuss related and future work.

2 Two Motivating Examples

In both of our examples we build on the well-known Needham-Schroeder-Lowe (NSL) protocol [19]. The first example of protocol composition, which appeared in [16], is an example of a one-parent, one-child protocol, which is subject to an unexpected attack not noticed before. In this protocol, the participants use NSL to agree on a secret nonce. We reproduce the NSL protocol below.

1. $A \rightarrow B : \{N_A, A\}_{pk(B)}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{pk(A)}$
3. $A \rightarrow B : \{N_B\}_{pk(B)}$

where $\{M\}_{pk(A)}$ means message M encrypted using the public key of principal with name A, N_A and N_B are nonces generated by the respective principals, and we use the comma as message concatenation.

The agreed nonce N_A is then used in a distance bounding protocol. This is a type of protocol, originally proposed by Desmedt [7] for smart cards, which has received new interest in recent years for its possible application in wireless environments [3]. The idea behind the protocol is that Bob uses the round trip time of a challenge-response protocol with Alice to compute an upper bound on her distance from him in the following protocol.

4. $B \rightarrow A : N'_B$
   Bob records the time at which he sent N'_B
5. $A \rightarrow B : N_A \oplus N'_B$
   Bob records the time he receives the response and checks the equivalence $N_A = N_A \oplus N'_B \oplus N'_B$. If it is equal, he uses the round-trip time of his challenge and response to estimate his distance from Alice.

where $\oplus$ is the exclusive-or operator satisfying associativity (i.e., $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$) and commutativity (i.e., $X \oplus Y = Y \oplus X$) plus the properties $X \oplus X = 0$ and $X \oplus 0 = X$. Note that Bob is the initiator and Alice is the responder of the distance bounding protocol, in contrast to the NSL protocol.

This protocol must satisfy two requirements. The first is that it must guarantee that $N_A \oplus N'_B$ was sent after N'_B was received, or Alice will be able to pretend that she is closer than she is. Note that if Alice and Bob do not agree on N_A beforehand, then Alice will be able to mount the following attack: $B \rightarrow A : N'_B$ and then $A \rightarrow B : N$. Of course, $N = N'_B \oplus X$ for some X. But Bob has no way of telling if Alice computed N using N'_B and X, or if she just sent a random N. Using NSL to agree on a $X = N_A$ in advance prevents this type of attack.

Bob also needs to know that the response comes from whom it is supposed to be from. In particular, an attacker should not be able to impersonate Alice. Using NSL to agree on N_A guarantees that only Alice and Bob can know N_A, so the attacker cannot impersonate Alice. However, it should also be the case that an attacker cannot pass off Alice's response as his own. But this is not the case for the NSL distance bounding protocol, which is subject to the following attack[1]:

a) Intruder I runs an instance of NSL with Alice as the initiator and I as the responder, obtaining a nonce N_A.
b) I then runs an instance of NSL with Bob with I as the initiator and Bob as the responder, using N_A as the initiator nonce.
c) $B \rightarrow I : N'_B$ where I does not respond, but Alice, seeing this, thinks it is for her.

[1] This is not meant as a denigration of [16], whose main focus is on timing models in strand spaces, not the design of distance bounding protocols.

d) $A \to I : N'_B \oplus N_A$ where Bob, seeing this thinks this is I's response.

If Alice is closer to Bob than I is, then I can use this attack to appear closer to Bob than he is. This attack is a textbook example of a composition failure. NSL has all the properties of a good key distribution protocol, but fails to provide all the guarantees that are needed by the distance bounding protocol. However, in this case we can fix the problem, not by changing NSL, but by changing the distance bounding protocol so that it provides a stronger guarantee:

4. $B \to A : \{N'_B\}$
5. $A \to B : \{h(N_A, A) \oplus N'_B\}$ where h is a collision-resistant hash function.

As we show in our analysis (not included in this paper but available online at `http://www.dsic.upv.es/~ssantiago/composition.html`), this prevents the attack. I cannot pass off Alice's nonce as his own because it is now bound to her name.

The distance bounding example is a case of a one parent, one child protocol composition. Each instance of the parent NSL protocol can have only one child distance bounding protocol, since the distance bounding protocol depends upon the assumption that N_A is known only by A and B. But because the distance bounding protocol reveals N_A, it cannot be used with the same N_A more than once.

Our next example is a one parent, many children composition, also using NSL. This type of composition arises, for example, in key distribution protocols in which the parent protocol is used to generate a master key, and the child protocol is used to generate a session key. In this case, one wants to be able to run an arbitrary number of child protocols.

In the distance bounding example the initiator of the distance bounding protocol was always the child of the responder of the NSL protocol and vice versa. In the key distribution example, the initiator of the session key protocol can be the child of either the initiator or responder of the NSL protocol. So, we have two possible child executions after NSL:

4. $A \to B : \{Sk_A\}_{h(N_A,N_B)}$
5. $B \to A : \{Sk_A; N'_B\}_{h(N_A,N_B)}$
6. $A \to B : \{N'_B\}_{h(N_A,N_B)}$

4. $B \to A : \{Sk_B\}_{h(N_A,N_B)}$
5. $A \to B : \{Sk_B; N'_A\}_{h(N_A,N_B)}$
6. $B \to A : \{N'_A\}_{h(N_A,N_B)}$

where Sk_A is the session key generated by principal A and h is again a collision-resistant hash function.

These two examples give a flavor for the variants of sequential composition that are used in constructing cryptographic protocols. A single parent instance can have either many children instances, or be constrained to only one. Likewise, parent roles can determine child roles, or child roles can be unconstrained. In this paper we will show how all these types of composition are specified and analyzed in Maude-NPA, using these examples as running illustrations.

3 Background on Term Rewriting

We follow the classical notation and terminology from [22] for term rewriting and from [20,21] for rewriting logic and order-sorted notions. We assume an *order-sorted signature* Σ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ denotes the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. We write $\mathcal{V}ar(t)$ for the set of variables present in a term t. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_\Sigma(t)$. The subterm of t at position p is $t|_p$, and $t[u]_p$ is the result of replacing $t|_p$ by u in t. A *substitution* σ is a sort-preserving mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, $t' \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}'}$, and s and s' are sorts in the same connected component of the poset $(\mathsf{S}, \leq)$. For a set E of Σ-equations, an E-*unifier* for a Σ-equation $t = t'$ is a substitution σ s.t. $\sigma(t) =_E \sigma(t')$. A *complete* set of E-unifiers of an equation $t = t'$ is written $CSU_E(t = t')$. We say $CSU_E(t = t')$ is *finitary* if it contains a finite number of E-unifiers. A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The rewriting relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \xrightarrow{p}_{R,E} t'$ (or $\to_{R,E}$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $t|_p =_E \sigma(l)$, and $t' = t[\sigma(r)]_p$ for some σ. Assuming that E has a finitary and complete unification algorithm, the narrowing relation $\rightsquigarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\rightsquigarrow}_{\sigma,R,E} t'$ (or $\rightsquigarrow_{\sigma,R,E}$, $\rightsquigarrow_{R,E}$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $\sigma \in CSU_E(t|_p = l)$, and $t' = \sigma(t[r]_p)$.

4 Maude-NPA's Execution Model

Given a protocol $\mathcal{P}$, a *state* in the protocol execution is a term t of sort State, $t \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)_\mathsf{State}$, where $\Sigma_\mathcal{P}$ is the signature defining the sorts and function symbols for the cryptographic functions and for all the state constructor symbols, and $E_\mathcal{P}$ is a set of equations specifying the *algebraic properties* of the cryptographic functions and the state constructors. A protocol $\mathcal{P}$ is specified with a notation derived from strand spaces [13]. In a *strand*, a local execution of a protocol by a principal is indicated by a sequence of messages $[msg_1^-, msg_2^+, msg_3^-, \ldots, msg_{k-1}^-, msg_k^+]$ where each msg_i is a term of sort Msg (i.e., $msg_i \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)_\mathsf{Msg}$). Strand items representing input messages are assigned a negative sign, and strand items representing output messages are assigned a positive sign. For each positive message msg_i in a sequence of messages $[msg_1^\pm, msg_2^\pm, msg_3^\pm, \ldots, msg_i^+, \ldots, msg_{k-1}^\pm, msg_k^\pm]$ the non-fresh variables (see below) occurring in an output message msg_i^+ must appear in previous messages $msg_1, msg_2, msg_3, \ldots, msg_{i-1}$. In Maude-NPA [10,11], strands evolve over time and thus we use the symbol | to divide past and future in a strand, i.e., $[nil, msg_1^\pm, \ldots, msg_{j-1}^\pm \mid msg_j^\pm, msg_{j+1}^\pm, \ldots, msg_k^\pm, nil]$, where $msg_1^\pm, \ldots,$

$msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message). The nils are present so that the bar may be placed at the beginning or end of the strand if necessary. A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is a shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. We often remove the nils for clarity, except when there is nothing else between the vertical bar and the beginning or end of a strand. We write $\mathcal{P}$ for the set of strands in a protocol, including the strands that describe the intruder's behavior.

Maude-NPA uses a special sort Msg of messages that allows the protocol specifier to describe other sorts as subsorts of the top sort Msg. The specifier can make use of a special sort Fresh in the protocol-specific signature Σ for representing fresh unguessable values, e.g., nonces. The meaning of a variable of sort Fresh is that it will never be instantiated by an E-unifier generated during the backwards reachability analysis. This ensures that if two nonces are represented using different variables of sort Fresh, they will never be identified and no approximation for nonces is necessary. We make explicit the Fresh variables $r_1, \ldots, r_k (k \geq 0)$ generated by a strand by writing $:: r_1, \ldots, r_k :: [msg_1^{\pm}, \ldots, msg_n^{\pm}]$, where $r_1, \ldots, r_k$ appear somewhere in $msg_1^{\pm}, \ldots, msg_n^{\pm}$. Fresh variables generated by a strand are unique to that strand.

A *state* is a set of Maude-NPA strands unioned together by an associative and commutativity union operator _&_ with identity operator $\emptyset$, along with an additional term describing the intruder knowledge at that point. The *intruder knowledge* is represented as a set of facts unioned together with an associative and commutativity union operator _,_ with identity operator $\emptyset$. There are two kinds of intruder facts: positive knowledge facts (the intruder knows message m, i.e., $m \in \mathcal{I}$), and negative knowledge facts (the intruder does not yet know m but will know it in a future state, i.e., $m \notin \mathcal{I}$).

In the case in which new strands are not introduced into the state, the rewrite rules $R_{\mathcal{P}}$ obtained from the protocol strands $\mathcal{P}$ are as follows[2], where L, L_1, L_2 denote lists of input and output messages $(+m, -m)$, IK, IK' denote sets of intruder facts ($m \in \mathcal{I}$,$m \notin \mathcal{I}$), and SS, SS' denote sets of strands:

$$SS \;\&\; [L \mid M^-, L'] \;\&\; (M \in \mathcal{I}, IK) \rightarrow SS \;\&\; [L, M^- \mid L'] \;\&\; (M \in \mathcal{I}, IK) \quad (1)$$

$$SS \;\&\; [L \mid M^+, L'] \;\&\; IK \rightarrow SS \;\&\; [L, M^+ \mid L'] \;\&\; IK \quad (2)$$

$$SS \;\&\; [L \mid M^+, L'] \;\&\; (M \notin \mathcal{I}, IK) \rightarrow SS \;\&\; [L, M^+ \mid L'] \;\&\; (M \in \mathcal{I}, IK) \quad (3)$$

In a *forward execution* of the protocol strands, Rule (1) synchronizes an input message with a message already in the channel (i.e., learned by the intruder), Rule (2) accepts output messages but the intruder's knowledge is not increased, and Rule (3) accepts output messages and the intruder's knowledge is positively increased. Note that Rule (3) makes explicit *when* the intruder learned a message M, which is recorded in the previous state by the negative fact $M \notin \mathcal{I}$. A fact $M \notin \mathcal{I}$ can be paraphrased as: "the intruder does not yet know M, but will learn it in the future".

[2] Note that to simplify the exposition, we omit the fresh variables at the beginning of each strand in a rewrite rule.

New strands are added to the state by explicit introduction through dedicated rewrite rules (one for each honest or intruder strand). It is also the case that when we are performing a backwards search, only the strands that we are searching for are listed explicitly, and extra strands necessary to reach an initial state are dynamically added. Thus, when we want to introduce new strands into the explicit description of the state, we need to describe additional rules for doing that, as follows:

$$\text{for each } [\ l_1,\ u^+,\ l_2\ ] \in \mathcal{P} : SS\ \&\ [\ l_1 \mid u^+, l_2\ ]\ \&\ (u \notin \mathcal{I}, IK) \rightarrow SS\ \&\ (u \in \mathcal{I}, IK) \quad (4)$$

where u denotes a message, l_1, l_2 denote lists of input and output messages ($+m$,$-m$), IK denotes a set of intruder facts ($m \in \mathcal{I}$,$m \notin \mathcal{I}$), and SS denotes a set of strands. For example, intruder concatenation of two learned messages is described as follows:

$$SS\ \&\ [M_1^-, M_2^-\ \mid (M_1; M_2)^+]\ \&\ ((M_1; M_2) \notin \mathcal{I}, IK) \rightarrow SS\ \&\ ((M_1; M_2) \in \mathcal{I}, IK)$$

In summary, for a protocol $\mathcal{P}$, the set of rewrite rules obtained from the protocol strands that are used for backwards narrowing reachability analysis *modulo* the equational properties $E_{\mathcal{P}}$ is $R_{\mathcal{P}} = \{(1), (2), (3)\} \cup (4)$.

5 Syntax for Protocol Specification and Composition

We begin by describing the new syntactic features we need to make explicit in each protocol to later define sequential protocol compositions. Each strand in a protocol specification in the Maude-NPA is now extended with *input parameters* and *output parameters*. Input parameters are a sequence of variables of different sorts placed at the beginning of a strand. Output parameters are a sequence of terms placed at the end of a strand. Any variable contained in an output parameter must appear either in the body of the strand, or as an input parameter. The strand notation we will now use is $[\{\overrightarrow{I}\}, \overrightarrow{M}, \{\overrightarrow{O}\}]$ where $\overrightarrow{I}$ is a list of input parameter variables, $\overrightarrow{M}$ is a list of positive and negative terms in the strand notation of the Maude-NPA, and $\overrightarrow{O}$ is a list of output terms all of whose variables appear in $\overrightarrow{M}$ or $\overrightarrow{I}$. The input and output parameters describe the exact assumptions about each principal.

In the following, we first describe our syntax for protocol specification and then introduce a new syntax for protocol composition. Similarly to the Maude syntax for modules, we define a protocol modularly as follows:

```
prot Name is sorts Sorts . subsorts Subsorts .
    Operators Variables Equations DYStrands Strands
endp
```

where *Name* is a valid Maude module name, *Sorts* is a valid Maude-NPA declaration of sorts, *Subsorts* is a valid Maude-NPA declaration of subsorts, *Operators* is a valid Maude-NPA declaration of operators, *Variables* is a valid Maude-NPA declaration of variables to be used in the equational properties and in the

honest and Dolev-Yao strands, *Equations* is a valid Maude-NPA declaration of equational properties, *DYStrands* is a sequence of valid Maude-NPA Dolev-Yao strands, each starting with the word `DYstrand` and ending with a period, and *Strands* is a sequence of valid Maude-NPA strands, each starting with the word `strand` and ending with a period. The Maude-NPA protocol specifications of all the examples can be found in `http://www.dsic.upv.es/~ssantiago/composition.html`.

Example 1. The following description of the NSL protocol contains more technical details than the informal description of NSL in Section 2. A nonce generated by principal A is denoted by $n(A, r)$, where r is a unique variable of sort Fresh. Concatenation of two messages, e.g., N_A and N_B, is denoted by the operator $\_;\_$, e.g., $n(A, r)\ ;\ n(B, r')$. Encryption of a message M with the public key K_A of principal A is denoted by $pk(A, M)$, e.g., $\{N_B\}_{pk(B)}$ is denoted by $pk(B, n(B, r'))$. Encryption with a secret key is denoted by $sk(A, M)$. The public/private encryption cancellation properties are described using the equations $pk(X, sk(X, Z)) = Z$ and $sk(X, pk(X, Z)) = Z$. The two strands $\mathcal{P}$ associated to the three protocol steps shown above are as follows:

```
strand [NSL.init] :: r :: [ {A,B} |
 +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)), {A,B,n(A,r),N}] .
strand [NSL.resp] :: r :: [ {A,B} |
 -(pk(B,N;A)), +(pk(A,N;n(B,r);B)), -(pk(B,n(B,r))), {A,B,N,n(B,r)}] .
```

Note that we allow each honest or Dolev-Yao strand to be *labeled* (e.g. `init` or `resp`), in contrast to the standard Maude-NPA syntax for strands. These strand labels play an important role in our protocol composition method as explained below.

Example 2. Similarly to the NSL protocol, there are several technical details missing in the previous informal description of DB. The exclusive-or operator is $\_\oplus\_$ and its equational properties are described using associativity and commutativity of $\oplus$ plus the equations[3] $X \oplus 0 = X$, $X \oplus X = 0$, and $X \oplus X \oplus Y = Y$. Since Maude-NPA does not yet include timestamps, we do not include all the actions relevant to calculating time intervals, sending timestamps, and checking them. The two strands $\mathcal{P}$ associated to the three protocol steps shown above are as follows:

```
strand [DB.init] :: r :: [{A,B,NA} |
 +(n(B,r)), -(n(B,r)*NA),  {A,B,NA,n(B,r)}] .
strand [DB.resp] :: nil :: [{A,B,NA} | -(NB), +(NB * NA), {A,B,NA,NB}] .
```

In this protocol specification, it is made clear that the nonce N_A used by the initiator is a parameter and is never generated by A during the run of DB. However, the initiator B does generate a new nonce.

[3] Note that the redundant equational property $X \oplus X \oplus Y = Y$ is necessary in Maude-NPA for coherence purposes; see [11].

Example 3. The previous informal description of the KD protocol also lacks several technical details, which we supply here. Encryption of a message M with key K is denoted by $e(K, M)$, e.g., $\{N'_B\}_{h(N_A,N_B)}$ is denoted by $e(h(n(A,r), n(B,r')), n(B,r''))$. Cancellation properties of encryption and decryption are described using the equations $e(X, d(X, Z)) = Z$ and $d(X, e(X, Z)) = Z$. Session keys are written $skey(A, r)$, where A is the principal's name and r is a Fresh variable. The two strands $\mathcal{P}$ associated to the KD protocol steps shown above are as follows:

```
strand [KD.init] :: r :: [ {A,B,K} | +(e(K,skey(A,r)),
  -(e(K,skey(A,r) ; N)), +(e(K, N)), {A,B,K,skey(A,r),N}] .
strand [KD.resp] :: r :: [ {A,B,K} | -(e(K,SK)), +(e(K,SK ; n(B,r))),
  -(e(K,n(B,r))), {A,B,K,SK,n(B,r)} ] .
```

Sequential composition of two strands describes a situation in which one strand (the child strand), can only execute after the parent strand has completed execution. Each composition of two strands is obtained by matching the output parameters of the parent strand with the input parameters of the child strand in a user-defined way. Note that it may be possible for a single parent strand to have more than one child strand.

The relevant fact in the DB protocol is that both nonces are required to be unknown to an attacker before they are sent, but the nonce originating from the responder must be previously agreed upon between the two principals. Therefore, this protocol is usually composed with another protocol ensuring secrecy and authentication of nonces. Furthermore, according to [16], there are two extra issues related to the DB protocol that must be considered: (i) the initiator of the previous protocol plays the role of the responder in DB and viceversa, and (ii) nonces generated by the parent protocol cannot be shared by more than one child so that an initiator of NSL will be connected to one and only one responder of DB. In our working example, we use the NSL protocol to provide these capabilities.

Similarly to the syntax for protocols, we define protocol composition as follows[4]:

`prot` *Name* `is` *Name1* ; *Name2*
$a_1\{\overrightarrow{O_1}\}$; $\{\overrightarrow{I_1}\}b_1$ [1-1] (or [1-*]) . $\cdots$ $a_n\{\overrightarrow{O_n}\}$; $\{\overrightarrow{I_n}\}b_n$ [1-1] (or [1-*]) .
`endp`

where *Name* is a valid Maude-NPA module name, *Name1* and *Name2* are protocol names previously defined, $a_1, \ldots, a_n$ are labels of strands in protocol *Name1*, and $b_1, \ldots, b_n$ are labels of strands in protocol *Name2*. Furthermore, for each composition $a_i\{\overrightarrow{O_i}\}; \{\overrightarrow{I_i}\}b_i$, strand definition $:: \overrightarrow{r_{a_i}} :: [\{\overrightarrow{I_{a_i}}\}, \overrightarrow{a_i}, \{\overrightarrow{O_{a_i}}\}]$ for role a_i, and strand definition $:: \overrightarrow{r_{b_i}} :: [\{\overrightarrow{I_{b_i}}\}, \overrightarrow{b_i}, \{\overrightarrow{O_{b_i}}\}]$ for role b_i, we have that:

1. variables are properly renamed, i.e. $V_{ab} = \mathcal{V}ar(\overrightarrow{I_i}) \cup \mathcal{V}ar(\overrightarrow{O_i})$, $V_a = \mathcal{V}ar(\overrightarrow{I_{a_i}}) \cup \mathcal{V}ar(\overrightarrow{O_{a_i}})$, $V_b = \mathcal{V}ar(\overrightarrow{I_{b_i}}) \cup \mathcal{V}ar(\overrightarrow{O_{b_i}})$, and $V_{ab} \cap V_a \cap V_b = \emptyset$;
2. the variables of $\overrightarrow{I_i}$ must appear in $\overrightarrow{O_i}$ (no extra variables are allowed in a protocol composition);

[4] Operator and sort renaming is indeed necessary, as in the Maude module importation language, but we do not consider those details in this paper.

3. the formal output parameters $\overrightarrow{O_{a_i}}$ must match the actual output parameters $\overrightarrow{O_i}$, i.e., $\exists \sigma_a$ s.t. $\overrightarrow{O_{a_i}} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O_i})$; and
4. the actual input parameters $\overrightarrow{I_i}$ must match the formal input parameters $\overrightarrow{I_{b_i}}$, i.e., $\exists \sigma_b$ s.t. $\overrightarrow{I_i} =_{E_\mathcal{P}} \sigma_b(\overrightarrow{I_{b_i}})$.

The expressions [1−1] (or [1−∗]) indicate whether a one-to-one (or a one-to-many) composition is desired for those two strands. Note that for each composition, if there are substitutions σ_a and σ_b as described above, then there is a substitution σ_{ab} combining both, i.e., $\sigma_{ab}(X) = \sigma_a(\sigma_b(X))$ for any variable X, and then $\sigma_a(\overrightarrow{I_i}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_{b_i}})$. This ensures that any protocol composition is feasible and avoids the possibility of failing protocol compositions.

Let us consider again our two NSL and DB protocols and their composition. Note that we do not have to modify either the NSL or the DB specification above. The composition of both protocols is specified as follows:

```
prot NSL-DB is NSL ; DB
  NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
  NSL.resp {A,B,NA,NB} ; {A,B,NA} DB.init [1-1] .
endp
```

Let us now consider the NSL and KD protocols and their composition. The composition of both protocols, which is an example of a one-to-many composition, is specified as follows:

```
prot NSL-KD is NSL ; KB
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
endp
```

In the remainder of this paper we remove irrelevant parameters (i.e. input parameters for strands with no parents, and output parameters for strands with no children) in order to simplify the exposition.

6 Maude-NPA's Composition Execution Model

In this section we define a concrete execution model for the one-to-one and one-to-many protocol compositions by extending the Maude-NPA execution model. However, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics, and illustrate our results on our two running examples.

6.1 Composition Execution Model

As explained in Section 4, the operational semantics of protocol execution and analysis is based on rewrite rules denoting state transitions which are applied

for each one-to-one composition $\{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1]$ with
strand definition $[\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a, σ_{ab} s.t. $\overrightarrow{O_a} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_b})$,
we add the following rule :

$$\begin{aligned} & SS \,\&\, \overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \\ \rightarrow\; & SS \,\&\, \overrightarrow{a}, \{\overrightarrow{O_a}\} \mid nil] \;\&\; [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \end{aligned} \tag{5}$$

$$\begin{aligned} & SS \,\&\, \overrightarrow{a} \mid \{\overrightarrow{O}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \\ \rightarrow\; & SS \,\&\, [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \end{aligned} \tag{6}$$

Fig. 1. Semantics for one-to-one composition

modulo the algebraic properties $E_\mathcal{P}$ of the given protocol $\mathcal{P}$. Therefore, in the one-to-one and one-to-many cases we must add new state transition rules in order to deal with protocol composition. Maude-NPA performs backwards search modulo $E_\mathcal{P}$ by reversing the transition rules expressed in a forward way; see [10,11]. Again, we define forward rewrite rules which will happen to be executed in a backwards way.[5]

In the one-to-one composition, we add the state transition rules of Figure 1. Rule 5 composes a parent and a child strand already present in the current state. Rule 6 adds a parent strand to the current state and composes it with an existing child strand. Note that since a strand specification is a symbolic specification representing many concrete instances and the same applies to a composition of two protocol specifications, we need to relate actual and formal parameters of the protocol composition w.r.t. the two protocol specifications by using the substitutions σ_a and σ_{ab} in Figure 1. For example, given the following composition of the NSL-DB protocol

```
NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp were defined in Section 5, we add the following transition rule for Rule (5) where both the parent and the child strands are present and thus synchronized

```
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N))
        | {A,B,n(A,r),N} ]
:: nil :: [ nil | {A,B,n(A,r)}, -(NB), +(NB * n(A,r)) ] & SS & IK
->
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)),
```

[5] Note however that we represent unification explicitly via a substitution σ instead of implicitly via variable equality as in Section 4. This is because output and input parameters are not required to match, e.g. in the composition NSL-KD, the output parameters of the parent strand are $\{A, B, N_A, N_B\}$ whereas the input parameters of the child strand are $\{A, B, h(N_A, N_B)\}$.

for each one-to-many composition $\{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*]$ with
strand definition $[\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a, σ_{ab} s.t. $\overrightarrow{O_a} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_b})$,
we add one Rule 5, one Rule 6, and the following rule :

$$SS\,\&\,[\overrightarrow{a} \mid \{\overrightarrow{O_a}\}]\;\&\;[nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})]\,\&\,IK$$
$$\rightarrow SS\,\&\,[\overrightarrow{a} \mid \{\overrightarrow{O_a}\}]\;\&\;[\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})]\,\&\,IK \tag{7}$$

Fig. 2. Semantics for one-to-many composition

```
          {A,B,n(A,r),N} | nil ]
:: nil :: [{A,B,n(A,r)} | -(NB), +(NB * n(A,r)) ] & SS & IK
```

One-to-many composition uses the rules in Figure 1 for the first child plus an additional rule for subsequent children, described in Figure 2. Rule 7 composes a parent strand and a child strand but the bar in the parent strand is not moved, in order to allow further backwards child compositions. For example, given the following composition of the NSL-KD protocol

```
NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.resp and KD.init are as defined in Section 5, we add the following transition rule for Rule (7) using substitution $\sigma_{ab} = \{\texttt{A'} \mapsto \texttt{A}, \texttt{B'} \mapsto \texttt{B}, \texttt{K} \mapsto \texttt{h(NA,n(B,r))}\}$:

```
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r)))
        | {A,B,NA,n(B,r)}] .
:: r' :: [ nil | {A,B,h(NA,n(B,r))}, +(e(h(NA,n(B,r)),skey(A,r')),
          -(e(h(NA,n(B,r)),skey(A,r') ; N)), +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
->
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r))),
         {A,B,NA,n(B,r)} | nil ] .
:: r' :: [ {A,B,h(NA,n(B,r))} | +(e(h(NA,n(B,r)),skey(A,r')),
          -(e(h(NA,n(B,r)),skey(A,r') ; N)), +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
```

Thus, for a protocol composition $\mathcal{P}_1;\mathcal{P}_2$, the rewrite rules governing protocol execution are $R^{\circ}_{\mathcal{P}_1;\mathcal{P}_2} = \{(1), (2), (3)\} \cup (4) \cup (5) \cup (6) \cup (7)$.

6.2 Protocol Composition by Protocol Transformation

Instead of implementing a new version of the Maude-NPA generating new transition rules for each protocol composition, we have defined a *protocol transformation* that achieves the same effect using the current Maude-NPA tool.

The protocol transformation is given in Figure 3. Its output is a single, composed protocol specification where:

$$\Phi(\mathcal{P}_1;\mathcal{P}_2) = \begin{cases} \text{add strand } [\overrightarrow{a}, -(role_b(r)), +(role_a(r)\ .\ \sigma_{ab}(\dot{I}_b))] \text{ and} \\ \text{strand } [+(role_b(r)), -(role_a(r)\ .\ \dot{I}_b), \overrightarrow{b}\,] \\ \quad \text{whenever } \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \quad\quad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \quad\quad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \quad\quad \exists\sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_b}) \\ \quad\quad \text{and } r \text{ is a fresh variable} \\ \text{add strand } [\overrightarrow{a}, +(role_a(r)\ .\ \sigma_{ab}(\dot{I}_b))] \text{ and strand } [-(role_a(r)\ .\ \dot{I}_b), \overrightarrow{b}\,] \\ \quad \text{whenever } \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \quad\quad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \quad\quad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \quad\quad \exists\sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_b}) \\ \quad\quad \text{and } r \text{ is a fresh variable} \end{cases}$$

Fig. 3. Protocol Transformation

1. Sorts, symbols, and equational properties of both protocols are put together into a single specification[6]. Strands of both protocols are transformed and added to this single specification as described in Figure 3.
2. For each composition we transform the input parameters $\{\overrightarrow{I_b}\}$ into an input message exchange of the form $-(\overrightarrow{I_b})$, and the output parameters $\{\overrightarrow{O_a}\}$ into an output message exchange of the form $+(\sigma_{ab}(\overrightarrow{I_b}))$. The sort Param of these messages is disjoint from the sort Msg used by the protocol in the honest and intruder strands. This ensures that they are harmless, since no intruder strand will be able to use them. In order to avoid type conflicts, we use a *dot* for concatenation within protocol composition exchange messages, e.g. input parameters $\overrightarrow{I} = \{A, B, NA\}$ are transformed into the sequence $\dot{I} = A\ .\ B\ .\ NA$.
3. Each composition is uniquely identified by using a composition identifier (a variable of sort Fresh). Strands exchange such composition identifier by using input/output messages of the form $role_j(r)$, which make the role explicit. The sort Role of these messages is disjoint from the sorts Param and Msg.
   (a) In a one-to-one protocol composition, the child strand uniquely generates a fresh variable that is added to the area of fresh identifiers at the beginning of its strand specification. This fresh variable must be passed from the child to the parent before the parent generates its output parameters and sends it back again to the child.
   (b) In a one-to-many protocol composition, the parent strand uniquely generates a fresh variable that is passed to the child. Since an (a priori) unbounded number of children will be composed with it, no reply of the fresh variable is expected by the parent from the children.

[6] Note that we allow shared items but require the user to solve any possible conflict. Operator and sort renaming is an option, as in the Maude module importation language, but we do not consider those details in this paper.

For example, for the following one-to-one protocol composition in NSL-DB

```
NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp are as defined in Section 5 we have the following two transformed strands:

```
[NSL.init] :: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)),
                     -(db-resp(r#)), +(nsl-init(r#) . A . B . n(A,r)) ] .
[DB.resp] :: r# :: [ +(db-resp(r#)), -(nsl-init(r#) . A . B . NA),
                     -(NB), +(NB * NA) ] .
```

For the following one-to-many protocol composition in the NSL-KD

```
NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.resp and KD.init are as defined in Section 5 we have the following two transformed strands:

```
[NSL.resp] :: r,r# :: [-(pk(B,N;A)), +(pk(A,N;n(B,r);B)), -(pk(B,n(B,r))),
                       +(nsl-init(r#) . A . B . h(N,n(B,r))) ] .
[KD.init] :: r :: [ -(nsl-init(r#) . A . B . K), +(e(K,skey(A,r)),
                    -(e(K,skey(A,r) ; N')), +(e(K,N')) ] .
```

Soundness and completeness for this protocol transformation is provided in the following result. The proof of this theorem is available in [12]. Note that a state St is called *associated to a rewrite theory* $\mathcal{R}$ if it is a valid term w.r.t. the order-sorted signature of $\mathcal{R}$. We call a state *initial* if there are no backwards narrowing steps from it.

Theorem 1 (Soundness and Completeness). *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition, as defined in Section 5. Let $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the composition rewrite theory defined in Section 6.2, and let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the original Maude-NPA rewrite theory, as described in Section 6.1. Then there is a binary relation $\equiv_\Phi$ between the states associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ and the states associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ such that given St and St' associated to $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$, respectively, then $St \equiv_\Phi St'$ implies that there is an initial state In reachable from St by backwards narrowing in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ iff there is an initial state In' such that $In \equiv_\Phi In'$ and In' is reachable from St' by backwards narrowing in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$.*

We also have experimental results for protocol composition based on this theorem: (i) we found the attack for the NSL-DB protocol composition, (ii) we proved the security of the fixed version of the NSL-DB composition using a hash function, and (iii) we proved the security of NSL-KD. Due to space limitations, our experiments are not included here but they are available at `http://www.dsic.upv.es/~ssantiago/composition.html`.

7 Related Work and Conclusions

Our work addresses a somewhat different problem than most existing work on cryptographic protocol composition, which generally does not address model-checking. Indeed, to the best of our knowledge, most protocol analysis model-checking tools simply use hand-coded concatenation of protocol specifications

to express sequential composition. However, we believe that the problem we are addressing is an important one that tackles a widely acknowledged source of protocol complexity. For example, in the Internet Key Exchange Protocol [18] there are sixteen different one-to-many parent-child compositions of Phase One and Phase Two protocols. The ability to synthesize compositions automatically would greatly simplify the specification and analysis of protocols like these.

Now that we have a mechanism for synthesizing compositions, we are ready to revisit existing research on composing protocols and their properties and determine how we could best make use of it in our framework. There have been two approaches to this problem. One (called *nondestructive* composition in [6]) is to concentrate on properties of protocols and conditions on them that guarantee that properties satisfied separately are not violated by the composition. This is, for example, the approach taken by Gong and Syverson [14], Guttman and Thayer [17], Cortier and Delaune [5] and, in the computational model, Canetti's Universal Composability [2]. The conditions in this case are usually ones that can be verified syntactically, so Maude-NPA, or any other model checker, would not be of much assistance here.

Of more interest to us is the research that addresses the compositionality of the protocol properties themselves (called *additive* composition in [6]). This addresses the development of logical systems and tools such as CPL, PDL, and CPSA cited earlier in this paper, in which inference rules are provided for deriving complex properties of a protocol from simpler ones. Since these are pure logical systems, they necessarily start from very basic statements concerning, for example, what a principal can derive when it receives a message. But there is no reason why the properties of the component protocols could not be derived using model checking, and then composed using the logic. This would give us the benefits of both model checking (for finding errors and debugging), and logical derivations (for building complex systems out of simple components), allowing to switch between one and the other as needed. Indeed, Maude-NPA is well positioned in that respect. For example, the notion of state in strand spaces that it uses is very similar to that used by PDL [4], and we have already developed a simple property language that allows us to translate the "shapes" produced by CPSA into Maude-NPA attack states. The next step in our research will be to investigate the connection more closely from the point of view of compositionality.

References

1. Anlauff, M., Pavlovic, D., Waldinger, R., Westfold, S.: Proving authentication properties in the protocol derivation assistant. In: Proc. of Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (2006)
2. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC, pp. 494–503 (2002)
3. Capkun, S., Hubaux, J.P.: Secure positioning in wireless networks. IEEE Journal on Selected Areas in Communication 24(2) (February 2006)
4. Cervesato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key establishment protocols. In: IEEE Computer Security Foundations Workshop (2005)

5. Cortier, V., Delaune, S.: Safely composing security protocols. Formal Methods in System Design 34(1), 1–36 (2009)
6. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Secure protocol composition. In: Proc. Mathematical Foundations of Programming Semantics. ENTCS, vol. 83 (2003)
7. Desmedt, Y.: Major security problems with the "unforgeable" (Feige-)Fiat-Shamir Proofs of identity and how to overcome them. In: Securicom 88, 6th Worldwide Congress on Computer and Communications Security and Protection, Paris, France, March 1988, pp. 147–159 (1988)
8. Doghim, S., Guttman, J., Thayer, F.J.: Searching for Shapes in Cryptographic Protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007)
9. Durgin, N., Mitchell, J., Pavlovic, D.: A Compositional Logic for Program Correctness. In: Fifteenth Computer Security Foundations Workshop — CSFW-14, Cape Breton, NS, Canada, June 11-13, IEEE Computer Society Press, Los Alamitos (2001)
10. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. Theor. Comput. Sci. 367(1-2), 162–202 (2006)
11. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: FOSAD 2008/2009 Tutorial Lectures, vol. 5705, pp. 1–50. Springer, Heidelberg (2009)
12. Escobar, S., Meadows, C., Meseguer, J., Santiago, S.: Sequential Protocol Composition in Maude-NPA. Technical Report DSIC-II/06/10, Universidad Politécnica de Valencia (June 2010)
13. Thayer Fabrega, F.J., Herzog, J., Guttman, J.: Strand Spaces: What Makes a Security Protocol Correct? Journal of Computer Security 7, 191–230 (1999)
14. Gong, L., Syverson, P.: Fail-stop protocols: An approach to designing secure protocols. In: Proc. of the 5th IFIP International Working Conference on Dependable Computing for Critical Applications, pp. 79–99. IEEE Computer Society Press, Los Alamitos (1998)
15. Guttman, J.: Security protocol design via authentication tests. In: Proc. Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2001)
16. Guttman, J.D., Herzog, J.C., Swarup, V., Thayer, F.J.: Strand spaces: From key exchange to secure location. In: Workshop on Event-Based Semantics (2008)
17. Guttman, J.D., Thayer, F.J.: Protocol independence through disjoint encryption. In: CSFW, pp. 24–34 (2000)
18. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE), IETF RFC 2409 (November 1998)
19. Lowe, G.: Breaking and fixing the Needham-Schroeder public key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
20. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
21. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998)
22. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)

Verifying Security Property of Peer-to-Peer Systems Using CSP

Tien Tuan Anh Dinh and Mark Ryan

School of Computer Science,
University of Birmingham, Birmingham
United Kingdom, B15 2TT
{ttd,mdr}@cs.bham.ac.uk

Abstract. Due to their nature, Peer-to-Peer (P2P) systems are subject to a wide range of security issues. In this paper, we focus on a specific security property, called the *root authenticity* (or RA) property of the so-called structured P2P overlays. We propose a P2P architecture that uses Trusted Computing as the security mechanism. We formalize that system using a process algebra (CSP), then verify that it indeed meets the RA property.

Keywords: Peer-to-Peer, Trusted Computing, formal verification, CSP.

1 Introduction

Peer-to-Peer (P2P) overlays are *heterogeneous* systems that consists of *autonomous* peers (or nodes), in which most traffic is among peers. The most well-known P2P applications, file-sharings [1,2], are built on top of *unstructured overlays*. In these overlays, peers form a random topology, i.e. a peer can connect to any other peer. Searching is done by broadcasting the query to neighboring nodes (which is not scalable). Unstructured overlays support approximate search, i.e. searching for items *close* to the search key k. They are suitable for dynamic environments (where peers leave and join frequently).

In this paper, we focus on *structured P2P overlays*, in which peers form rigid topologies, i.e. a node only connects to a certain set of neighbors. Examples are Chord [3], Pastry [4], etc. It is necessary to update the topology when nodes join or leave the network, which is an expensive operation. Exact-match searching is done deterministically and is very efficient. In many overlays, it takes $O(logN)$ hops, where N is the number of peers. On the one hand, structured overlays are restricted to relatively stable environments where joining and leaving are infrequent. On the other hand, they are much more scalable than the unstructured counterpart, and therefore can support applications with very large numbers of participants. Existing applications of structured overlays range from global storage systems [5], P2P-based communications [6], application-level multicast [7], P2P-based marketplaces [8] to botnets [9].

Due to the decentralization nature, P2P systems are subject to a wide range of security attacks. They are normally caused by the lack of an identity management mechanism or of a central authority. In this work, we investigate an

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 319–339, 2010.

attack, called the *false-destination* attack, in which the adversary falsely claims that it is the destination of a search key k. We say that a P2P system satisfies the *root authenticity* (or RA) property if it is secure from this attack. In structured overlays, where the data key k is uniquely assigned to a *root node*, there are various reasons behind this attack. For example, in the P2P storage system, the adversary wishes to censor a piece of data identified by the key k. It will (falsely) claim to be the root node of k. As the consequence, all traffic regarding k (queries or deposit of the data) will be forwarded to the adversary, which now has total control of the data it wanted to censor.

Despite the early recognition of security problems in P2P systems, not much progress has been made in addressing them. Most related work propose solutions that are probabilistic. They normally do not scale well and incur great overhead when churn (nodes joining and leaving the network) is frequent. More importantly, there is a serious lack of formal studies of P2P security. For P2P overlays to be used in applications that demand a certain level of security, such formal studies are vital.

Contribution. Our main contributions from this paper are as follows:

1. We discuss security issues of P2P systems, categorizing those issues as inherently belonging to different levels of abstraction. This enables us to clarify how the RA property relates to the various sets of security problems.
2. We propose a P2P architecture aiming to satisfy the RA property. This system makes use of Trusted Computing as the underlying security mechanism.
3. We explain our formalization in CSP of the proposed system, and describe our approach to verifying that the system does indeed satisfy the RA property.

The detailed CSP model can be found in the Appendix. For the complete proof, see [10].

Related Work. Sit et al. [11] present a taxonomy of security attacks on structured P2P overlays. Castro et al [12] propose a secure routing mechanism based on a number of components: secure ID assignment, secure neighbor maintenance and secure message forwarding. Their solution is probabilistic and incurs large overhead.

Regarding the false destination attack, the closest to our work is that by Wang et al. [13]. Their solution assumes the existence of a certificate authority (or CA) that when a node joins issues certificates to $2.l + 1$ neighbors, where l is the number of closest neighbor to a peer in one direction (i.e. its *leafset*). This approach is probabilistic and its effectiveness increases with l. Ganesh et al. [14] propose another solution that assumes peers regular publishing their ID certificates. For verification, it relies on name-space density estimation, which is probabilistic.

Work on formalizing and verifying P2P systems are limited in numbers. Borgstrom et al. [15] model Distributed K-ary Search (DKS), a structured overlay in Calculus of Communicating Systems (CCS). They verify that the routing

protocol in DKS, under the static case (when no node joining or leaving), is correct. Bakhshi et al. [16] model Chord in π-calculus and verify that the stabilization protocol in Chord is correct.

Paper's Organization. In the next section, we discuss the range of security issues in P2P systems, focusing on the RA property. We show that RA is implied by another property, called the *neighbor authenticity* (NA) property. In Section 3, we introduce our P2P architecture built on top of Trusted Computing. We show details of the routing and churn protocols that make use of the Trusted Platform Modules (TPMs) running at each peer. In Section 4, we describe the CSP model of the above architecture and define (in CSP) the NA property. We then explain our approach for verifying that the current model satisfies the NA property in Section 5. Finally, we conclude and discuss future work in Section 6.

2 Security Issues in P2P Systems

2.1 Overview

Example of a Structured P2P System. There are a number of structured P2P systems, each differs from another in its topology, routing or maintenance protocols. In this paper, we consider Chord, one of the earliest structured P2P systems. Chord is more popular than the other systems, due to its simplicity and efficient routing protocol.

Let $\mathcal{P}$ and $\mathcal{D}$ be the set of peers and data objects. In Chord, members of both sets are *hashed* using a non-collision hash function into the same circular ID space $\mathcal{ID}$. Fig. 1 shows an example in which $\mathcal{ID} = [0, 2^{10})$ and there are 6 peers with IDs of $144, 296, 498, 609, 775$ and 1000. The two data objects are hashed to the value 550 and 744. Each peer in Chord connects to a peer immediate on

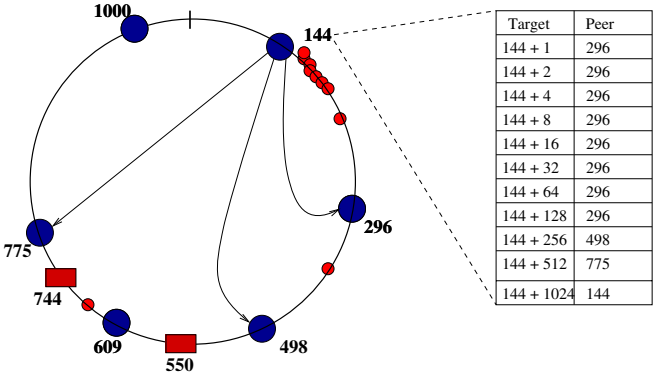


Fig. 1. Example of a Chord overlay. The big circles represent peers, the rectangles represent data objects, and the small circles represent the IDs that are used to construct the finger table of peer 144.

its left (*predecessor*) and on its right (*successor*). In Fig.1, the predecessor and successor of peer 144 are 1000 and 296 respectively.

Let *successor(k)* be the peer immediate on the right of the key k in the ID ring. In Chord, $successor(k)$ is the **destination node** of k, which is responsible for storing the data identified by the key k. For instance, $successor(300) = successor(400) = 498$ and $successor(250) = 296$. The data 744 is stored at peer 775. For efficient routing, each node p in Chord also maintains a *finger* table of size m, where 2^m is the size of $\mathcal{ID}$ and $finger[i] = successor(p + 2^{i-1})$ for $1 \leqslant i \leqslant m$. In the above example, the 4^{th} and 9^{th} finger of peer 144 points to node 296 and 498 respectively.

To search for $successor(k)$, the searching peer forwards its query to the neighbor, which is one of its successor, predecessor or finger pointers and is furthest from it but still on the left of k. The query is then executed at the new node, until the current node is the closest on the left of k. The search then stops and the current node's successor is returned. In Fig. 1, the routing path from node 144 for $successor(744)$ is $144 \rightarrow 498 \rightarrow 609$. Finally, 775 is returned as the destination of k.

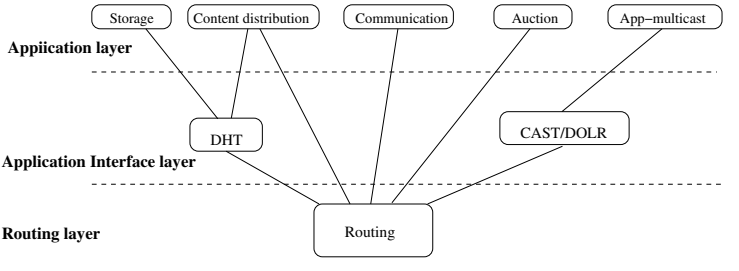


Fig. 2. Different levels of abstraction in P2P systems. Adapted from [17].

P2P Abstraction Layers. A P2P system can be studied at different levels of abstractions, as illustrated in Fig. 2:

+ *Routing layer*: implements the **route(k)** protocol, which, if successful, returns $successor(k)$. In *unstructured* P2P systems, this protocol is implemented by broadcasting the query to all neighbors. In *structured* P2P systems, this is implemented more efficiently by deterministically forwarding the query to neighbors closer to k.
+ *Application interface layer*: implements the **store(k,data)**, which places the tuple (k,data) at a node. In a Distributed Hash Table (DHT) system, for example, this tuple is stored at the destination node of k. In other systems, it is also stored at nodes along the routing path.
+ *Application layer*: consists of application-specific protocols that utilize the lower levels. For instance, P2P communication systems make use of the routing layer, whereas P2P storage systems use the application interface layer to store data in a scalable way.

Security Issues. Given the abstraction in Fig. 2, an adversary can perform attacks against a P2P system at more than one level. In the following, we discuss inherent security concerns at each level. The hierarchy in Fig. 2 suggests that to achieve security at one level, one must at first address the security issues at levels below it.

1. Routing layer: the adversary can corrupt the routing protocol. For example:
   + No routing: queries are dropped. As the consequence, the network is partitioned into parts that can not reach each other.
   + Redirection: queries are forwarded to malicious nodes. They could also be forwarded to innocent nodes, in attempt at a DDoS attack.
   + Impersonate the final node in the routing path
2. Application interface layer: the adversary can corrupt the store(k,data) protocol in the following ways:
   + Dropping the tuple (k,data) that is destined to stored at its node.
   + Tampering with the data.
3. Application layer: the adversary can compromise application-specific properties. For example:
   + Corrupting data: malware, etc.
   + Attacking other mechanisms such as replication, access control, etc.

2.2 Root Authenticity Property of a P2P System

As previously described, the adversary may attempt to impersonate the destination of a search key, called the false-destination attack. For example, in Fig.1, an adversary controlling peer 498 and 775 could convince peer 144 that node 775 is the destination for the data 550 (the correct destination, given the current configuration of the network, is node 609). The RA property implies that such an attack is not possible. In the following, we present a more formal definition of this property.

Let $\mathcal{ID} = [0, 2^m)$ for a reasonable large value of m. For any $x, y, z \in \mathcal{ID}$, denote $inBetween(z, x, y)$ as the predicate that indicates that going clock-wise from x one gets to z before y. More precisely:

$$inBetween(z, x, y) = \big(|z \ominus x| + |y \ominus z| = |y \ominus x| \big)$$

where $\oplus$ and $\ominus$ are addition and subtraction in *modulo* 2^m, and $|x \ominus y|$ is the function returning the clock-wise distance between y and x[1].

Let $\mathcal{P}$ be the set of peers in the network. We can refer to them by their unique IDs. Let V be an honest peer that searches for the destination of a key k. V can perform the following operations:

1. $V.route(k)$: the P2P routing protocol. We will not consider the details of the function *route*. In our analysis, we allow it to be any function returning a value in $\mathcal{P}$.

[1] $|x \ominus y| = t \;\Leftrightarrow\; 0 \leqslant t < 2^m \;\wedge\; y \oplus t = x.$

```
1. L <- V.getPredecessor(D);
2. if (V.neighborVerification(L,D))
      if (inBetween(k,L,D))
         return true;
3. return false;
```

Fig. 3. Details of the `V.destVerification(k,D)` protocol

2. $V.getPredecessor(D)$: V contacts D and asks for its predecessor. Similar to *route*, we allow $V.getPredecessor$ to be any function returning a value in $\mathcal{P}$.
3. $V.neighborVerification(L, R)$: for any $L, R \in \mathcal{P}$, V checks if L is the predecessor of R in the current network. The details of this protocol is the main focus of Section 3.
4. $V.destVerification(k, D)$: for any $k \in \mathcal{ID}$, V checks if D is the destination of k, given the current configuration of the network. The details of this protocol are shown in Fig.3.

Definition 1 (Root Authenticity (RA) Property). *Let $\mathcal{P}^t$ be the set of current nodes in the P2P system, at a given time t. Assume that the system evolves from t to $(t+1)$ as a new peer joins or an existing peer leaves the system. The RA property is defined as:*

$$\forall D, k \in \mathcal{ID}, t\,.\;\; V.destVerification(k, D) \Rightarrow\; D \in \mathcal{P}^t \;\wedge\; \left(\forall D' \in \mathcal{P}^t \setminus \{D\}\,.\, |D' \ominus k| > |D \ominus k|\right)$$

Definition 2 (Neighbor Authenticity (NA) Property). *Let $\mathcal{P}^t$ be the set of current nodes in the P2P system, at a given time t. Assume that the system evolves from t to $(t+1)$ as a new peer joins or an existing peer leaves the system. The NA property is defined as:*

$$\forall L, D, t\,.\;\; V.neighborVerification(L, D) \Rightarrow\; \{L, D\} \subseteq \mathcal{P}^t \;\wedge\; \left(\forall D' \in \mathcal{P}^t \setminus \{L\}\,.\, |D' \ominus L| \geqslant |D \ominus L|\right)$$

Informally speaking, the *RA* property requires that for any key k and a peer D at time t, if $destVerification(k, D)$ returns true then D is the closest peer on the right of k at time t. The *NA* property requires that at time t for any peer L and D in the network, if $neighborVerification(L, D)$ returns true then L is in fact the immediate left neighbor of D at time t. From these definitions, we have the following proposition:

Proposition 1. $NA \;\Rightarrow\; RA$

This theorem means that if the neighbor verification protocol is correct, then the system satisfies the RA property.

Why RA Property. In a system not satisfying the RA property, an adversary A can falsely convince an honest peer that it is the destination of a key k. There are various reasons behind such attacks. Since the data identified by k is stored at the destination of k, the attacker may launch this attack to gain control or censor a particular piece of data. In P2P storage systems, for example, the attacker having the data can modify or removing them from the system. In other applications where controlling more data could imply economic gains (such as P2P-based marketplaces), there are more tangible incentives for A to initiate the attack. In P2P-based communication systems (such as Voice Over IP (VOIP) or instant messaging), the data generally contains the connection details of the communicating hosts. Having control over such data means that A might be able to eavesdrop the communication, or even prevent it from happening. The impact cause by these attacks can be worsened by the adversary launching *Sybil* attacks, in which the adversary has many identities and therefore controlling multiple nodes at different locations in the network.

Recently, there are much research on reputation systems for P2P. One fundamental element of a reputation system is the feedback mechanism, by which a peer can rate the behavior of another. In many cases, this requires a peer being able to verify if another is telling the truth or not. The RA property implies that a peer can check if the result of route(k) is the correct destination, therefore it can confidently give good or bad feedback to the nodes involved in the routing path.

3 A Secure P2P System Using Trusted Computing

3.1 Trusted Computing and Trusted Platform Modules

Trusted Computing is a collection of current and future initiatives to root security in hardware that have been under development since about 2003. It is set to transform the computing security landscape over the next decade. Currently, the most noticeable manifestations are the *Trusted Platform Module* (TPM), Intel's *Trusted eXecution Technology* (TXT) and *Virtualisation Technology* (VT-d).

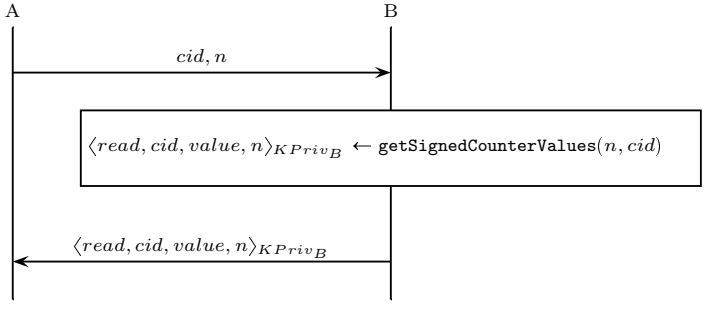
The TPM is a hardware chip currently shipped in high-end laptops, desktops and servers made by all the major manufacturers and destined to be in all devices within the next few years. It is specified by an industry consortium [18], and the specification is now an ISO standard [19]. There are currently 100M TPMs in existence as of 2008, and this figure is expected to be 250M by 2010 [20,21]. The TPM provides hardware-secured storage, secure platform integrity measurement and reporting, and platform authentication. Software that uses this functionality will be rolled out over the coming years. The TPM is commonly used for:

1. Secure storage. User processes can store content that is encrypted by keys only available to the TPM.
2. Platform measurement and reporting. A platform can create reports of its integrity and configuration state that can be relied on by a remote verifier.
3. Platform authentication. A platform can obtain keys by which it can authenticate itself reliably.

In the P2P context, we make use of the following set of the TPM features:

1. A TPM can create a public/private key pair $\langle KPriv, KPub \rangle$, called an *Attestation Identity Key* (AIK). The TPM can identify itself using the AIK, which can be certified using a certificate authority.
2. *Monotonic counters:* each TPM maintains a set of monotonic counters. On a counter with ID *cid*, one can perform the following operations:
   (a) `TPM_ReadCounter`(*cid*): returns the current value of the counter *cid*.
   (b) `TPM_IncrementCounter`(*cid*): increments and returns the new value of the counter *cid*.
3. *Transport sessions:* the TPM commands can be grouped and executed together within a transport session. The session can be *exclusive*, meaning that no other commands can be executed outside of the session when it is active. Furthermore, the session's log can be signed by the TPM.
   (a) `TPM_EstablishTransport`(*exc*): sets up a transport session. The flag *exc* determines if the session is exclusive. A session handle, *sHandle* is returned.
   (b) `TPM_ExecuteTransport`($comm, sHandle$): executes *comm*, which contains a *wrapped* TPMs command, inside the session *sHandle*.
   (c) `TPM_ReleaseTransportSigned`($n, sHandle$): closes the transport session and signs its log containing input, output of all the commands executed in $sHandle$. n is used as the non-replay nonce in the signature.

Example. As an example, suppose the TPM of an agent B has a counter *cid*, whose value is of interest to an agent A. Protocol 1 and Fig. 4 illustrates how A finds out the latest value of *cid*.



Protocol 1: A queries B for the latest value of its counter *cid*

First, A sends *cid* and a freshly generated nonce n to B, which then executes `getSignedCounterValues`(n, cid) on its local TPM. The `getSignedCounterValues`(n, cid) procedure, detailed in Fig.4, first establishes a transport session with the TPM, then executes `TPM_ReadCounter`(*cid*) within that session. Finally, the session is closed and the TPM's signature on the session's log is returned, in which n is used as the non-replay nonce. Notice that it is not possible for B to generate such a signature without having its TPM executing the `TPM_ReadCounter`(*cid*) command inside a transport session.

```
1. sHandle <- TPM_EstablishTransport(true)
2. wc <- wrap command TPM_ReadCounter(cid)
3. TPM_ExecuteTransport(wc, sHandle)
4. sig <- TPM_ReleaseTransportSigned(sHandle,n)
5. return sig
```

Fig. 4. `getSignedCounterValues`(n, cid) is executed by the local TPM at B

3.2 A Secure P2P System Using Trusted Computing

Assumptions. In our proposed P2P system, we assume that all peers have support for the trusted computing infrastructure. In particular, peers are equipped with TPMs. For a large-scale P2P system, one may question if this assumption is reasonable. We wish to stress that firstly, more computers are being shipped with trusted computing support. Secondly, our proposal could work with any other infrastructure that supports the features listed in Section 3.1. It could be in the form of smart-cards or online services. These alternatives could be better choices than TPMs due to their flexibility and wider range of trusted functionalities.

Regarding the *churn* model, we assume that peers leave the network gracefully. This means peers notify their neighbors (or other relevant entities) before exiting.

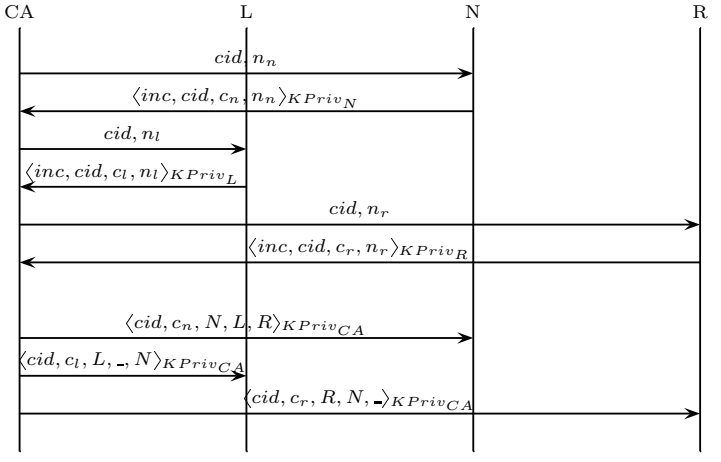
Certificate Authority. In our system, there exists a certificate authority (CA) which is trusted to issue *neighbor certificates* as peers join and leave the network. The CA does not have to run on trusted hardware. It acts as a single point of trust, but as discussed later, is unlikely to be a performance bottleneck.

The CA has an asymmetric key pair $\langle KPriv_{CA}, KPub_{CA}\rangle$. In the joining process, for example, a new peer N contacts CA to be issued a neighbor certificate, which is of the form:

$$\langle cid, v, N, L, R\rangle_{PrivK_{CA}}$$

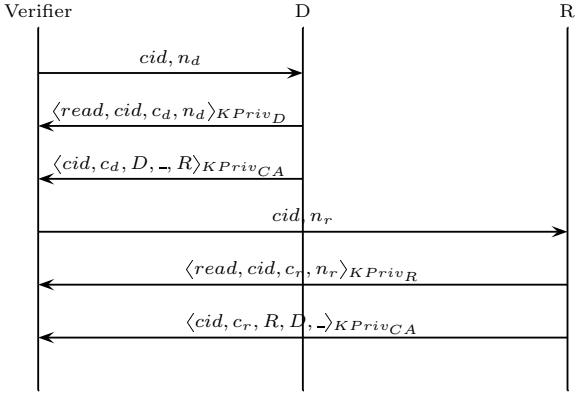
where v is the current value of the counter cid. L, R are the immediate left and right neighbors of N, at the moment the certificate is issued. These nodes also receive new certificates from the CA. It is important that the CA *knows* the correct immediate left and right neighbors of N at any given time in order to issue such certificates. There are several ways for the CA to acquire this knowledge. For simplicity, we assume that it maintains a list of peers currently in the network. When N joins, it checks that N is not already in the list, then issues the relevant certificates and adds N to the list. It performs the opposite when N leaves the network.

Joining/Leaving Protocol. Protocol 2 illustrates the protocol between the CA and other nodes when N joins the network. The CA knows that L, R are the immediate left and right neighbor of N in the current network. First, it asks N, L and R to increment their specific counters (cid). Once received the signatures on the new counter values, the CA adds N to its list of existing peers, then issues new certificates for N, L and R containing information of the new neighbors.



Protocol 2: Peer N joins in between L and R in the network

When a peer E leaves the network, the protocol is similar, except that the CA only issues certificates for E's current neighbors.



Protocol 3: Peer V verifies if R is the current right neighbor of peer D

Routing Protocol. Consider a peer V searching for the destination node of a key k. First, the normal P2P routing protocol (Chord or Pastry routing, for example) is used, which returns a peer D. Before *accepting* D as the destination of k, V performs the verification protocol with D, as illustrated in Protocol 3. The Verifier queries the latest value of D's counter cid, namely c_d. It then asks D for the certificate of C_d that contains c_d. By doing this, the Verifier is confident that C_d is the latest certificate issued by the CA to D.

C_d contains information of D's right neighbor, namely R. The Verifier then asks for R's latest certificate, C_r in the same way it did for D. The verification returns true if C_d and C_r match, meaning that in C_d, R is the right neighbor of D and in C_r, D is the left neighbor of R.

The reason for requiring certificates from both D and R is to avoid the following scenario. D is an adversary, it executed the joining protocol properly and has already left the network (gracefully). However, the routing protocol returns D, and since it is still online, D returns its out-of-date certificate, which would be accepted by the Verifier. In other words, D is accepted as the destination of k. This violates the RA property, which requires the destination node to be a node currently in the network.

Discussion. It can be seen from the description of the routing and joining protocols that the CA is a relatively *off-line* entity, since its only involvement is during churn events. The CA is not consulted during the routing process. In a typical P2P system, the rate of query-routing is considerably more frequent than the rate of churn. We therefore argue that the CA is unlikely to be a performance bottleneck.

More specifically, let M be the number of nodes in the network. Churn events can be modeled by a Poisson distribution. This means that a peer's *session time* (duration during which the peer stays in the network) follows an exponential distribution. Let c be the *churn rate*, so that the session times are exponentially distributed with the expected value of $\frac{1}{c}$. It then follows that the expected number of churn events that the CA has to deal with per time unit is $c.M$. For a large (but relatively stable) network, i.e. M is in order of millions and the average session time is in the order of days, $c.M$ is small enough so that the CA would not become a bottleneck.

In our current design, the CA maintains a list of peers currently in the network, which could be a concern. A typical computer nowadays can deal with M in the size of millions. For scalability, however, it will be better to relieve the CA from maintaining such a list. Instead, during the joining or leaving process, the CA also asks for the certificates of the joining or leaving peer as well as of its immediate neighbors. If the certificates match, the CA then issues new certificates as described early. We conjecture that if all the certificates before a churn event were issued correctly, then so are the new ones after the event is completed. We plan to investigate this system in future work.

Finally, the current churn model is quite strict, as we consider peers leaving gracefully, i.e. they notify the CA before leaving. We could make it more realistic by taking the *fail-stop* and *Byzantine* failure models into account. To deal with these failures, a *time-out* mechanism is needed that indicates when a certificate will expire. More specifically, peers need to contact the CA regularly to have their certificates renewed, or else they will be considered having left the network. This would imply more overhead for the CA, as it needs to issues more certificates and keeps track of which peers have left the network. Detailed investigation of the *time-out* mechanism is left for future work.

4 Formal Model in CSP

A brief introduction to CSP syntax and its semantic can be found in the Appendix. CSP has three denotational semantic models: *traces*, *stable failures* and *failures/divergences*. In this work, we only use the traces model, especially the *refinement* relation on traces. In particular, let $traces(P)$ and $traces(Q)$ be set of traces of the process P and Q, then Q is said to refine P, written as $P \sqsubseteq_T Q$ if:

$$traces(Q) \subseteq traces(P)$$

4.1 The System Model in CSP

The model consists of several agents, as shown in Fig.5

1. *Nonce Manager*: supplies fresh and unique nonces for other agents. These values are used during joining, leaving and verification to avoid replaying of old counter values. They are communicated by the NonceManager process to others via the *send* channel.
2. *TPM*: models the trusted hardware used in the system. Each TPM has a counter *cid* for the P2P operations. The counter ID is known to all peers. In addition, each TPM is identified by an unique ID, which can be used as the peer ID. Balfe et al. [22] propose a mechanism for enforcing *stable* IDs based on the Direct Anonymous Attestation (DAA) protocol. In our context, however, we could just assign the ID to be the public part of an AIK. During P2P operations, such ID shall be unique, even though more than one AIKs can be generated for a TPM. It is because the counter *cid* is unique in each TPM. If multiple IDs are used by a TPM, then updating the counter value of one ID will effectively invalidate the states of the other IDs.

   The CSP process representing a TPM receives nonces on the channel *receive*. It then sends back a signature on the latest counter value, after a

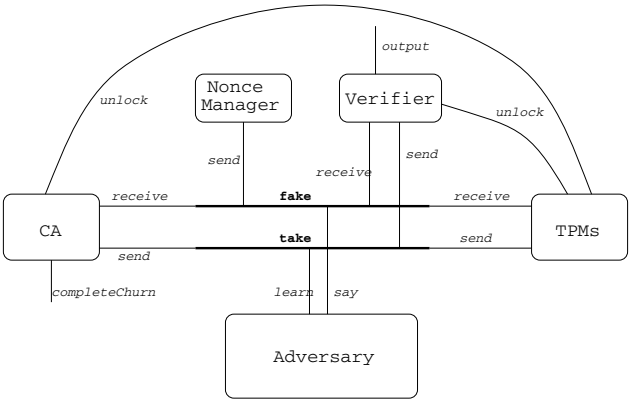


Fig. 5. Channels used by the processes

read or an increment operation. The signature is of the form of the event $SqR.\langle n, i, c\rangle$ (after a read operation) or $SqI.\langle n, i, c\rangle$ (after an increment operation) where n, i, c are the received nonce, the TPM's ID and the counter value.

3. *Certificate Authority*: issues certificates during churn events, as explained in the previous section. The CA process uses the *send* and *receive* channels for sending nonces and receiving other messages. Once it has issued all relevant certificates for a churn event, it outputs on the channel *completeChurn* to signal that the churn event has completed. For example, the event $completeChurn.Churn.\langle join, i\rangle$ where i is a peer indicates that i has successfully joined the network.
4. *Verifier*: picks a random peer D and asks it to returns its immediate right neighbor R. The Verifier performs the verification and then outputs whether it *accepts* that R is the immediate right neighbor of D. This basically models the `neighborVerification` protocol described in section 2. It follows from Proposition 1 that if the Verifier cannot be *fooled* then the RA property is met.

   The Verifier process sends nonces and receives other messages on the *send* and *receive* channel respectively. Events on the channel *output* indicates that the Verifier accepts that one peer is the immediate right neighbor of another. For instance, $output.L.R$ means it is convinced that R is the immediate right neighbor of L in the current network.
5. *Adversary*: models the attacker trying to break the RA property of the system. We give the adversary total control of all the peers in the network, i.e. it controls all the TPMs, even though it cannot fake signatures generated by the TPM.

   The Adversary process uses the *learn* and *say* channel to eavesdrop and send messages from and to the other agents. We model the Adversary being able to remember all messages it has seen, i.e. having infinite memory, so that it could replay old messages. It can be seen from Fig.5 that all the *send* and *receive* channels are renamed to *take* and $fake$, which are in turned mapped to the *learn* and *say* channels (by a many-to-one mapping function). This renaming scheme introduces non-determinism and as an effect increases the adversary's power.

The detailed CSP models for these agents can be found in the Appendix.

4.2 Checking the RA Property

As Proposition 1 indicates, the RA property is implied by the correctness of the neighbor verification protocol `neighborVerification`, which can be formalized by the following process:

$$
\begin{aligned}
Spec(ps, pn) = \quad & \Box_{i \in pn}\ completeChurn.Churn.\langle join, i\rangle \rightarrow Spec(ps \cup \{i\}, pn \backslash \{i\}) \\
\Box\ & \Box_{i \in ps}\ completeChurn.Churn.\langle leave, i\rangle \rightarrow Spec(ps \backslash \{i\}, pn \cup \{i\}) \\
\Box\ & \Box_{i \in ps}\ output.i.right(i,ps) \rightarrow Spec(ps, pn)
\end{aligned}
$$

ps, pn are the sets of peers currently in and not in the network respectively. These sets change after events in the *completeChurn* channels occur, i.e. after a churn event completed. The function $succ(i, ps)$ returns the immediate right neighbor of i in the set ps, which is defined as follows:

$$right(p, ps) = r \quad \text{if } r \in ps \ \wedge \ \forall p' \in ps \backslash \{r\} \,.\, |r \ominus p| \leqslant |r \ominus p'|$$

Let *System* be the CSP model of the entire system (the details can be found in the Appendix). To prove that the system satisfies the RA property is equivalent to showing the following:

$$Spec(\{\}, \mathcal{P}) \ \sqsubseteq_T \ System \tag{1}$$

5 Verification

The current CSP model *System* is very large and complex. Even if one only considers a network with a small number of peers, this model still contains too many states and transitions to be checked automatically by a model checker. Our approach for the verification is to firstly find an abstraction of the original model, called *Abstraction*, whose state-space is smaller. In particular, the abstraction satisfies: $Abstraction \sqsubseteq_T System$. Next, we show that $Spec(\{\}, \mathcal{P}) \sqsubseteq_T Abstraction$, which then implies that Eq.1 is correct.

Due to the space constraint, we describe our approach only briefly here. More details can be found at [10]. First, we arrive at *Abstraction* through the following steps:

1. Weakening the adversary. The original adversary has infinite memory that helps it remember and replay old messages. We weaken it by removing the memory, i.e. allowing the adversary to only *relay* messages. It turns out that the new model using this weakened adversary has the same traces as the original.
2. Reducing the nonce set. The NonceManager process supplies unique and fresh nonces from a potentially *infinite* set. We make use of the *data independence* techniques, developed by [23,24], to derive an abstraction of the original model that uses only 2 nonces (one used by the CA and another by the Verifier). In *System* (in all processes except for NonceManager), nonces are used only for equality check and polymorphic operations (tupling, listing). In other words, these processes are independent of the nonce type.
3. Reducing the counter set. We use the same technique as above to reduce the counter set (which is potentially infinite) to only one value. In the current model, the TPM process uses counter values with the '>' operators, which makes it dependent of the counter type. Therefore, before applying the data independence techniques, we transform the model (without reducing its trace set) to make it independent of the counter type.

After these reduction steps, we arrive at *Abstraction*, the model less complex but refined by *System*. We have implemented a small instance of *Abstraction*

for a system with 3 peers in FDR [25], the model-checker tool for CSP. The check for $Spec(\{\}, \mathcal{P}) \sqsubseteq_T Abstraction$ returns **true** after **13,501,797** states and **73,831,002** transitions. Next, we generalize this result to an arbitrary number of peers by proving that $traces(Abstraction) \subseteq traces(Spec(\{\}, \mathcal{P}))$ for any $\mathcal{P}$. The proof is constructed via induction, as follows:

1. (Base case). Let tr be a trace of *Abstraction* such that $tr \upharpoonright \{|completeChurn|\} = \langle\rangle$ where $\upharpoonright$ is the restriction operator (for example, $sq \upharpoonright X$ removes non-X elements from sq). Then $tr \in traces(Spec(\{\}, \mathcal{P}))$.
2. (Inductive case). For any $\theta \neq \langle\rangle$, let tr be a trace of *Abstraction* such that:

$$tr \upharpoonright \{|completeChurn|\} = \theta \ \wedge \ tr \in traces(Spec(\{\}, \mathcal{P}))$$

Let tr' be another trace of *Abstraction*, then:

$$\forall e \,.\, tr' \upharpoonright \{|completeChurn|\} = \theta \hat{} \langle e \rangle \ \Rightarrow \ tr' \in traces(Spec(\{\}), \mathcal{P})$$

6 Conclusion

In this paper, we have discussed various security problems in structured P2P systems. We focus on the false-destination attacks, in which an adversary can falsely claim to be the destination of a search key. Due to the nature of structured P2P overlays (keys are stored at unique root nodes), there are a number of reasons for such attacks. A P2P system is secure from these attacks if it satisfies the root authenticity (or RA) property, which is implied by the neighbor authenticity (NA) property. We propose a P2P architecture aiming to meet this property, using Trusted Computing as the security mechanism. We then describe our formalization of the proposed architecture in CSP, then our verification that the RA property is indeed met.

We identify a few avenues to be explored in the future work. Our current churn model is relatively strict, since peers are only allowed to leave gracefully. To be more realistic, it must allow for fail-stop and Byzantine failure. The CA may not have to keep information of which nodes currently in the network. It means that before issuing certificate to a peer, the CA would ask for its neighbor certificates and only continue if they match. A further extension would be to remove the CA altogether. In this case, the certificate and verification mechanism might become much more complex and the system might not be able to cope with a complex churn model. In another direction, because the TPM was not designed with P2P applications in mind, its operation set may be too restricted for such systems. Therefore, it would be interesting to find an abstraction of general-purposed trusted hardware that is more powerful, flexible and suitable for P2P. Finally, we plan to carry out performance analysis (via simulation) of our proposed P2P architecture as well as its extensions in order to evaluate the overhead incurred from having the RA property.

References

1. Gnutella Project: Gnutella specification. WorldWide Web (July 2007), http://rfc-gnutella.sourceforge.net/developer/testing/
2. Emule Project: emule homepage. World Wide Web (May 2002), http://www.emule-project.net/
3. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 149–160 (2001)
4. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, pp. 329–350 (2001)
5. Rowstron, A., Druschel, P.: Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. SIGOPS Operating Systems Review 35(5), 188–201 (2001)
6. Bryan, D.A., Lowekamp, B.B., Jennings, C.: Sosimple: A serverless, standards-based, p2p sip communication system. In: International Workshop on Advanced Archtectures and Algorithms for Internet Delivery and Applications, pp. 42–49 (2005)
7. Castro, M., Duschel, P., Kermarrec, A.M., Rowstron, A.: Scribe: a large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communication 20(8) (2002)
8. Dinh, T.T.A., Chothia, T., Ryan, M.: A trusted infrastructure for p2p-based marketplaces. In: 9th IEEE International Conference on P2P Computing, pp. 151–154 (2009)
9. Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats, pp. 1–9 (2008)
10. Dinh, T.T.A., Ryan, M.: Checking security property of P2P systems in CSP. Technical Report CSR-10-07, School of Computer Science, University of Birmingham (2010), http://www.cs.bham.ac.uk/~ttd/files/technicalReport.pdf
11. Sit, E., Morris, R.: Security considerations for peer-to-peer distributed hash tables. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 261–269. Springer, Heidelberg (2002)
12. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Systems Review 36(SI), 299–314 (2002)
13. Wang, P., Hopper, N., Osipkov, I., Kim, Y.: Myrmic: Secure and robust DHT routing. Technical report, University of Minnesota (2006)
14. Ganesh, L., Zhao, B.Y.: Identity theft protection in structured overlays. In: EEE Workshop on Secure Network Protocols, pp. 49–54 (2005)
15. Borgström, J., Nestmann, U., Alima, L.O., Gurov, D.: Verifying a structured peerto- peer overlay network: The static case. In: Global Computing, pp. 250–265 (2004)
16. Bakhshi, R., Gurov, D.: Verification of peer-to-peer algorithms: A case study. Electronic Notes in Theoretical Computer Science 181, 35–47 (2007)
17. Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., Stoica, I.: Towards a common api for structured peer-to-peer overlays. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 33–44. Springer, Heidelberg (2003)

18. Trusted Computing Group: TPM Specification version 1.2. Parts 1–3 (2007), http://www.trustedcomputinggroup.org/specs/TPM/
19. ISO/IEC PAS DIS 11889: Information technology – Security techniques – Trusted platform module
20. Trusted Computing Group: Press release (2008), http://www.trustedcomputinggroup.org/news/press/member_releases/WAVETCGPROMOTI%ONMW5_31_FINAL_.pdf
21. Trusted Computing Group: TCG timeline (2008), http://www.trustedcomputinggroup.org/about/corporate_documents/
22. Balfe, S., Lakhani, A.D., Paterson, K.G.: Trusted computing: Providing security for peer-to-peer networks. In: International Conference on Peer-to-Peer Computing, pp. 117–124. IEEE Computer Society, Los Alamitos (2005)
23. Lazic, R.S.: A Semantic Study of Data-Independence with Applications to the Mechanical Verification of Concurrent Systems. PhD thesis, Oxford University (1997)
24. Broadfoot, P.J.: Data Independence in the Model Checking of Security Protocols. PhD thesis, Oxford University (2001)
25. Formal System Europe Ltd: FDR2 model checker tool. World Wide Web, http://www.fsel.com/software.html

Appendix

CSP

Events.

a, b	event communicated by CSP processes
Σ	universal sets of events; $a, b \in \Sigma$
$c.v$	communication of event v on channel c
$\lvert c \rvert$	set of all events on channel c

Processes.

P, Q	CSP process
$STOP$	do nothing
$a \rightarrow P$	prefix
$c.v \rightarrow P$	communicate value v on channel c, then behave as P
$P \,\Box\, Q,\ P \trianglelefteq b \trianglerighteq Q$	external choice and condition (if b then P; else Q)
$P \underset{X}{\parallel} Q$	P and Q executed concurrently, synchronized on X
$P \,\lvert\lvert\lvert\, Q$	interleaving
$P[\![R]\!],\ P \setminus X$	renaming relation and hiding events in X

Trace Models. This is one of the three denotational semantics supported by CSP. The other two are stable failure and failure-divergence semantics.

$$
\begin{aligned}
traces(STOP) &= \{\langle\rangle\} \\
traces(a \rightarrow P) &= \{\langle\rangle\} \cup \{\langle a\rangle^{\frown} s \mid s \in traces(P)\} \\
traces(P \,\Box\, Q) &= traces(P) \cup traces(Q) \\
traces(P[\![R]\!]) &= \{\, tr \mid \exists s \in traces(P) \bullet s\ R^*\ tr \,\} \\
traces(P \,\lvert\lvert\lvert\, Q) &= \bigcup \{s \,\lvert\lvert\lvert\, t \mid s \in traces(P), t \in traces(Q)\} \\
traces(P \underset{X}{\parallel} Q) &= \bigcup \{s \underset{X}{\parallel} t\} \mid s \in traces(P), t \in traces(Q)
\end{aligned}
$$

where R^*, $\lvert\lvert\lvert$ and $\underset{X}{\parallel}$ are defined as follows:

$$\langle a_1, .., a_n\rangle \; R^* \; \langle b_1, .., b_m\rangle \leftrightarrow n = m \;\wedge\; \forall i \leqslant n \bullet \; a_i \; R \; b_i$$

$$\begin{aligned}
\langle\rangle \;|||\; s &= \{s\} \\
s \;|||\; t &= t \;|||\; s \\
\langle a\rangle^\frown s \;|||\; \langle b\rangle^\frown t &= \{\langle a\rangle^\frown u \parallel u \in s \;|||\; \langle b\rangle^\frown t\} \\
&\quad \cup \{\langle b\rangle^\frown u \parallel u \in t \;|||\; \langle a\rangle^\frown s\}
\end{aligned}$$

$$\begin{aligned}
s \underset{X}{\parallel} t &= t \underset{X}{\parallel} s \\
\langle\rangle \underset{X}{\parallel} \langle\rangle &= \{\langle\rangle\} \\
\langle\rangle \underset{X}{\parallel} \langle x\rangle &= \{\langle\rangle\} \qquad (x \in X) \\
\langle\rangle \underset{X}{\parallel} \langle y\rangle &= \{\langle y\rangle\} \qquad (y \notin X) \\
\langle x\rangle^\frown s \underset{X}{\parallel} \langle y\rangle^\frown t &= \{\langle y\rangle^\frown u \mid u \in \langle x\rangle^\frown s \underset{X}{\parallel} t\} \\
\langle x\rangle^\frown s \underset{X}{\parallel} \langle x\rangle^\frown t &= \{\langle x\rangle^\frown u \mid u \in s \underset{X}{\parallel} t\} \\
\langle x\rangle^\frown s \underset{X}{\parallel} \langle x'\rangle^\frown t &= \{\langle\rangle\} \qquad (x, x' \in X \;\wedge\; x \neq x') \\
\langle y\rangle^\frown s \underset{X}{\parallel} \langle y'\rangle^\frown t &= \{\langle y\rangle^\frown u \mid u \in s \underset{X}{\parallel} \langle y'\rangle^\frown t\} \\
&\quad \cup \{\langle y'\rangle^\frown u \mid u \in t \underset{X}{\parallel} \langle y\rangle^\frown u\}
\end{aligned}$$

System Model in CSP

Events. Let $\mathcal{P}$ and $Nonces = \{Nonce.id \mid id \in NonceIDs\}$ be the set of peers (each having a unique ID) and the set of fresh nonces. Let

$$ChurnEvents = \{Churn.\langle c, i\rangle \mid c \in \{join, leave\}, i \in \mathcal{P}\}$$

be the set of churn events. Let $Counts$ be the set of counter values. We define the set of events representing TPMs' signature on counter values as:

$$SigMessages = \{SqR.\langle n, p, c\rangle, SqI.\langle n, p, c\rangle \mid n \in Nonces, p \in \mathcal{P}, c \in Counts\}$$

For any n, p, c, $SqR.\langle n, p, c\rangle$ and $SqI.\langle n, p, c\rangle$ represent signatures on the results of the `TPM_ReadCounter` and `TPM_IncrementCounter` commands respectively.

Let $NonceMessages = \{SqN.\langle n\rangle \mid n \in Nonces\}$ be a set of events representing a nonce value being passed in the network. Finally, the set of events representing certificates issued by the CA is defined as:

$$CertMessages = \{Cert.\langle p, l, r, c\rangle \mid p, l, r \in \mathcal{P}, c \in Counts\}$$

Then, the set of all events used in the model is defined as:

$$\begin{aligned}
Messages = {} & ChurnMessages \cup SigMessages \\
& \cup NonceMessages \cup CertMessages
\end{aligned}$$

Channels. The following channels are used:

$send, receive$: $Agents.Agents.Messages$
where $Agents = \mathcal{P} \cup \{NM, CA, VF\}$
$take, fake$: $Agents.Agents.Messages$
$learn, say$: $Messages$
$output$: $\mathcal{P}.\mathcal{P}$
$completeChurn$: $ChurnMessages$
$unlock$: $Agents.\mathcal{P}$

NonceManager Process

$$NManager(X) = \Box_{\substack{n \in X \\ j \in Agents}} send.NM.j.SqN.\langle n \rangle \rightarrow NManager(X \backslash \{n\})$$

$$NonceManager = NManager(Nonces)$$

TPM Process

$$TPM(i,c) = \Box_{\substack{n \in Nonces \\ j \in Agents}} receive.j.i.SqN.\langle n \rangle \rightarrow \left(\begin{array}{l} \Box_{\substack{d \geqslant c \\ d \in Counts}} send.i.j.SqR.\langle n, i, d \rangle \rightarrow unlock.VF.i \rightarrow TPM(i,d) \\ \Box \Box_{\substack{d > c \\ d \in Counts}} send.i.j.SqI.\langle n, i, d \rangle \rightarrow unlock.CA.i \rightarrow TPM(i,d) \end{array} \right)$$

$$TPMs = |||_{i \in \mathcal{P}} TPM(i,0)$$

CA Process

$$\begin{array}{l} CAProcess(ps,pn) = \quad |ps| == 0 \ \& \ Join0(ps,pn) \\ \qquad\qquad\qquad\qquad \Box \ |ps| == 1 \ \& \ Join1(ps,pn) \\ \qquad\qquad\qquad\qquad \Box \ |ps| > 1 \ \& \ JoinAndLeaveN(ps,pn) \end{array}$$

$$\begin{array}{l} JoinAndLeaveN(ps,pn) = \Box_{i \in pn} receive.i.CA.Churn.\langle join, i \rangle \rightarrow JoinN(i,ps,pn) \\ \qquad\qquad\qquad\qquad \Box \Box_{i \in ps} receive.i.CA.Churn.\langle leave, i \rangle \rightarrow \\ \qquad\qquad\qquad\qquad\qquad\qquad \textit{if } |ps| > 2 \textit{ then } LeaveN(i,ps,pn) \\ \qquad\qquad\qquad\qquad\qquad\qquad \textit{else } Leave2(i,ps,pn) \end{array}$$

$$
\begin{array}{l}
\mathit{JoinN(i,ps,pn)} = \\
\quad \Box_{n1,n2,n3 \in Nonces} \; \mathit{receive.NM.CA.SqN}.\langle n1\rangle \rightarrow \mathit{send.CA.i.SqN}.\langle n1\rangle \\
\qquad \rightarrow \left(\begin{array}{ll}
\Box_{c1,c2,c3 \in Counts} & \mathit{receive.i.CA.SqI}.\langle n1, i, c1\rangle \rightarrow \\
& \mathit{let}\ S = ps \cup \{i\} \\
& \quad (l, r) = \mathit{neighbor}(i, S) \\
& \quad (l1, r1) = \mathit{neighbor}(l, S) \\
& \quad (l2, r2) = \mathit{neighbor}(r, S)\ \mathit{within} \\
& \qquad \mathit{send.CA.i.Cert}.\langle i, l, r, c1\rangle \\
& \quad \rightarrow \mathit{receive.NM.CA.SqN}.\langle n2\rangle \\
& \quad \rightarrow \mathit{send.CA.l.SqN}.\langle n2\rangle \\
& \quad \rightarrow \mathit{receive.l.CA.SqI}.\langle n2, l, c2\rangle \\
& \quad \rightarrow \mathit{send.CA.l.Cert}.\langle l, l1, i, c2\rangle \\
& \quad \rightarrow \mathit{receive.NM.CA.SqN}.\langle n3\rangle \\
& \quad \rightarrow \mathit{send.CA.r.SqN}.\langle n3\rangle \\
& \quad \rightarrow \mathit{receive.r.CA.SqI}.\langle n3, r, c3\rangle \\
& \quad \rightarrow \mathit{send.CA.r.Cert}.\langle r, i, r2, c3\rangle \\
& \quad \rightarrow \mathit{completeChurn.Churn}.\langle \mathrm{join}, i\rangle \\
& \quad \rightarrow \mathit{unlock.CA.i} \rightarrow \mathit{unlock.CA.l} \\
& \quad \rightarrow \mathit{unlock.CA.r} \rightarrow \mathit{CAProcess(S,pn\backslash\{i\})}
\end{array} \right)
\end{array}
$$

Other sub-processes, namely *Join0(ps,pn), Join1(ps,pn), Leave2(ps,pn)* and *LeaveN(ps,pn)* are be defined similarly. The function $neighbor(p, ps)$ returns the left and right neighbor of p in ps. More precisely,

$$
\begin{array}{l}
neighbor(p, ps) = (left(p, ps), right(p, ps)) \\
left(p, ps) = l \quad \text{if} \;\; l \in ps \;\wedge\; \forall p' \in ps \backslash \{l\} \,.\, |p \ominus l| \leqslant |p' \ominus l| \\
right(p, ps) = r \quad \text{if} \;\; r \in ps \;\wedge\; \forall p' \in ps \backslash \{r\} \,.\, |r \ominus p| \leqslant |r \ominus p'|
\end{array}
$$

Verifier Process.

$$
\begin{array}{l}
\mathit{VerifierProcess} = \\
\quad \Box_{n \in Nonces} \; \mathit{receive.NM.VF.SqN}.\langle n\rangle \\
\qquad \rightarrow \left(\begin{array}{ll}
\Box_{i,l,r \in \mathcal{P}} & \mathit{send.VF.i.SqN}.\langle n\rangle \\
& \rightarrow \left(\begin{array}{ll}
\Box_{c \in Counts} & \mathit{receive.i.VF.SqR}.\langle n, i, c\rangle \\
& \rightarrow \mathit{receive.i.VF.Cert}.\langle i, l, r, c\rangle \\
& \rightarrow \mathit{if}\ l = r\ \mathit{and}\ l = i\ \mathit{then} \\
& \qquad \mathit{output.i.i} \rightarrow \mathit{unlock.VF.i} \\
& \qquad \rightarrow \mathit{STOP} \\
& \quad \mathit{else}\ \mathit{VerifierProcessN(l,i)}
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{VerifierProcessN(l,i)} = \\
\quad \Box_{n \in Nonces} \; \mathit{receive.NM.VF.SqN}.\langle n\rangle \rightarrow \mathit{send.VF.l.SqN}.\langle n\rangle \\
\qquad \rightarrow \left(\begin{array}{ll}
\Box_{cl \in Counts} & \mathit{receive.l.VF.SqR}.\langle n, l, cl\rangle \\
& \rightarrow \left(\begin{array}{ll}
\Box_{ll \in \mathcal{P}} & \mathit{receive.l.VF.Cert}.\langle l, ll, i, cl\rangle \\
& \quad \rightarrow \mathit{output.l.i} \rightarrow \mathit{unlock.VF.i} \\
& \quad \rightarrow \mathit{unlock.VF.l} \rightarrow \mathit{STOP}
\end{array} \right)
\end{array} \right)
\end{array}
$$

Adversary Process

$$MemoryNonce(n) = learn.SqN.\langle n\rangle \rightarrow ReplayNonce(n)$$
$$ReplayNonce(n) = say.SqN.\langle n\rangle \rightarrow ReplayNonce(n)$$

$$MemorySigR(n,i,c) = learn.SqR.\langle n,i,c\rangle \rightarrow ReplaySigR(n,i,c)$$
$$MemorySigI(n,i,c) = learn.SqI.\langle n,i,c\rangle \rightarrow ReplaySigI(n,i,c)$$
$$ReplaySigR(n,i,c) = say.SqR.\langle n,i,c\rangle \rightarrow ReplaySigR(n,i,c)$$
$$ReplaySigI(n,i,c) = say.SqI.\langle n,i,c\rangle \rightarrow ReplaySigI(n,i,c)$$

$$MemoryCert(i,l,r,c) = learn.Cert.\langle i,l,r,c\rangle \rightarrow ReplayCert(i,l,r,c)$$
$$ReplayCert(i,l,r,c) = say.Cert.\langle i,l,r,c\rangle \rightarrow ReplayCert(i,l,r,c)$$

$$Memory = \Vert\!\Vert_{n\in Nonces} MemoryNonce(n) \Vert\!\Vert \Vert\!\Vert_{n\in Nonces, i\in\mathcal{P}, c\in Counts} \begin{pmatrix} MemorySigR(n,i,c) \\ \Vert\!\Vert\ MemorySigI(n,i,c) \end{pmatrix} \Vert\!\Vert \Vert\!\Vert_{i,l,r\in\mathcal{P}, c\in Counts} MemoryCert(i,l,r,c)$$

$$ChurnInitiator = \Box_{i\in\mathcal{P}} \begin{pmatrix} say.Churn.\langle \text{join}, i\rangle \rightarrow ChurnInitiator \\ \Box\ say.Churn.\langle \text{leave}, i\rangle \rightarrow ChurnInitiator \end{pmatrix}$$
$$Adversary = Memory \Vert\!\Vert ChurnInitiator$$

Putting It Together

$$Network = \left(Adversary \underset{\chi_i}{\parallel} TPMs \right) \setminus \chi_i$$
$$CAandVFProcess = CAProcess(\{\}, \mathcal{P}) \Vert\!\Vert VerifierProcess$$
$$OtherAgents = \left(NonceManager \underset{\{|fake.NM|\}}{\parallel} CAandVFProcess \right)$$
$$Impl = \left(OtherAgents \underset{\chi_e}{\parallel} Network\ Big \right) \setminus \{|take, fake, unlock|\}$$

Table 1. Renaming relations and synchronization sets

Name	Details	Applied to
RAd_1	$learn \leftarrow take.i.j \mid i,j \in Agents, \{i,j\} \cup \mathcal{P} \neq \varnothing$	$Adversary$
RAd_2	$say \leftarrow fake.i.j \mid i,j \in Agents, \{i,j\} \cup \mathcal{P} \neq \varnothing$	$Adversary$
$RCom_1$	$send \leftarrow take$	$TPMs$, $CAProcess$ and $VerifierProcess$
$RCom_2$	$receive \leftarrow fake$	$TPMs$, $CAProcess$ and $VerifierProcess$
$RNonce_1$	$send.NM.i \leftarrow take.NM.i \mid i \in \mathcal{P}$	$NonceManager$
$RNonce_2$	$send.NM.j \leftarrow fake.NM.j \mid j \notin \mathcal{P}$	$NonceManager$
χ_i	$\{\|take.i.a, fake.a.i \mid i \in \mathcal{P}, a \in Agents\|\}$	
χ_e	$\{\|take.a.i, fake.i.a \mid i \in \mathcal{P}, a \in Agents\|\}$	

Modeling and Analyzing Security in the Presence of Compromising Adversaries

David Basin and Cas Cremers*

Department of Computer Science, ETH Zurich

Abstract. We present a framework for modeling adversaries in security protocol analysis, ranging from a Dolev-Yao style adversary to more powerful adversaries who can reveal different parts of principals' states during protocol execution. Our adversary models unify and generalize many existing security notions from both the computational and symbolic settings. We extend an existing symbolic protocol-verification tool with our adversary models, resulting in the first tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries capable of state-reveal queries. In case studies, we automatically find new attacks and rediscover known attacks that previously required detailed manual analysis.

1 Introduction

Problem context. Many cryptographic protocols are designed to work in the face of various forms of corruption. For example, a Diffie-Hellman key agreement protocol, where signatures are used to authenticate the exchanged half-keys, provides perfect forward secrecy [17,30]: the resulting key remains secret even when the signature keys are later compromised by the adversary. Designing protocols that work even in the presence of different forms of adversary compromise has considerable practical relevance. It reflects the multifaceted computing reality with different rings of protection (user-space, kernel space, hardware security modules) offering different levels of assurance with respect to the computation of cryptographic functions (e. g., the quality of the pseudo-random numbers generated) and the storage of keys and intermediate results.

Symbolic and computational approaches have addressed this problem to different degrees. Most symbolic formalisms are based on the Dolev-Yao model. These offer, with few exceptions, a limited view of honesty and conversely corruption: either principals are honest from the start and always keep their secrets to themselves or they are completely malicious and always under adversary control. Under this limited view, it is impossible to distinguish between the security provided by early key-exchange protocols such as the Bilateral key-exchange [12]

* This work was supported by ETH Research Grant ETH-30 09-3 and the FP7-ICT-2007-1 Project no. 216471 (AVANTSSAR).

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 340–356, 2010.

and state-of-the art protocols such as (H)MQV [27,23]. It is also impossible to discern any benefit from storing the long-term keys in a tamper-proof module or performing part of a computation in a cryptographic coprocessor.

In contrast to the above, researchers in the computational setting, such as [9,25,32,7,21], have explored stronger adversary models, whereby principals may be selectively corrupted during protocol execution. For example, their short-term or long-term secrets, or the results of intermediate computations may be revealed (at different times) to the adversary. These models are used to establish stronger properties, such as perfect forward secrecy or resilience against state-reveal attacks. There are, however, drawbacks to these computational models. These models have been defined just for key-agreement protocols, whereas one may expect similar definitions to exist for any security protocol. Moreover, contrary to the security models used in symbolic approaches, there is no automated tool support available for the stronger adversary models.

Contributions. We present a framework for analyzing security protocols in the presence of adversaries with a wide range of compromise capabilities. We show how analogs of adversary models studied in the computational setting can be modeled in our framework. For example, we can model attacks against implementations of cryptographic protocols involving the mixed use of cryptographic co-processors for the secure storage of long-term secrets with the computation of intermediate results in less-secure main memory for efficiency reasons.

Our models bridge another gap between the computational and symbolic approaches by providing symbolic definitions for adversaries and security properties that were previously only available in the computational setting. Moreover, by decomposing security properties into an adversary model and a basic security property, we unify and generalize many existing security properties.

Our framework directly lends itself to protocol analysis. As an example, we extend Scyther [14], a symbolic protocol analysis tool. This results in the first automated tool that systematically supports notions such as weak perfect forward secrecy, key compromise impersonation, and adversaries that can reveal the local state of agents. We analyze a set of protocols with the tool and rediscover many attacks previously reported in the cryptographic literature. Furthermore, our tool finds previously unreported attacks, including a novel attack on HMQV. This shows that symbolic methods can be effectively extended for analyses that previously were possible only using a manual computational analysis.

Organization. We present our framework in Section 2 and show several applications in Section 3. We discuss related work in Section 4 and conclude in Section 5.

2 Compromising Adversary Model

We define an operational semantics that is modular with respect to the adversary's capabilities. Our framework is compatible with the majority of existing semantics for security protocols, including trace and strand-space semantics. We

have kept our execution model minimal to focus on the adversary rules. However, it would be straightforward to incorporate a more elaborate execution model, e. g., with control-flow commands.

Notational preliminaries. Let f be a function. We write $dom(f)$ and $ran(f)$ to denote f's domain and range. We write $f[b \hookleftarrow a]$ to denote f's update, i. e., the function f' where $f'(x) = b$ when $x = a$ and $f'(x) = f(x)$ otherwise. We write $f : X \nrightarrow Y$ to denote a partial function from X to Y. For any set S, $\mathcal{P}(S)$ denotes the power set of S and S^* denotes the set of finite sequences of elements from S. We write $\langle s_0, \ldots, s_n \rangle$ to denote the sequence of elements s_0 to s_n, and we omit brackets when no confusion can result. For s a sequence of length $|s|$ and $i < |s|$, s_i denotes the i-th element. We write $s \hat{} s'$ for the concatenation of the sequences s and s'. Abusing set notation, we write $e \in s$ iff $\exists i. s_i = e$. We write $union(s)$ for $\bigcup_{e \in s} e$. We define $last(\langle\rangle) = \emptyset$ and $last(s \hat{} \langle e \rangle) = e$.

We write $[t_0, \ldots, t_n \,/\, x_0, \ldots, x_n] \in \mathcal{S}ub$ to denote the substitution of t_i for x_i, for $0 \leq i \leq n$. We extend the functions *dom* and *ran* to substitutions. We write $\sigma \cup \sigma'$ to denote the union of two substitutions, which is defined when $dom(\sigma) \cap dom(\sigma') = \emptyset$, and write $\sigma(t)$ for the application of the substitution σ to t. Finally, for R a binary relation, R^* denotes its reflexive transitive closure.

2.1 Terms and Events

We assume given the infinite sets *Agent*, *Role*, *Fresh*, *Var*, *Func*, and *TID* of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, function names, and thread identifiers. We assume that *TID* contains two distinguished thread identifiers, *Test* and $tid_{\mathcal{A}}$. These identifiers single out a distinguished "point of view" thread of an arbitrary agent and an adversary thread, respectively.

To bind local terms, such as freshly generated terms or local variables, to a protocol role instance (thread), we write $t \sharp tid$. This denotes that the term t is local to the protocol role instance identified by tid.

Definition 1. *Terms*

$$\begin{aligned} Term ::= & \; Agent \mid Role \mid Fresh \mid Var \mid Fresh \sharp TID \mid Var \sharp TID \\ & \mid (Term, Term) \mid pk(Term) \mid sk(Term) \mid k(Term, Term) \\ & \mid \{\!|\, Term \,|\!\}^{a}_{Term} \mid \{\!|\, Term \,|\!\}^{s}_{Term} \mid Func(Term^*) \end{aligned}$$

For each $X, Y \in Agent$, $sk(X)$ denotes the long-term private key, $pk(X)$ denotes the long-term public key, and $k(X, Y)$ denotes the long-term symmetric key shared between X and Y. Moreover, $\{\!|\, t_1 \,|\!\}^{a}_{t_2}$ denotes the asymmetric encryption (for public keys) or the digital signature (for signing keys) of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^{s}_{t_2}$ denotes symmetric encryption. The set *Func* is used to model other cryptographic functions, such as hash functions. Freshly generated terms and variables are assumed to be local to a thread (an instance of a role).

Depending on the protocol analyzed, we assume that symmetric or asymmetric long-term keys have been distributed prior to protocol execution. We assume

the existence of an inverse function on terms, where t^{-1} denotes the inverse key of t. We have that $pk(X)^{-1} = sk(X)$ and $sk(X)^{-1} = pk(X)$ for all $X \in \mathit{Agent}$, and $t^{-1} = t$ for all other terms t.

We define a binary relation $\vdash$, where $M \vdash t$ denotes that the term t can be inferred from the set of terms M. Let $t_0, \ldots, t_n \in \mathit{Term}$ and let $f \in \mathit{Func}$. We define $\vdash$ as the smallest relation satisfying:

$$t \in M \Rightarrow M \vdash t \qquad M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash (t_1, t_2)$$
$$M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2} \wedge M \vdash t_2 \Rightarrow M \vdash t_1 \qquad M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2}$$
$$M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2} \wedge M \vdash (t_2)^{-1} \Rightarrow M \vdash t_1 \qquad M \vdash t_1 \wedge M \vdash t_2 \Rightarrow M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2}$$
$$\bigwedge_{0 \le i \le n} M \vdash t_i \Rightarrow M \vdash f(t_0, \ldots, t_n)$$

The term t' is a subterm of t, written $t' \sqsubseteq t$, when t' is a syntactic subterm of t, e. g., $t_1 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2}$ and $t_2 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2}$. We write $FV(t)$ for the free variables of t, where $FV(t) = \{t' \mid t' \sqsubseteq t\} \cap \big(\mathit{Var} \cup \{v \sharp \mathit{tid} \mid v \in \mathit{Var} \wedge \mathit{tid} \in \mathit{TID}\}\big)$.

Definition 2. *Events*

$$\begin{aligned}
\mathit{AgentEvent} ::= {} & \mathsf{create}(\mathit{Role}, \mathit{Agent}) \mid \mathsf{send}(\mathit{Term}) \mid \mathsf{recv}(\mathit{Term}) \\
& \mid \mathsf{generate}(\mathcal{P}(\mathit{Fresh})) \mid \mathsf{state}(\mathcal{P}(\mathit{Term})) \mid \mathsf{sessionkeys}(\mathcal{P}(\mathit{Term})) \\
\mathit{AdversaryEvent} ::= {} & \mathsf{LKR}(\mathit{Agent}) \mid \mathsf{SKR}(\mathit{TID}) \mid \mathsf{SR}(\mathit{TID}) \mid \mathsf{RNR}(\mathit{TID}) \\
\mathit{Event} ::= {} & \mathit{AgentEvent} \mid \mathit{AdversaryEvent}
\end{aligned}$$

We explain the interpretation of the agent and adversary events shortly. Here we simply note that the first three agent events are standard: starting a thread, sending a message, and receiving a message. The message in the send and receive events does not include explicit sender or recipient fields although, if desired, they can be given as subterms of the message. The last three agent events tag state information, which can possibly be compromised by the adversary. The four adversary events specify which information the adversary compromises. These events can occur any time during protocol execution and correspond to different kinds of *adversary queries* from computational models. All adversary events are executed in the single adversary thread $\mathit{tid}_{\mathcal{A}}$.

2.2 Protocols and Threads

A protocol is a partial function from role names to event sequences, i. e., $\mathit{Protocol} : \mathit{Role} \nrightarrow \mathit{AgentEvent}^*$. We require that no thread identifiers occur as subterms of events in a protocol definition.

Example 1 (Simple protocol). Let $\{\mathrm{Init}, \mathrm{Resp}\} \subseteq \mathit{Role}$, $\mathit{key} \in \mathit{Fresh}$, and $x \in \mathit{Var}$. We define the simple protocol SP as follows.

$$\begin{aligned}\mathrm{SP}(\mathrm{Init}) = \langle &\mathsf{generate}(\{key\}), \mathsf{state}(\{key, \{\!|\,\mathrm{Resp}, key\,|\!\}^a_{sk(\mathrm{Init})}\}),\\ &\mathsf{send}(\mathrm{Init}, \mathrm{Resp}, \{\!|\,\{\!|\,\mathrm{Resp}, key\,|\!\}^a_{sk(\mathrm{Init})}\,|\!\}^a_{pk(\mathrm{Resp})}), \mathsf{sessionkeys}(\{key\})\rangle\\ \mathrm{SP}(\mathrm{Resp}) = \langle &\mathsf{recv}(\mathrm{Init}, \mathrm{Resp}, \{\!|\,\{\!|\,\mathrm{Resp}, x\,|\!\}^a_{sk(\mathrm{Init})}\,|\!\}^a_{pk(\mathrm{Resp})}),\\ &\mathsf{state}(\{x, \{\!|\,\mathrm{Resp}, x\,|\!\}^a_{sk(\mathrm{Init})}\}), \mathsf{sessionkeys}(\{x\})\rangle\end{aligned}$$

Here, the initiator generates a key and sends it (together with the responder name) signed and encrypted, along with the initiator and responder names. The recipient expects to receive a message of this form. The additional events mark session keys and state information. The state information is implementation-dependent and marks which parts of the state are stored at a protection level lower than the long-term private keys. The state information in SP corresponds to, e. g., implementations that use a hardware security module for encryption and signing and perform all other computations in ordinary memory.

Protocols are executed by agents who execute roles, thereby instantiating role names with agent names. Agents may execute each role multiple times. Each instance of a role is called a *thread*. We distinguish between the fresh terms and variables of each thread by assigning them unique names, using the function $\mathit{localize} : \mathit{TID} \to \mathcal{S}ub$, defined as $\mathit{localize}(\mathit{tid}) = \bigcup_{cv \in \mathit{Fresh} \cup \mathit{Var}} [cv\sharp\mathit{tid} \,/\, cv]$. Using $\mathit{localize}$, we define a function $\mathit{thread} : (\mathit{AgentEvent}^* \times \mathit{TID} \times \mathcal{S}ub) \to \mathit{AgentEvent}^*$ that yields the sequence of agent events that may occur in a thread.

Definition 3 (Thread). *Let l be a sequence of events, $\mathit{tid} \in \mathit{TID}$, and let σ be a substitution. Then $\mathit{thread}(l, \mathit{tid}, \sigma) = \sigma(\mathit{localize}(\mathit{tid})(l))$.*

Example 2. Let $\{A, B\} \subseteq \mathit{Agent}$. For a thread $t_1 \in \mathit{TID}$ performing the Init role from Example 1, we have $\mathit{localize}(t_1)(\mathit{key}) = \mathit{key}\sharp t_1$ and

$$\begin{aligned}&\mathit{thread}(\mathrm{SP}(\mathrm{Init}), t_1, [A, B \,/\, \mathrm{Init}, \mathrm{Resp}]) =\\ &\quad\langle \mathsf{generate}(\{\mathit{key}\sharp t_1\}), \mathsf{state}(\{\mathit{key}\sharp t_1, \{\!|\,B, \mathit{key}\sharp t_1\,|\!\}^a_{sk(A)}\}),\\ &\quad\ \mathsf{send}(A, B, \{\!|\,\{\!|\,B, \mathit{key}\sharp t_1\,|\!\}^a_{sk(A)}\,|\!\}^a_{pk(B)}), \mathsf{sessionkeys}(\{\mathit{key}\sharp t_1\})\rangle\,.\end{aligned}$$

Test thread. When verifying security properties, we will focus on a particular thread. In the computational setting, this is the thread where the adversary performs a so-called *test query*. In the same spirit, we call the thread under consideration the *test thread*, with the corresponding thread identifier Test. For the test thread, the substitution of role names by agent names, and all free variables by terms, is given by σ_{Test} and the role is given by R_{Test}. For example, if the test thread is performed by Alice in the role of the initiator, trying to talk to Bob, we have that $R_{\mathit{Test}} = \mathrm{Init}$ and $\sigma_{\mathit{Test}} = [\mathrm{Alice}, \mathrm{Bob} \,/\, \mathrm{Init}, \mathrm{Resp}]$.

2.3 Execution Model

We define the set Trace as $(\mathit{TID} \times \mathit{Event})^*$, representing possible execution histories. The state of our system is a four-tuple $(\mathit{tr}, \mathit{IK}, \mathit{th}, \sigma_{\mathit{Test}}) \in \mathit{Trace} \times \mathcal{P}(\mathit{Term}) \times (\mathit{TID} \nrightarrow \mathit{Event}^*) \times \mathcal{S}ub$, whose components are (1) a trace tr, (2)

$$\frac{R \in dom(P) \quad dom(\sigma) = Role \quad ran(\sigma) \subseteq Agent \quad tid \notin (dom(th) \cup \{tid_A, Test\})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{create}(R, \sigma(R)))\rangle, IK, th[thread(P(R), tid, \sigma) \leftarrow tid], \sigma_{Test})}[\mathsf{create}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \quad Test \notin dom(th)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(Test, \mathsf{create}(R_{Test}, a))\rangle, IK, th[thread(P(R_{Test}), Test, \sigma_{Test}) \leftarrow Test], \sigma_{Test})}[\mathsf{createTest}]$$

$$\frac{th(tid) = \langle \mathsf{send}(m)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{send}(m))\rangle, IK \cup \{m\}, th[l \leftarrow tid], \sigma_{Test})}[\mathsf{send}]$$

$$\frac{th(tid) = \langle \mathsf{recv}(pt)\rangle^\frown l \quad IK \vdash \sigma(pt) \quad dom(\sigma) = FV(pt)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{recv}(\sigma(pt)))\rangle, IK, th[\sigma(l) \leftarrow tid], \sigma_{Test})}[\mathsf{recv}]$$

$$\frac{th(tid) = \langle \mathsf{generate}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{generate}(M))\rangle, IK, th[l \leftarrow tid], \sigma_{Test})}[\mathsf{generate}]$$

$$\frac{th(tid) = \langle \mathsf{state}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{state}(M))\rangle, IK, th[l \leftarrow tid], \sigma_{Test})}[\mathsf{state}]$$

$$\frac{th(tid) = \langle \mathsf{sessionkeys}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{sessionkeys}(M))\rangle, IK, th[l \leftarrow tid], \sigma_{Test})}[\mathsf{sessionkeys}]$$

Fig. 1. Execution-model rules

the adversary's knowledge IK, (3) a partial function th mapping the thread identifiers of initiated threads to sequences of events, and (4) the role to agent and variable assignments of the test thread. We include the trace as part of the state to facilitate defining the partner function later.

Definition 4 ($\mathcal{T}est\mathcal{S}ub_P$)**.** *Given a protocol P, we define the set of* test substitutions *$\mathcal{T}est\mathcal{S}ub_P$ as the set of ground substitutions σ_{Test} such that $dom(\sigma_{Test}) = dom(P) \cup \{v \sharp Test \mid v \in Var\}$ and $\forall r \in dom(P).\ \sigma_{Test}(r) \in Agent$.*

For P a protocol, the set of initial system states $IS(P)$ is defined as

$$IS(P) = \bigcup_{\sigma_{Test} \in \mathcal{T}est\mathcal{S}ub_P} \{(\langle\rangle, Agent \cup \{pk(a) \mid a \in Agent\}, \emptyset, \sigma_{Test})\}.$$

In contrast to Dolev-Yao models, the initial adversary knowledge does not include any long-term secret keys. The adversary may learn these from long-term key reveal (LKR) events.

The semantics of a protocol $P \in Protocol$ is defined by a transition system that combines the execution-model rules from Figure 1 with a set of adversary rules from Figure 2. We first present the execution-model rules.

The create rule starts a new instance of a protocol role R (a *thread*). A fresh thread identifier tid is assigned to the thread, thereby distinguishing it from existing threads, the adversary thread, and the test thread. The rule takes the protocol P as a parameter. The role names of P, which can occur in events associated with the role, are replaced by agent names by the substitution σ. Similarly, the createTest rule starts the test thread. However, instead of choosing an arbitrary role, it takes an additional parameter R_{Test}, which represents the test role and will be instantiated in the definition of the transition relation in Def. 7. Additionally, instead of choosing an arbitrary σ, the test substitution σ_{Test} is used.

The send rule sends a message m to the network. In contrast, the receive rule accepts messages from the network that match the pattern pt, where pt is a term that may contain free variables. The resulting substitution σ is applied to the remaining protocol steps l.

The last three rules support our adversary rules, given shortly. The generate rule marks the fresh terms that have been generated, the state rule marks the current local state, and the sessionkeys rule marks a set of terms as session keys.

Auxiliary functions. We define the long-term secret keys of an agent a as

$$LongTermKeys(a) = \{sk(a)\} \cup \bigcup_{b \in Agent} \{k(a,b), k(b,a)\}\,.$$

For traces, we define an operator $\downarrow$ that projects traces on events belonging to a particular thread identifier. For all tid, tid', and tr, we define $\langle\rangle \downarrow tid = \langle\rangle$ and

$$(\langle(tid', e)\rangle^\frown tr) \downarrow tid = \begin{cases} \langle e\rangle^\frown (tr \downarrow tid) & \text{if } tid = tid', \text{ and} \\ tr \downarrow tid & \text{otherwise.} \end{cases}$$

Similarly, for event sequences, the operator $\lfloor$ selects the contents of events of a particular type. For all $\mathsf{evtype} \in \{\mathsf{create}, \mathsf{send}, \mathsf{recv}, \mathsf{generate}, \mathsf{state}, \mathsf{sessionkeys}\}$, we define $\langle\rangle \lfloor \mathsf{evtype} = \langle\rangle$ and

$$(\langle e\rangle^\frown l) \lfloor \mathsf{evtype} = \begin{cases} \langle m\rangle^\frown (l \lfloor \mathsf{evtype}) & \text{if } e = \mathsf{evtype}(m), \text{ and} \\ l \lfloor \mathsf{evtype} & \text{otherwise.} \end{cases}$$

During protocol execution, the test thread may intentionally share some of its short-term secrets with other threads, such as a session key. Hence some adversary rules require distinguishing between the intended *partner threads* and other threads. There exist many notions of partnering in the literature. In general, we use partnering based on matching histories for protocols with two roles, as defined below.

Definition 5 (Matching histories). *For sequences of events l and l', we define* $\mathrm{MH}(l, l') \equiv \big(l \lfloor \mathsf{recv} = l' \lfloor \mathsf{send}\big) \wedge \big(l \lfloor \mathsf{send} = l' \lfloor \mathsf{recv}\big)$.

Our partnering definition is parameterized over the protocol P and the test role R_{Test}. These parameters are later instantiated in the transition-system definition.

Definition 6 (Partnering). *Let R be the non-test role, i. e., $R \in dom(P)$ and $R \neq R_{Test}$. For tr a trace, $Partner(tr, \sigma_{Test}) = \big\{tid \;\big|\; tid \neq Test \wedge \big(\exists a.\mathsf{create}(R, a) \in tr \downarrow tid\big) \wedge \exists l\,.\,\mathrm{MH}(\sigma_{Test}(P(R_{Test})), (tr \downarrow tid)^\frown l)\big\}$.*

A thread tid is a partner iff (1) tid is not $Test$, (2) tid performs the role different from $Test$'s role, and (3) tid's history matches the $Test$ thread (for $l = \langle\rangle$) or the thread may be completed to a matching one (for $l \neq \langle\rangle$).

2.4 Adversary-Compromise Rules

We define the adversary-compromise rules in Figure 2. They factor the security definitions from the cryptographic protocol literature along three dimensions

$$\frac{a \notin \{\sigma_{Test}(R) \mid R \in dom(P)\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{others}}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad a \notin \{\sigma_{Test}(R) \mid R \in dom(P) \setminus \{R_{Test}\}\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{actor}}]$$

$$\frac{th(Test) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{after}}]$$

$$\frac{th(Test) = \langle\rangle \qquad tid \in Partner(tr, \sigma_{Test}) \qquad th(tid) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LKR}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{aftercorrect}}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{SKR}(tid))\rangle, IK \cup union((tr \downarrow tid) \lfloor \mathsf{sessionkeys}), th, \sigma_{Test})}[\mathsf{SKR}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test}) \qquad th(tid) \neq \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{SR}(tid))\rangle, IK \cup last((tr \downarrow tid) \lfloor \mathsf{state}), th, \sigma_{Test})}[\mathsf{SR}]$$

$$\frac{}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{RNR}(tid))\rangle, IK \cup union((tr \downarrow tid) \lfloor \mathsf{generate}), th, \sigma_{Test})}[\mathsf{RNR}]$$

Fig. 2. Adversary-compromise rules

of adversarial compromise: *which* kind of data is compromised, *whose* data it is, and *when* the compromise occurs. Not all combinations of capabilities have been used for analyzing protocols. Some combinations are not covered because of impossibility results (e. g. [23]), whereas other combinations appear to have been previously overlooked.

Compromise of long-term keys. The first four rules model the compromise of an agent a's long-term keys, represented by the long-term key reveal event $\mathsf{LKR}(a)$. In traditional Dolev-Yao models, this event occurs implicitly for dishonest agents before the honest agents start their threads.

The $\mathsf{LKR}_{\mathsf{others}}$ rule formalizes the adversary capability used in the symbolic analysis of security protocols since Lowe's attack on Needham-Schroeder [28]: the adversary can learn the long-term keys of any agent a that is not an intended partner of the test thread. Hence, if the test thread is performed by Alice, communicating with Bob, the adversary can learn, e. g., Charlie's long-term key.

The $\mathsf{LKR}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term key of the agent executing the test thread (also called the *actor*). The intuition is that a protocol may still function as long as the long-term keys of the other partners are not revealed. This rule allows the adversary to perform so-called Key Compromise Impersonation attacks [21]. The rule's second premise is required because our model allows agents to communicate with themselves.

The $\mathsf{LKR}_{\mathsf{after}}$ and $\mathsf{LKR}_{\mathsf{aftercorrect}}$ rules restrict when the compromise may occur. In particular, they allow the compromise of long-term keys only after the test thread has finished, captured by the premise $th(Test) = \langle\rangle$. This is the sole premise of $\mathsf{LKR}_{\mathsf{after}}$. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{after}}$, it is said to satisfy perfect forward secrecy (PFS) [17,30]. $\mathsf{LKR}_{\mathsf{aftercorrect}}$ has the additional premise that a finished partner

thread must exist for the test thread. This condition stems from [23] and excludes the adversary from both inserting fake messages during protocol execution and learning the key of the involved agents later. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LKR}_{\mathsf{aftercorrect}}$, it is said to satisfy weak perfect forward secrecy (wPFS). This property is motivated by a class of protocols given in [23] whose members fail to satisfy PFS, although some satisfy this weaker property.

The left-hand side of Figure 3 depicts the relationships between our long-term key compromise rules in the relevant dimensions: the rows specify *when* the compromise occurs and the columns specify *whose* long-term keys are compromised. With respect to *when* a compromise occurs, we differentiate between before, during, and after the test thread. With respect to *whose* keys are compromised, we differentiate between agents not involved in the communication (others), the agent performing the test thread (actor), and the other partner (peer). The ovals specify the effects of each of the long-term key reveal rules.

Compromise of short-term data. The three remaining adversary rules correspond to the compromise of short-term data, that is, data local to a specific thread. In the right-hand side of Figure 3, we show the relevant dimensions: *whose* data, specified by the columns, and *which* kind of data, specified by the rows. Whereas we assumed a long-term key compromise reveals *all* long-term keys of an agent, we differentiate here between the different kinds of local data. Because we assume that local data does not exist before or after a session, we can ignore the temporal dimension.

We differentiate between three kinds of local data: *randomness*, *session keys*, and *other local data* such as the results of intermediate computations. The notion that the adversary may learn the randomness used in a protocol stems from [25]. Considering adversaries that can reveal session keys, e. g., by cryptanalysis, is found in many works, such as [4]. An adversary capable of revealing an agent's local state was described in [9].

In our adversary-compromise models, the session-key reveal event $\mathsf{SKR}(tid)$ and state reveal event $\mathsf{SR}(tid)$ indicate that the adversary gains access to the session key or, respectively, the local state of the thread *tid*. These are marked respectively by the sessionkeys and state events.

The contents of the state change over time and are erased when the thread ends. This is reflected in the SR rule by the *last* state marker for the state contents and the third premise requiring that the thread *tid* has not ended. The random number reveal event $\mathsf{RNR}(tid)$ indicates that the adversary learns the random numbers generated in the thread *tid*.

The rules SKR and SR allow for the compromise of session keys and the contents of a thread's local state. Their premise is that the compromised thread is not a partner thread. In contrast, the premise of the RNR rule allows for the compromise of all threads, including the partner threads. This rule stems from [25], where it is shown that it is possible to construct protocols that are correct in the presence of an adversary capable of RNR.

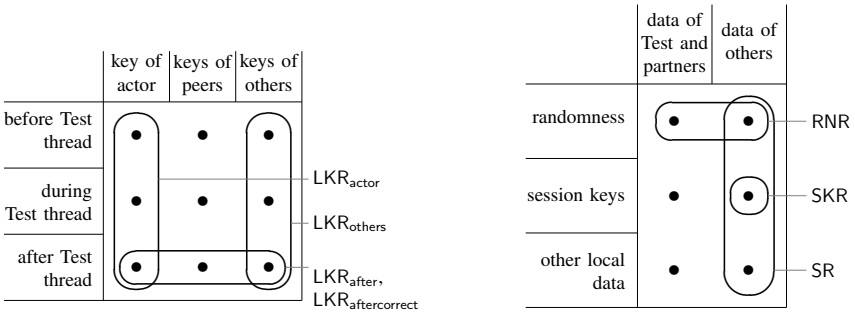


Fig. 3. Relating long-term and short-term data reveal rules

For protocols that establish a session key, we assume the session key is shared by all partners and should be secret: revealing it trivially violates the protocols' security. Hence the rules disallow the compromise of session keys of the test or partner threads. Similarly, our basic rule set does not contain a rule for the compromise of other local data of the partners. Including such a rule is straightforward. However it is unclear whether any protocol would be correct with respect to such an adversary.

We call each subset of the set of adversary rules from Figure 2 an *adversary-compromise model*.

2.5 Transition Relation and Security Properties

Given a protocol and an adversary-compromise model, we define the possible protocol behaviors as a set of reachable states.

Definition 7 (Transition relation and reachable states). *Let P be a protocol, Adv an adversary-compromise model, and R_{Test} a role. We define a* transition relation $\rightarrow_{P,Adv,R_{Test}}$ *from the execution-model rules from Figure 1 and the rules in Adv. The variables P, Adv, and R_{Test} in the adversary rules are instantiated by the corresponding parameters of the transition relation. For states s and s', $s \rightarrow_{P,Adv,R_{Test}} s'$ iff there exists a rule in either Adv or the execution-model rules with the premises $Q_1(s), \ldots, Q_n(s)$ and the conclusion $s \rightarrow s'$ such that all of the premises hold. We define the set of* reachable states RS *as*

$$\mathrm{RS}(P, Adv, R_{Test}) = \left\{ s \mid \exists s_0.\, s_0 \in IS(P) \wedge s_0 \rightarrow^*_{P,Adv,R_{Test}} s \right\}.$$

We provide a symbolic definition of session-key secrecy which, when combined with different adversary models, gives rise to different notions of secrecy found in the literature. Other security properties, such as secrecy of general terms, symbolic indistinguishability, or different variants of authentication, can be defined analogously.

Table 1. Decomposing security properties

Security property	Decomposition	
	Basic property	Adversary model
Perfect Forward Secrecy [17,30]	Secrecy	$\{\mathsf{LKR}_{\mathsf{after}}\}$
Weak Perfect Forward Secrecy [23]	Secrecy	$\{\mathsf{LKR}_{\mathsf{aftercorrect}}\}$
Known-Key Security [30]	Secrecy (of session key)	$\{\mathsf{SKR}\}$
Key Compromise Impersonation [21]	Authentication	$\{\mathsf{LKR}_{\mathsf{actor}}\}$

Definition 8 (Session-key secrecy). *Let P be a protocol and Adv an adversary model. We say that P satisfies* session-key secrecy *with respect to Adv if and only if*

$$\forall R_{Test} \in dom(P).\, \forall (tr, IK, th, \sigma_{Test}) \in \mathrm{RS}(P, Adv, R_{Test}).$$
$$th(Test) = \langle\rangle \Rightarrow \forall k \in union((tr \downarrow Test) \lfloor \mathsf{sessionkeys}).\, IK \nvdash k\,.$$

Many definitions of security properties, such as perfect forward secrecy, also contain elements of adversary capabilities. In our framework, such properties are cleanly separated into a basic security property (e. g. secrecy or authentication) and an adversary model. In Table 1, we decompose different security properties from the literature this way.

Our way of modeling security properties provides a uniform view and also allows for direct generalizations of security properties. This leads to new, practically relevant combinations of adversary models and basic security properties. For example, for a hardware security module restricted to protecting long-term keys, relevant properties could be secrecy or agreement, resilient against state-reveal. Further properties arise by considering the combination of our adversary models with other basic properties like non-repudiation, plausible deniability, anonymity, or resistance to denial-of-service attacks.

3 Applications and Case Studies

Modeling adversary notions from the literature. We use our modular semantics to provide a uniform formalization of different adversary models, including a number of established adversary models from the computational setting [9,3,5,23,25]. We focus on the adversary capabilities only, abstracting from subtle differences between the computational models. For example, the model in [9] has an execution model that restricts the agents' choice of thread identifiers, leading to a different notion of partner threads than in other models. Here we define partnering uniformly by matching histories. We refer the reader to [10,29,8,11] for further details on the differences between computational models.

Table 2 provides an overview of different adversary models, interpreted as instances of our semantics. We write Adv_{CK} to denote the adversary model extracted from the CK model [9] and similarly for other models. We use a check

Table 2. Mapping adversary-compromise models from the literature

	Long-term data				Short-term data			
	Owner		Timing		Type			
Name	others	actor	after	aftercorrect	SessionKey	State	Random	Origin of model
Adv_{EXT}								external Dolev-Yao
Adv_{INT}	✓							Dolev-Yao [28]
Adv_{CA}		✓						Key Compromise Impersonation [21]
Adv_{AFC}				✓				Weak Perfect Forward Secrecy [23]
Adv_{AF}			✓	✓				Perfect Forward Secrecy [17,30]
Adv_{BR}	✓				✓			BR93 [4], BR95 [5]
Adv_{CKw}	✓	✓		✓	✓	✓		CK2001-wPFS [23]
Adv_{CK}	✓		✓	✓	✓	✓		CK2001 [9]
$Adv_{\text{eCK-1}}$	✓				✓		✓	eCK [25]
$Adv_{\text{eCK-2}}$	✓	✓		✓	✓			

(✓) to denote that the rule labeling the column is included in the adversary model named in the row.

Tool support. We extended the symbolic security-protocol verification tool Scyther [14,15] with our adversary rules from Figure 2. We used this tool to automatically analyze a set of protocols, described below. The tool, all protocol models, and test scripts can be downloaded from [13].

Attack example. The MQV protocol family [24,33,27] is a class of authenticated key-exchange protocols designed to provide strong security guarantees. The HMQV protocol was proven secure with respect to the adversary model in [23]. This model is the analog of our Adv_{CKw} model, where the local state of HMQV is defined as the random values generated for the Diffie-Hellman key-exchange. Surprisingly, our tool finds that the HMQV protocol is, depending on the definition of the state, insecure in adversary models that contain SR rules, such as the CK model [9].

Below we describe a new attack, which shows that MQV and HMQV are insecure in, e. g., Adv_{CKw}, if the final exponentiation in the computation of the session key is performed in the local state. It is possible for an adversary to reuse the inputs to this exponentiation to impersonate an agent in future sessions. The attack is not covered in [23] because both the proof and the extended analysis given there assume that the local state contains only the ephemeral keys (the temporary private keys).

Using notation from [23], we show the attack in Figure 4, where $d_1 = \bar{H}(X, \text{Bob})$, $e_1 = \bar{H}(Z, \text{Alice})$, and $e_2 = \bar{H}(Y, \text{Alice})$. The attack starts with Bob receiving a message g^z apparently coming from Alice. This message may have been sent by an agent or have been generated by the adversary. Next, Bob generates x and sends $X = g^x$, which is intercepted by the adversary. Thread 1 is not a partner of the test thread because its history does not match the test thread's. Hence the adversary can compromise thread 1's state, accessing $x + d_1 b$. At any desired time, the adversary sends X to the responder test thread of Alice. Alice computes and sends $Y = g^y$ and computes the session key based on X and y. The adversary intercepts Y and computes $H((YA^{e_1})^{x+d_1 b})$. This yields the session key of the test thread.

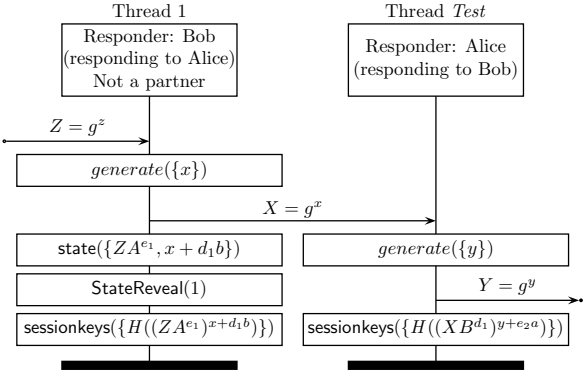


Fig. 4. SR attack on HMQV

We assume that in critical scenarios the protocol is implemented entirely in a tamper-proof module or cryptographic coprocessor and the local state is therefore empty, which prevents this attack. Conversely, if (H)MQV will be implemented entirely in unprotected memory, the state will also include the long-term keys, which enables an attack where the adversary compromises these keys using SR. This example shows how analysis with respect to our models can help sharpen protocol implementation requirements.

Further case studies. In Table 3, we summarize the attacks found using our tool on protocols with respect to the adversary models from Table 2. A cross (×) in the table denotes that an attack was found. Attacks marked (A) were previously unreported. Attacks marked (B) were previously found by manual computational analysis. The set of protocols includes both formally analyzed protocols (NS, NSL, BKE, Yahalom) as well as protocols recently proposed in computational settings (HMQV, DH-ISO, Naxos, KEA+). Our tool rediscovers the attacks described in the literature, e. g., that DH-ISO is insecure in the eCK model [25] and that the implicitly authenticated two-message protocols KEA+, Naxos, and HMQV do not satisfy perfect forward secrecy. Additionally our tool finds new attacks on KEA+ and HMQV. The time needed for finding the attacks in the table ranged from less than a second to three minutes for each attack.

4 Related Work

Related work in computational analysis. Most research on adversary compromise has been performed in the context of key-exchange protocols in the computational setting, e. g. Canetti and Krawczyk [9,23], Shoup [32], Bellare et al. [5,3,4], Katz and Yung [22], LaMacchia et al. [25], and Bresson and Manulis [7]. In general, any two computational models are incomparable due to (often minor) differences not only in the adversary notions, but also in the definitions of partnership, the execution models, and security property specifics. As these models are generally presented in a monolithic way, where all parts are intertwined, it is

Table 3. Attacks found: (A) new, and (B) rediscovered automatically

	Adv_{EXT}	Adv_{INT}	Adv_{CA}	Adv_{AFC}	Adv_{AF}	Adv_{BR}	Adv_{CKw}	Adv_{CK}	$Adv_{\mathrm{eCK\text{-}1}}$	$Adv_{\mathrm{eCK\text{-}2}}$
DH-ISO [18,25]									×(B)	
DH-ISO-C [18]								×	×	
DHKE-1 [18]							×(A)		×	×
HMQV-C [24]							×	×		
HMQV [24]					×(B)		×(A)	×		
NAXOS [25,16]					×(B)		×	×		
KEA+ [26]				×(A)	×(B)		×	×	×	×
NSL [28]			×(A)	×(B)	×(B)		×	×	×	×
BKE [12]			×(A)	×(B)	×(B)		×	×	×	×
Yahalom-Paulson [31]			×(A)	×(B)	×(B)	×(A)	×	×	×	×
NS [28]		×	×(A)	×(B)	×(B)	×	×	×	×	×

difficult to separate these notions. Details of some of these definitions and their relationships have been studied by, e. g., Choo et al. [11,10], Bresson et al. [8], LaMacchia et al. [25], and Menezes and Ustaoglu [29].

The CryptoVerif tool of Blanchet [6] is a mechanized tool for computational analysis. Its adversary model covers Adv_{INT}, corresponding to static corruption, i. e., the classical Dolev-Yao adversary.

Related work in symbolic analysis. In the symbolic setting, Guttman [20] has modeled a form of forward secrecy. With respect to verification, the only work we are aware of is where researchers have verified (or discovered attacks on) key-compromise related properties of particular protocols. These cases do not use a compromising adversary model, but are ad-hoc constructions of key compromise, made for specific protocols, which can be verified in a Dolev-Yao style adversary model.

In [1], Abadi, Blanchet, and Fournet analyzed the JFK protocol in the Pi Calculus and showed it achieves perfect forward secrecy, by giving the adversary all long-term keys at the end of the protocol run. This corresponds to manually instrumenting the analog of our $\mathsf{LKR}_{\mathsf{after}}$ rule. Paulson used his inductive approach to reason about the compromise of short-term data [31]. To model compromise, he adds a rule to the protocol, called *Oops*, that directly gives short-term data to the adversary. This rule is roughly analogous to our SKR rule. Gupta and Shmatikov [19,18] link a symbolic adversary model that includes dynamic corruptions to an adversary model used in the computational analysis of key-agreement protocols. They describe in [19] a cryptographically-sound logic that can be used to prove security in the presence of adaptive corruptions, that is, the adversary can dynamically obtain the long-term keys of agents.

In [2], we have built upon the work presented here and introduce the concept of a *protocol-security hierarchy*, which classifies the relative strength of protocols against different forms of compromise.

5 Conclusions

We have provided the first symbolic framework capable of systematically modeling a family of adversaries endowed with different compromise capabilities. Our

adversary capabilities generalize those from the computational setting and combine them with a symbolic model. In doing so, we unify and generalize a wide range of adversary models from both settings.

Our definitions of adversaries and security properties from the computational setting allow us to apply symbolic techniques to problems that were previously tackled only by computational approaches. We developed the first tool capable of systematically handling notions such as weak perfect forward secrecy, key compromise impersonation, and session state compromise. In case studies, our tool not only rediscovered many attacks previously reported in the cryptographic literature, e. g., on DH-ISO, it also found new attacks, e. g., on HMQV and KEA+. These examples show that our symbolic adversary models are surprisingly effective for automatically establishing results that, until now, required labor-intensive manual computational analysis.

References

1. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi calculus. ACM Transactions on Information and System Security (TISSEC) 10(3), 1–59 (2007)
2. Basin, D., Cremers, C.: Degrees of security: Protocol guarantees in the face of compromising adversaries. In: 19th EACSL Annual Conferences on Computer Science Logic (CSL 2010). Advanced Research in Computing and Software Science, Springer, Heidelberg (2010) (to appear)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. 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)
5. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: STOC 1995, pp. 57–66. ACM Press, New York (1995)
6. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: IEEE Symposium on Security and Privacy, pp. 140–154 (May 2006)
7. Bresson, E., Manulis, M.: Securing group key exchange against strong corruptions. In: ASIACCS, pp. 249–260. ACM, New York (2008)
8. Bresson, E., Manulis, M., Schwenk, J.: On security models and compilers for group key exchange protocols. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 292–307. Springer, Heidelberg (2007)
9. 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)
10. Choo, K.-K., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment proofs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 624–643. Springer, Heidelberg (2005)
11. Choo, K.-K., Boyd, C., Hitchcock, Y., Maitland, G.: On session identifiers in provably secure protocols. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 351–366. Springer, Heidelberg (2005)
12. Clark, J., Jacob, J.: A survey of authentication protocol literature (1997), http://citeseer.ist.psu.edu/clark97survey.html

13. Cremers, C.: Scyther tool with compromising adversaries extension. Includes protocol description files and test scripts, http://people.inf.ethz.ch/cremersc/scyther/
14. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
15. Cremers, C.: Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In: CCS '08: Proc. of the 15th ACM Conference on Computer and Communications Security, pp. 119–128. ACM, New York (2008)
16. Cremers, C.: Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In: Abdalla, M., et al. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 20–33. Springer, Heidelberg (2009)
17. Günther, C.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990)
18. Gupta, P., Shmatikov, V.: Towards computationally sound symbolic analysis of key exchange protocols. In: Proc. FMSE 2005, pp. 23–32. ACM, New York (2005)
19. Gupta, P., Shmatikov, V.: Key confirmation and adaptive corruptions in the protocol security logic. In: FCS-ARSPA 2006, pp. 113–142 (2006)
20. Guttman, J.D.: Key compromise, strand spaces, and the authentication tests. In: 17th Annual Conference on Mathematical Foundations of Programming Semantics. Invited lecture, ENTCS, vol. 45, pp. 1–21 (2001)
21. Just, M., Vaudenay, S.: Authenticated multi-party key agreement. In: Kim, K.-c., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 36–49. Springer, Heidelberg (1996)
22. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
23. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176 (2005), http://eprint.iacr.org/ (retrieved on April 14, 2009)
24. 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)
25. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
26. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., et al. (eds.) PKC 2006. LNCS, vol. 3958, pp. 378–394. Springer, Heidelberg (2006)
27. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Designs, Codes and Cryptography 28, 119–134 (2003)
28. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
29. Menezes, A., Ustaoglu, B.: Comparing the pre- and post-specified peer models for key agreement. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 53–68. Springer, Heidelberg (2008)

30. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (October 1996)
31. Paulson, L.: Relations between secrets: Two formal analyses of the Yahalom protocol. Journal of Computer Security 9(3), 197–216 (2001)
32. Shoup, V.: On formal models for secure key exchange (version 4), (November 1999); revision of IBM Research Report RZ 3120 (April 1999)
33. Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Cryptography 46(3), 329–342 (2008)

On Bounding Problems of Quantitative Information Flow*

Hirotoshi Yasuoka and Tachio Terauchi

Tohoku University
yasuoka@kb.ecei.tohoku.ac.jp,
terauchi@ecei.tohoku.ac.jp

Abstract. Researchers have proposed formal definitions of quantitative information flow based on information theoretic notions such as the Shannon entropy, the min entropy, the guessing entropy, and channel capacity. This paper investigates the hardness of precisely checking the quantitative information flow of a program according to such definitions. More precisely, we study the "bounding problem" of quantitative information flow, defined as follows: Given a program M and a positive real number q, decide if the quantitative information flow of M is less than or equal to q. We prove that the bounding problem is not a k-safety property for any k (even when q is fixed, for the Shannon-entropy-based definition with the uniform distribution), and therefore is not amenable to the self-composition technique that has been successfully applied to checking non-interference. We also prove complexity theoretic hardness results for the case when the program is restricted to loop-free boolean programs. Specifically, we show that the problem is PP-hard for all the definitions, showing a gap with non-interference which is coNP-complete for the same class of programs. The paper also compares the results with the recently proved results on the comparison problems of quantitative information flow.

1 Introduction

We consider programs containing high security inputs and low security outputs. Informally, the quantitative information flow problem concerns the amount of information that an attacker can learn about the high security input by executing the program and observing the low security output. The problem is motivated by applications in information security. We refer to the classic by Denning [11] for an overview.

In essence, quantitative information flow measures *how* secure, or insecure, a program (or a part of a program –e.g., a variable–) is. Thus, unlike non-interference [9,12], that only tells whether a program is completely secure or not completely secure, a definition of quantitative information flow must be able to

* This work was supported by MEXT KAKENHI 20700019, 20240001, and 22300005, and Global COE Program "CERIES".

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 357–372, 2010.

distinguish two programs that are both interferent but have different degrees of "secureness."

For example, consider the programs $M_1 \equiv \texttt{if}\ H = g\ \texttt{then}\ O := 0\ \texttt{else}\ O := 1$ and $M_2 \equiv O := H$. In both programs, H is a high security input and O is a low security output. Viewing H as a password, M_1 is a prototypical login program that checks if the guess g matches the password.[1] By executing M_1, an attacker only learns whether H is equal to g, whereas she would be able to learn the entire content of H by executing M_2. Hence, a reasonable definition of quantitative information flow should assign a higher quantity to M_2 than to M_1, whereas non-interference would merely say that M_1 and M_2 are both interferent, assuming that there are more than one possible value of H.

Researchers have attempted to formalize the definition of quantitative information flow by appealing to information theory. This has resulted in definitions based on the Shannon entropy [11,6,15], the min entropy [24], the guessing entropy [14,1], and channel capacity [18,16,22]. All of these definitions map a program (or a part of a program) onto a non-negative real number, that is, they define a function $\mathcal{X}$ such that given a program M, $\mathcal{X}(M)$ is a non-negative real number. (Concretely, $\mathcal{X}$ is $SE[\mu]$ for the Shannon-entropy-based definition with the distribution μ, $ME[\mu]$ for the min-entropy-based definition with the distribution μ, $GE[\mu]$ for the guessing-entropy-based definition with the distribution μ, and CC for the channel-capacity-based definition.) Therefore, a natural verification problem for quantitative information flow is to decide, given M and a quantity $q \geq 0$, if $\mathcal{X}(M) \leq q$. The problem is well-studied for the case $q = 0$ as it is actually equivalent to checking non-interference (cf. Section 2.1). The problem is open for $q > 0$. We call this the *bounding problem* of quantitative information flow.

The problem has a practical relevance as a user is often interested in knowing if her program leaks information within some allowed bound. That is, the bounding problem is a form of quantitative information flow *checking* problem (as opposed to *inference*). Much of the previous research has focused on information theoretic properties of quantitative information flow and approximate (i.e., incomplete and/or unsound) algorithms for checking and inferring quantitative information flow. To fill the void, in a recent work [29], we have studied the hardness and possibilities of deciding the *comparison problem* of quantitative information flow, which is the problem of precisely checking if the information flow of one program is larger than that of the other, that is, the problem of deciding if $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ given programs M_1 and M_2. The study has lead to some remarkable results, summarized in Section 3 and Section 4 of this paper to contrast with the new results on the bounding problem. However, the hardness results on the comparison problem do not imply hardness of the bounding problem.[2] Thus, this paper settles the open question.

[1] Here, for simplicity, we assume that g is a program constant. See Section 2 for modeling attacker/user (i.e., low security) inputs.

[2] But, they imply the hardness of the inference problem because we can compare $\mathcal{X}(M_1)$ and $\mathcal{X}(M_2)$ once we have computed them.

We summarize the main results of the paper below. Here, $\mathcal{X}$ is $SE[U]$, $ME[U]$, $GE[U]$ or CC, where U is the uniform distribution.

- Checking if $\mathcal{X}(M) \leq q$ is not a k-safety property [25,8] for any k.
- Restricted to loop-free boolean programs, checking if $\mathcal{X}(M) \leq q$ is PP-hard.

Roughly, a verification problem being k-safety means that it can be reduced to a standard safety problem, such as the unreachability problem, via self composition [3,10]. For instance, non-interference is a 2-safety property (technically, for the termination-insensitive case[3]), and this has enabled its precise checking via a reduction to a safety problem via self composition and applying automated safety verification techniques [25,21,27]. Also, our recent work [29] has shown that deciding the comparison problem of quantitative information flow for all distributions for the entropy-based definitions (i.e., checking if $\forall\mu.SE[\mu](M_1) \leq SE[\mu](M_2)$, $\forall\mu.ME[\mu](M_1) \leq ME[\mu](M_2)$, and $\forall\mu.GE[\mu](M_1) \leq GE[\mu](M_2)$) are 2-safety problems (and in fact, all equivalent).

We also prove a complexity theoretic gap with these related problems. We have shown in the previous paper [29] that, for loop-free boolean programs, both checking non-interference and the above comparison problem for entropy-based definitions with universally quantified distributions are coNP-complete. (PP is believed to be strictly harder than coNP. In particular, coNP = PP implies the collapse of the polynomial hierarchy to level 1.)

Therefore, the results suggest that the bounding problems of quantitative information flow are harder than the related problems of checking non-interference and the quantitative information flow comparison problems with universally quantified distributions, and may require different techniques to solve (i.e., not self composition).

The rest of the paper is organized as follows. Section 2 reviews the existing information-theoretic definitions of quantitative information flow and formally defines the bounding problems. Section 3 proves that the bounding problems are not k-safety problems. Section 4 proves that the bounding problems are PP-hard (even) when restricted to loop-free boolean programs. Section 5 discusses some implications of the hardness results. Section 6 discusses related work, and Section 7 concludes. All the proofs appear in the extended report [28].

2 Preliminaries

We introduce the information theoretic definitions of quantitative information flow that have been proposed in literature. First, we review the notion of the *Shannon entropy* [23], $\mathcal{H}[\mu](X)$, which is the average of the information content, and intuitively, denotes the uncertainty of the random variable X.

Definition 1 (Shannon Entropy). *Let X be a random variable with sample space $\mathbb{X}$ and μ be a probability distribution associated with X (we write μ explicitly for clarity). The Shannon entropy of X is defined as*

[3] We restrict to terminating programs in this paper. (The termination assumption is nonrestrictive because we assume safety verification as a blackbox routine.).

$$\mathcal{H}[\mu](X) = \sum_{x \in \mathbb{X}} \mu(X = x) \log \frac{1}{\mu(X = x)}$$

(The logarithm is in base 2.)

Next, we define *conditional entropy*. Informally, the conditional entropy of X given Y denotes the uncertainty of X after knowing Y.

Definition 2 (Conditional Entropy). *Let X and Y be random variables with sample spaces $\mathbb{X}$ and $\mathbb{Y}$, respectively, and μ be a probability distribution associated with X and Y. Then, the conditional entropy of X given Y, written $\mathcal{H}[\mu](X|Y)$ is defined as*

$$\mathcal{H}[\mu](X|Y) = \sum_{y \in \mathbb{Y}} \mu(Y = y) \mathcal{H}[\mu](X|Y = y)$$

where

$$\mathcal{H}[\mu](X|Y = y) = \textstyle\sum_{x \in \mathbb{X}} \mu(X = x|Y = y) \log \frac{1}{\mu(X = x|Y = y)}$$
$$\mu(X = x|Y = y) = \frac{\mu(X = x, Y = y)}{\mu(Y = y)}$$

Next, we define (conditional) mutual information. Intuitively, the conditional mutual information of X and Y given Z represents the mutual dependence of X and Y after knowing Z.

Definition 3 (Mutual Information). *Let X, Y and Z be random variables and μ be an associated probability distribution.*[4] *Then, the conditional mutual information of X and Y given Z is defined as*

$$\begin{aligned} \mathcal{I}[\mu](X; Y|Z) &= \mathcal{H}[\mu](X|Z) - \mathcal{H}[\mu](X|Y, Z) \\ &= \mathcal{H}[\mu](Y|Z) - \mathcal{H}[\mu](Y|X, Z) \end{aligned}$$

Let M be a program that takes a high security input H and a low security input L, and gives the low security output O. For simplicity, we restrict to programs with just one variable of each kind, but it is trivial to extend the formalism to multiple variables (e.g., by letting the variables range over tuples). Also, for the purpose of the paper, unobservable (i.e., high security) outputs are irrelevant, and so we assume that the only program output is the low security output. Let μ be a probability distribution over the values of H and L. Then, the semantics of M can be defined by the following probability equation. (We restrict to terminating deterministic programs in this paper.)

$$\mu(O = o) = \sum_{\substack{h, \ell \in \mathbb{H}, \mathbb{L} \\ M(h, \ell) = o}} \mu(H = h, L = \ell)$$

Note that we write $M(h, \ell)$ to denote the low security output of the program M given inputs h and ℓ. Now, we are ready to introduce the Shannon-entropy based definition of quantitative information flow (QIF) [11,6,15].

[4] We abbreviate the sample spaces of random variables when they are clear from the context.

Definition 4 (Shannon-Entropy-based QIF). *Let M be a program with a high security input H, a low security input L, and a low security output O. Let μ be a distribution over H and L. Then, the Shannon-entropy-based quantitative information flow is defined*

$$\begin{aligned} SE[\mu](M) &= \mathcal{I}[\mu](O; H|L) \\ &= \mathcal{H}[\mu](H|L) - \mathcal{H}[\mu](H|O, L) \end{aligned}$$

Intuitively, $\mathcal{H}[\mu](H|L)$ denotes the initial uncertainty knowing the low security input and $\mathcal{H}[\mu](H|O, L)$ denotes the remaining uncertainty after knowing the low security output.

As an example, consider the programs M_1 and M_2 from Section 1. For concreteness, assume that g is the value 01 and H ranges over the space $\{00, 01, 10, 11\}$. Let U be the uniform distribution over $\{00, 01, 10, 11\}$, that is, $U(h) = 1/4$ for all $h \in \{00, 01, 10, 11\}$. Computing their Shannon-entropy based quantitative information flow, we have,

$$\begin{aligned} SE[U](M_1) &= \mathcal{H}[U](H) - \mathcal{H}[U](H|O) = \log 4 - \tfrac{3}{4} \log 3 \approx .81128 \\ SE[U](M_2) &= \mathcal{H}[U](H) - \mathcal{H}[U](H|O) = \log 4 - \log 1 = 2 \end{aligned}$$

Hence, if the user was to ask if $SE[U](M_1) \leq 1.0$, that is, "does M_1 leak more than one bit of information (according to $SE[U]$)?", then the answer would be no. But, for the same query, the answer would be yes for M_2.

Next, we introduce the *min entropy*, which Smith [24] recently suggested as an alternative measure for quantitative information flow.

Definition 5 (Min Entropy). *Let X and Y be random variables, and μ be an associated probability distribution. Then, the min entropy of X is defined*

$$\mathcal{H}_\infty[\mu](X) = \log \frac{1}{\mathcal{V}[\mu](X)}$$

and the conditional min entropy of X given Y is defined

$$\mathcal{H}_\infty[\mu](X|Y) = \log \frac{1}{\mathcal{V}[\mu](X|Y)}$$

where

$$\begin{aligned} \mathcal{V}[\mu](X) &= \max_{x \in \mathbb{X}} \mu(X = x) \\ \mathcal{V}[\mu](X|Y = y) &= \max_{x \in \mathbb{X}} \mu(X = x|Y = y) \\ \mathcal{V}[\mu](X|Y) &= \textstyle\sum_{y \in \mathbb{Y}} \mu(Y = y) \mathcal{V}[\mu](X|Y = y) \end{aligned}$$

Intuitively, $\mathcal{V}[\mu](X)$ represents the highest probability that an attacker guesses X in a single try. We now define the min-entropy-based definition of quantitative information flow.

Definition 6 (Min-Entropy-based QIF). *Let M be a program with a high security input H, a low security input L, and a low security output O. Let μ be a distribution over H and L. Then, the min-entropy-based quantitative information flow is defined*

$$ME[\mu](M) = \mathcal{H}_\infty[\mu](H|L) - \mathcal{H}_\infty[\mu](H|O, L)$$

Whereas Smith [24] focused on programs lacking low security inputs, we extend the definition to programs with low security inputs in the definition above. It is easy to see that our definition coincides with Smith's for programs without low security inputs. Also, the extension is arguably natural in the sense that we simply take the conditional entropy with respect to the distribution over the low security inputs.

Computing the min-entropy based quantitative information flow for our running example programs M_1 and M_2 from Section 1 with the uniform distribution, we obtain,

$$ME[U](M_1) = \mathcal{H}_\infty[U](H) - \mathcal{H}_\infty[U](H|O) = \log 4 - \log 2 = 1$$
$$ME[U](M_2) = \mathcal{H}_\infty[U](H) - \mathcal{H}_\infty[U](H|O) = \log 4 - \log 1 = 2$$

Hence, if a user is to check whether $ME[U]$ is bounded by q for $1 \leq q < 2$, then the answer would be yes for M_1, but no for M_2.

The third definition of quantitative information flow treated in this paper is the one based on the guessing entropy [17], that is also recently proposed in literature [14,1].

Definition 7 (Guessing Entropy). *Let X and Y be random variables, and μ be an associated probability distribution. Then, the guessing entropy of X is defined*

$$\mathcal{G}[\mu](X) = \sum_{1 \leq i \leq m} i \times \mu(X = x_i)$$

where $m = |\mathbb{X}|$ and $x_1, x_2, \ldots, x_m$ satisfies $\forall i, j. i \leq j \Rightarrow \mu(X = x_i) \geq \mu(X = x_j)$.

The conditional guessing entropy of X given Y is defined

$$\mathcal{G}[\mu](X|Y) = \sum_{y \in \mathbb{Y}} \mu(Y = y)\mathcal{G}[\mu](X|Y = y)$$

where

$$\mathcal{G}[\mu](X|Y = y) = \textstyle\sum_{1 \leq i \leq m} i \times \mu(X = x_i|Y = y)$$
$$m = |\mathbb{X}| \ \text{ and } \ \forall i, j. i \leq j \Rightarrow \mu(X = x_i|Y = y) \geq \mu(X = x_j|Y = y)$$

Intuitively, $\mathcal{G}[\mu](X)$ represents the average number of times required for the attacker to guess the value of X. We now define the guessing-entropy-based quantitative information flow.

Definition 8 (Guessing-Entropy-based QIF). *Let M be a program with a high security input H, a low security input L, and a low security output O. Let μ be a distribution over H and L. Then, the guessing-entropy-based quantitative information flow is defined*

$$GE[\mu](M) = \mathcal{G}[\mu](H|L) - \mathcal{G}[\mu](H|O, L)$$

Like with the min-entropy-based definition, the previous research on guessing-entropy-based quantitative information flow only considered programs without low security inputs [14,1]. But, it is easy to see that our definition with low security inputs coincides with the previous definitions for programs without low security inputs. Also, as with the extension for the min-entropy-based definition, it simply takes the conditional entropy over the low security inputs.

We test GE on the running example from Section 1 by calculating the quantities for the programs M_1 and M_2 with the uniform distribution.

$$GE[U](M_1) = \mathcal{G}[U](H) - \mathcal{G}[U](H|O) = \tfrac{5}{2} - \tfrac{7}{4} = 0.75$$
$$GE[U](M_2) = \mathcal{G}[U](H) - \mathcal{G}[U](H|O) = \tfrac{5}{2} - 1 = 1.5$$

Hence, if a user is to check whether $GE[U]$ is bounded by q for $0.75 \leq q < 1.5$, then the answer would be yes for M_1, but no for M_2.

The fourth and the final existing definition of quantitative information flow that we introduce in this paper is the one based on *channel capacity* [18,16,22], which is simply defined to be the maximum of the Shannon-entropy based quantitative information flow over the distribution.

Definition 9 (Channel-Capacity-based QIF). *Let M be a program with a high security input H, a low security input L, and a low security output O. Then, the channel-capacity-based quantitative information flow is defined*

$$CC(M) = \max_{\mu} \mathcal{I}[\mu](O; H|L)$$

Unlike the Shannon-entropy based, the min-entropy based, and the guessing-entropy based definitions, the channel-capacity based definition of quantitative information flow is not parameterized by the distribution over the inputs. As with the other definitions, let us test the definition on the running example from Section 1 by calculating the quantities for the programs M_1 and M_2:

$$CC(M_1) = \max_{\mu} \mathcal{I}[\mu](O; H) = 1$$
$$CC(M_2) = \max_{\mu} \mathcal{I}[\mu](O; H) = 2$$

Note that $CC(M_1)$ (resp. $CC(M_2)$) is equal to $ME[U](M_1)$ (resp. $ME[U](M_2)$). This is not a coincidence. In fact, it is known that $CC(M) = ME[U](M)$ for all programs M without low security inputs [24].

2.1 Non-interference

We recall the notion of non-interference [9,12].

Definition 10 (Non-intereference). *A program M is said to be non-interferent iff for any $h, h' \in \mathbb{H}$ and $\ell \in \mathbb{L}$, $M(h, \ell) = M(h', \ell)$.*

It can be shown that for the definitions of quantitative information flow $\mathcal{X}$ introduced above, $\mathcal{X}(M) \leq 0$ iff M is non-interferent.[5] That is, the bounding

[5] Technically, we need the non-zero-ness condition on the distribution for the entropy-based definitions. (See below.).

problem (which we only officially define for positive bounds –see Section 2.2–) degenerates to checking non-interference when 0 is given as the bound.

Theorem 1. *Let μ be a distribution such that $\forall h \in \mathbb{H}, \ell \in \mathbb{L}.\mu(h,\ell) > 0$. Then,*

- *M is non-interferent if and only if $SE[\mu](M) \leq 0$.*
- *M is non-interferent if and only if $ME[\mu](M) \leq 0$.*
- *M is non-interferent if and only if $GE[\mu](M) \leq 0$.*
- *M is non-interferent if and only if $CC(M) \leq 0$.*

The equivalence result on the Shannon-entropy-based definition is proven by Clark et al. [5]. The proofs for the other three definitions are given in the extended report [28].

2.2 Bounding Problem

We define the *bounding problem* of quantitative information flow for each definition of the quantitative information flow introduced above. The bounding problem for the Shannon-entropy based definition $B_{SE}[\mu]$ is defined as follows: Given a program M and a positive real number q, decide if $SE[\mu](M) \leq q$. Similarly, we define the bounding problems for the other three definitions $B_{ME}[\mu]$, $B_{GE}[\mu]$, and B_{CC} as follows.

$$\begin{aligned} B_{ME}[\mu] &= \{(M,q) \mid ME[\mu](M) \leq q\} \\ B_{GE}[\mu] &= \{(M,q) \mid GE[\mu](M) \leq q\} \\ B_{CC} &= \{(M,q) \mid CC(M) \leq q\} \end{aligned}$$

3 K-Safety Property

We show that none of the bounding problems are k-safety problems for any k. Informally, a program property is said to be a *k-safety* property [25,8] if it can be refuted by observing k number of (finite) execution traces. A k-safety problem is the problem of checking a k-safety property. Note that the standard safety property is a 1-safety property. An important property of a k-safety problem is that it can be reduced to a standard safety (i.e., 1-safety) problem, such as the unreachability problem, via a simple program transformation called *self composition* [3,10]. This allows one to verify k-safety problems by applying powerful automated safety verification techniques [2,13,20,4] that have made remarkable progress recently.

As stated earlier, we prove that no bounding problem is a k-safety property for any k. To put the result in perspective, we compare it to the results of the related problems, summarized below. Here, $\mathcal{X}$ is $SE[U]$, $ME[U]$, $GE[U]$, or CC, and $\mathcal{Y}$ is SE, ME, or GE. (Recall that U denotes the uniform distribution.)

(1) Checking non-interference is a 2-safety problem, but it is not 1-safety.
(2) Checking $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ is not a k-safety problem for any k.
(3) Checking $\forall \mu.\mathcal{Y}[\mu](M_1) \leq \mathcal{Y}[\mu](M_2)$ is a 2-safety problem.

The result (1) on non-interference is classic (see, e.g., [19,3,10]). The results (2) and (3) on comparison problems are proven in our recent paper [29]. Therefore, this section's results imply that the bounding problems are harder to verify (at least, via the self-composition approach) than non-interference and the comparison problems for the entropy-based definitions of quantitative information flow with universally quantified distributions.

Formally, k-safety property is defined as follows.

Definition 11 (k-safety property). *We say that a property $P \subseteq Prog \times \mathbb{R}^+$ is a k-safety property iff $(M, q) \notin P$ implies that there exists $T \subseteq [\![M]\!]$ such that $|T| \leq k$ and $\forall M'.T \subseteq [\![M']\!] \Rightarrow (M', q) \notin P$.*

Here, *Prog* denotes the set of all programs, and $\mathbb{R}^+$ is the set of positive real numbers. $[\![M]\!]$ denotes the semantics (i.e., traces) of M, represented by the set of input/output pairs $\{((h, \ell), o) \mid h \in \mathbb{H}, \ell \in \mathbb{L}, o = M(h, \ell)\}$. Note that the original definition of k-safety property is only defined over programs [25,8]. However, because the bounding problems take the additional input q, we extend the notion to account for the extra parameter.

We now state the main results of this section which show that none of the bounding problems are k-safety problems for any k. Because we are interested in hardness, we focus on the case where the distribution is the uniform distribution. That is, the results we prove for the specific case applies to the general case.

Theorem 2. *Neither $B_{SE}[U]$, $B_{ME}[U]$, $B_{GE}[U]$, nor B_{CC} is a k-safety property for any k such that $k > 0$.*

We defer the details of the theorem to Section 3.1 (see also Section 5.2) as it can actually be obtained as a corollary of its results.

3.1 K-Safety under a Constant Bound

The result above appears to suggest that the bounding problems are equally difficult for all the definitions of quantitative information flow. However, holding the parameter q constant (rather than having it as an input) paints a different picture. We show that the problems become k-safety for different definitions for different k's under different conditions in this case.

First, for q fixed, we show that the bounding problem for the channel-capacity based definition of quantitative information flow is k-safety for $k = \lfloor 2^q \rfloor + 1$. (Also, this bound is tight.)

Theorem 3. *Let q be a constant. Then, B_{CC} is $\lfloor 2^q \rfloor + 1$-safety, but it is not k-safety for any $k \leq \lfloor 2^q \rfloor$.*

We briefly explain the intuition behind the above result. Recall that a problem being k-safety means the existence of a *counterexample* trace set of size at most k. That is, for $(M, q) \notin B_{CC}$, we have $T \subseteq [\![M]\!]$ such that $|T| \leq \lfloor 2^q \rfloor + 1$ such that any program that also contains T as its traces also does not belong to B_{CC} (with q), that is, its channel-capacity-based quantitative information flow

is greater than q. Then, the above result follows from the fact that the channel-capacity-based quantitative information flow coincides with the maximum over the low security inputs of the logarithm of the number of outputs [16], therefore, any T containing $\lfloor 2^q \rfloor + 1$ traces of the same low security input and disjoint outputs is a counterexample.

For concreteness, we show how to check B_{CC} via self composition. Suppose we are given a program M and a positive real q. We construct the self-composed program M' shown below.

$$\begin{array}{l} M'(H_1, H_2, \ldots, H_n, L) \equiv \\ \quad O_1 := M(H_1, L); O_2 := M(H_2, L); \ldots; O_n := M(H_n, L); \\ \quad \mathsf{assert}(\bigvee_{i,j \in \{1,\ldots,n\}} (O_i = O_j \wedge i \neq j)) \end{array}$$

where $n = \lfloor 2^q \rfloor + 1$. In general, a self composition involves making k copies the original program so that the resulting program would generate k traces of the original (having the desired property). By the result proven by Malacaria and Chen [16], it follows that M' does not cause an assertion failure iff $(M, q) \in B_{CC}$.

Next, we show that for programs without low security inputs, $B_{ME}[U]$ and $B_{GE}[U]$ are also both k-safety problems (but for different k's) when q is held constant.

Theorem 4. *Let q be a constant, and suppose $B_{ME}[U]$ only takes programs without low security inputs. Then, $B_{ME}[U]$ is $\lfloor 2^q \rfloor + 1$-safety, but it is not k-safety for any $k \leq \lfloor 2^q \rfloor$.*

Theorem 5. *Let q be a constant, and suppose $B_{GE}[U]$ only takes programs without low security inputs. If $q \geq \frac{1}{2}$, then, $B_{GE}[U]$ is $\lfloor \frac{(\lfloor q \rfloor + 1)^2}{\lfloor q \rfloor + 1 - q} \rfloor + 1$-safety, but it is not k-safety for any $k \leq \lfloor \frac{(\lfloor q \rfloor + 1)^2}{\lfloor q \rfloor + 1 - q} \rfloor$. Otherwise, $q < \frac{1}{2}$ and $B_{GE}[U]$ is 2-safety, but it is not 1-safety.*

The result for $ME[U]$ follows from the fact that for programs without low security inputs, the min-entropy based quantitative information flow with the uniform distribution is actually equivalent to the channel-capacity based quantitative information flow [24]. The result for $GE[U]$ may appear less intuitive, but, the key observation is that, like the channel-capacity based definition and the min-entropy based definition with the uniform distribution (for the case without low security inputs), for any set of traces $T = [\![M]\!]$, the information flow of a program containing T would be at least as large as that of M. Therefore, by holding q constant, we can always find a large enough counterexample T. The reason $B_{GE}[U]$ is 2-safety for $q < \frac{1}{2}$ is because, in the absence of low security inputs, the minimum non-zero quantity of $GE[U](M)$ is bounded (by 1/2), and so for such q, the problem $GE[U](M) \leq q$ is equivalent to checking non-interference.[6]

But, when low security inputs are allowed, neither $B_{ME}[U]$ nor $B_{GE}[U]$ are k-safety for any k, even when q is held constant.

[6] In fact, the minimum non-zero quantity property also exists for ME[U] without low security inputs and CC. There, the minimum non-zero quantity is 1, which agrees with the formulas given in the theorems.

Theorem 6. *Let q be a constant. (And let $B_{ME}[U]$ take programs with low security inputs.) Then, $B_{ME}[U]$ is not a k-safety property for any $k > 0$.*

Theorem 7. *Let q be a constant. (And let $B_{GE}[U]$ take programs with low security inputs.) Then, $B_{GE}[U]$ is not a k-safety property for any $k > 0$.*

Finally, we show that the Shannon-entropy based definition (with the uniform distribution) is the hardest of all the definitions and show that its bounding problem is not a k-safety property for any k, with or without low-security inputs, even when q is held constant.

Theorem 8. *Let q be a constant, and suppose $B_{SE}[U]$ only takes programs without low security inputs. Then, $B_{SE}[U]$ is not a k-safety property for any $k > 0$.*

Intuitively, Theorems 6, 7, and 8 follow from the fact that, for these definitions, given any potential counterexample $T \subseteq [\![M]\!]$ to show $(M, q) \notin B_{\mathcal{X}}$, it is possible to find M' containing T whose information flow is arbitrarily close to 0 (and so $(M', q) \in B_{\mathcal{X}}$). See Section 5.2 for further discussion.

Because k tends to grow large as q grows for all the definitions and it is impossible to bound k for all q, this section's results are unlikely to lead to a practical verification method. Nevertheless, the results reveal interesting disparities among the different proposals for the definition of quantitative information flow.

4 Complexities for Loop-Free Boolean Programs

In this section, we analyze the computational complexity of the bounding problems when the programs are restricted to loop-free boolean programs. The purpose of the section is to compare the complexity theoretic hardness of the bounding problems with those of the related problems for the same class of programs, as we have done with the k-safety property of the problems.

That is, we compare against the comparison problems of quantitative information flow and the problem of checking non-interference for loop-free boolean programs. The complexity results for these problems are summarized below. Here, $\mathcal{X}$ is $SE[U]$, $ME[U]$, $GE[U]$, or CC, and $\mathcal{Y}$ is SE, ME, or GE.

(1) Checking non-interference is coNP-complete
(2) Checking $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ is PP-hard.
(3) Checking $\forall\mu.\mathcal{Y}[\mu](M_1) \leq \mathcal{Y}[\mu](M_2)$ is coNP-complete.

The results (1) and (3) are proven in our recent paper [29]. The result (2) tightens our (oracle relative) #P-hardness result from the same paper, which states that for each C such that C is the comparison problem for $SE[U]$, $ME[U]$, $GE[U]$, or CC, we have $\#\mathrm{P} \subseteq \mathrm{FP}^C$. (Recall that the notation FP^A means the complexity class of function problems solvable in polynomial time with an oracle for the problem A.) #P is the class of counting problems associated with NP. PP is the class of decision problems solvable in probabilistic polynomial time. PP is known

$$
\begin{array}{rll}
M & ::= x := \psi \mid M_0; M_1 & wp(x := \psi, \phi) = \phi[\psi/x] \\
& \mid \ \mathsf{if}\ \psi\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1 & wp(\mathsf{if}\ \psi\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1, \phi) \\
\phi, \psi & ::= \mathsf{true} \mid x \mid \phi \wedge \psi \mid \neg\phi & \quad = (\psi \Rightarrow wp(M_0, \phi)) \wedge (\neg\psi \Rightarrow wp(M_1, \phi)) \\
& & wp(M_0; M_1, \phi) = wp(M_0, wp(M_1, \phi))
\end{array}
$$

Fig. 1. The syntax and semantics of loop-free boolean programs

to contain both coNP and NP, PH $\subseteq \mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$ [26], and PP is believed to be strictly larger than both coNP and NP. (In particular, PP = coNP would imply the collapse of the polynomial hierarchy (PH) to level 1.)

We show that, restricted to loop-free boolean programs, the bounding problems for the entropy-based definitions with the uniform distribution (i.e., $SE[U]$, $ME[U]$, and $GE[U]$) and the channel-capacity based definition (i.e., CC) are all PP-hard. The results strengthen the hypothesis that the bounding problems for these definitions are quite hard. Indeed, they show that they are complexity theoretically harder than non-interference and the comparison problems with the universally quantified distributions for loop-free boolean programs, assuming that coNP and PP are separate.

We define the syntax of loop-free boolean programs in Figure 1. We assume the usual derived formulas $\phi \Rightarrow \psi$, $\phi = \psi$, $\phi \vee \psi$, and false. We give the usual weakest precondition semantics in the figure.

To adapt the information flow framework to boolean programs, we make each information flow variable H, L, and O range over functions mapping boolean variables of its kind to boolean values. For example, if x and y are low security boolean variables and z is a high security boolean variable, then L ranges over the functions $\{x, y\} \to \{\mathsf{false}, \mathsf{true}\}$, and H and O range over $\{z\} \to \{\mathsf{false}, \mathsf{true}\}$.[7] (Every boolean variable is either a low security boolean variable or a high security boolean variable.) We write $M(h, \ell) = o$ for an input (h, ℓ) and an output o if $(h, \ell) \models wp(M, \phi)$ for a boolean formula ϕ such that $o \models \phi$ and $o' \not\models \phi$ for all output $o' \neq o$. Here, $\models$ is the usual logical satisfaction relation, using h, ℓ, o, etc. to look up the values of the boolean variables. (Note that this incurs two levels of lookup.)

As an example, consider the following program.

$$M \ \equiv \ z \ \texttt{:=}\ x; w \ \texttt{:=}\ y; \texttt{if}\ x \wedge y\ \texttt{then}\ z \ \texttt{:=}\ \neg z\ \texttt{else}\ w \ \texttt{:=}\ \neg w$$

Let x, y be high security variables and z, w be low security variables. Then,

$$
\begin{array}{ll}
SE[U](M) = 1.5 & GE[U](M) = 1.25 \\
ME[U](M) = \log 3 \approx 1.5849625 & CC(M) = \log 3 \approx 1.5849625
\end{array}
$$

[7] We do not distinguish input boolean variables from output boolean variables. But, a boolean variable can be made output-only by assigning a constant to the variable at the start of the program and made input-only by assigning a constant at the end.

We now state the main results of the section, which show that the bounding problems are PP-hard for all the definitions of quantitative information flow considered in this paper.

Theorem 9. $PP \subseteq B_{SE}[U]$

Theorem 10. $PP \subseteq B_{ME}[U]$

Theorem 11. $PP \subseteq B_{GE}[U]$

Theorem 12. $PP \subseteq B_{CC}$

We remind that the above results hold (even) when the bounding problems $B_{SE}[U]$, $B_{ME}[U]$, $B_{GE}[U]$, and B_{CC} are restricted to loop-free boolean programs. We also note that the results hold even when the programs are restricted to those without low security inputs. These results are proven by a reduction from MAJSAT, which is a PP-complete problem. MAJSAT is the problem of deciding, given a boolean formula ϕ over variables $\overrightarrow{x}$, if there are more than $2^{|\overrightarrow{x}|-1}$ satisfying assignments to ϕ (i.e., whether the majority of the assignments to ϕ are satisfying).

5 Discussion

5.1 Bounding the Domains

The notion of k-safety property, like the notion of safety property from where it extends, is defined over all programs regardless of their size. (For example, non-interference is a 2-safety property for all programs and unreachability is a safety property for all programs.) But, it is easy to show that the bounding problems would become "k-safety" properties if we constrained and bounded the input domains because then the size of the semantics (i.e., the input/output pairs) of such programs would be bounded by $|\mathbb{H}|\times|\mathbb{L}|$. In this case, the problems are at most $|\mathbb{H}|\times|\mathbb{L}|$-safety.[8] (And the complexity theoretic hardness degenerates to a constant.) But, like the k-safety bounds obtained by fixing q constant (cf. Section 3.1), these bounds are high for all but very small domains and are unlikely to lead to a practical verification method. Also, because a bound on the high security input domain puts a bound on the maximum information flow, the bounding problems become a tautology for $q \geq c$, where c is the maximum information flow for the respective definition.

5.2 Low Security Inputs

Recall the results from Section 3.1 that, under a constant bound, the bounding problems for both the min-entropy based definition and the guessing entropy-based definition with the uniform distribution are k-safety for programs without

[8] It is possible to get a tighter bound for the channel-capacity based definition by also bounding the size of the output domain.

low security inputs, but not for those with. The reason for the non-k-safety results is that the definitions of quantitative information flow ME and GE (and in fact, also SE) use the conditional entropy over the low security input distribution and are parameterized by the distribution. This means that the quantitative information flow of a program is averaged over the low security inputs according to the distribution. Therefore, by arbitrarily increasing the number of low security inputs, given any set of traces T, it becomes possible to find a program containing T whose information flow is arbitrarily close to 0 (at least under the uniform distribution). This appears to be a property intrinsic to any definition of quantitative information flow defined via conditional entropy over the low security inputs and is parameterized by the distribution of low security inputs. Note that the channel-capacity based definition does not share this property as it is defined to be the maximum over the distributions. The non-k-safety result for $B_{SE}[U]$ holds even in the absence of low security inputs because the Shannon entropy of a program is the average of the *surprisal* [7] of the individual observations, and so by increasing the number of high security inputs, given any set of traces T, it becomes possible to find a program containing T whose information flow is arbitrarily close to 0.

6 Related Work

This work continues our recent research [29] on investigating the hardness and possibilities of verifying quantitative information flow according to the formal definitions proposed in literature [11,6,15,24,14,1,18,16,22]. Much of the previous research has focused on information theoretic properties of the definitions and proposed approximate (i.e., incomplete and/or unsound) methods for checking and inferring quantitative information flow according to such definitions. In contrast, this paper (along with our recent paper [29]) investigates the hardness and possibilities of precisely checking and inferring quantitative information flow according to the definitions.

This paper has shown that the bounding problem, that is, the problem of checking $\mathcal{X}(M) \leq q$ given a program M and a positive real q, is quite hard (for various quantitative information flow definitions $\mathcal{X}$). This is in contrast to our previous paper that has investigated the hardness and possibilities of the comparison problem, that is, the problem of checking $\mathcal{X}(M_1) \leq \mathcal{X}(M_2)$ given programs M_1 and M_2. To the best of our knowledge, this paper is the first to investigate the hardness of the bounding problems. But, the hardness of quantitative information flow inference, a harder problem, follows from the results of our previous paper, and Backes et al. [1] have also proposed a precise inference method that utilizes self composition and counting algorithms.

While the focus of the work is on verification, in the light of the disparities among the different definitions (cf. Section 3.1 and Section 5), it may be interesting to judge the different proposals based on the hardness of verification. Researchers have also proposed definitions of quantitative information flow that are not considered in the paper. These include the definition based on the notion

of *belief* [7], and the ones that take the maximum over the low security inputs [15,14]. These are subjects of future study.

7 Conclusions and Future Work

In this paper, we have formalized and proved the hardness of the bounding problem of quantitative information flow, which is a form of (precise) checking problem of quantitative information flow. We have shown that no bounding problem is a k-safety property for any k, and therefore that it is not possible to reduce the problem to a safety problem via self composition, at least when the quantity to check against is unrestricted. The result is in contrast to non-interference and the comparison problem for the entropy-based quantitative information flow with universally quantified distribution, which are 2-safety properties. We have also shown a complexity theoretic gap with these problems, which are coNP-complete, by proving the PP-hardness of the bounding problems, when restricted to loop-free boolean programs.

We have also shown that the bounding problems for some quantitative information flow definitions become k-safety for different k's under certain conditions when the quantity to check against is restricted to be a constant, highlighting interesting disparities among the different definitions of quantitative information flow.

A possible future research direction is to investigate the entropy-based bounding problems with their distributions universally quantified, that is, the problem of deciding if $\forall\mu.\mathcal{Y}[\mu](M) \leq q$ where $\mathcal{Y}$ is instantiated with SE, ME, or GE. This is partly motivated by our recent work [29] that has obtained remarkable results by universally quantifying over the distributions in the entropy-based definitions in the comparison problems. (That is, checking $\forall\mu.SE[\mu](M_1) \leq SE[\mu](M_2)$, $\forall\mu.ME[\mu](M_1) \leq ME[\mu](M_2)$, and $\forall\mu.GE[\mu](M_1) \leq GE[\mu](M_2)$ are all equivalent and 2-safety, and so that they can all be checked simultaneously via self composition, and that they are coNP-complete when restricted to loop-free boolean programs –cf. Section 1–.) We actually already know the answer for the Shannon-entropy based definition. That is, $\forall\mu.SE[\mu](M) \leq q$, as this is simply is equivalent to $CC(M) \leq q$, the channel-capacity bounding problem. The problem is open for the other two entropy-based definitions of quantitative information flow.

References

1. Backes, M., Köpf, B., Rybalchenko, A.: Automatic discovery and quantification of information leaks. In: IEEE Symposium on Security and Privacy, pp. 141–153 (2009)
2. Ball, T., Rajamani, S.K.: The SLAM project: debugging system software via static analysis. In: POPL, pp. 1–3 (2002)
3. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. In: CSFW, pp. 100–114 (2004)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. STTT 9(5-6), 505–525 (2007)
5. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. Electr. Notes Theor. Comput. Sci. 112, 149–166 (2005)

6. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15(3), 321–371 (2007)
7. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Belief in information flow. In: CSFW, pp. 31–45 (2005)
8. Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: CSF, pp. 51–65 (2008)
9. Cohen, E.S.: Information transmission in computational systems. In: SOSP, pp. 133–139 (1977)
10. Darvas, Á., Hähnle, R., Sands, D.: A theorem proving approach to analysis of secure information flow. In: Hutter, D., Ullmann, M. (eds.) SPC 2005. LNCS, vol. 3450, pp. 193–209. Springer, Heidelberg (2005)
11. Denning, D.E.R.: Cryptography and data security. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1982)
12. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
13. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL, pp. 58–70 (2002)
14. Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: CCS, pp. 286–296 (2007)
15. Malacaria, P.: Assessing security threats of looping constructs. In: POPL, pp. 225–235 (2007)
16. Malacaria, P., Chen, H.: Lagrange multipliers and maximum information leakage in different observational models. In: PLAS, pp. 135–146 (2008)
17. Massey, J.L.: Guessing and entropy. In: ISIT, p. 204 (1994)
18. McCamant, S., Ernst, M.D.: Quantitative information flow as network flow capacity. In: PLDI, pp. 193–205 (2008)
19. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: IEEE Security and Privacy, pp. 79–93 (1994)
20. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
21. Naumann, D.A.: From coupling relations to mated invariants for checking information flow. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 279–296. Springer, Heidelberg (2006)
22. Newsome, J., McCamant, S., Song, D.: Measuring channel capacity to distinguish undue influence. In: PLAS, pp. 73–85 (2009)
23. Shannon, C.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423, 623–656 (1948)
24. 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)
25. Terauchi, T., Aiken, A.: Secure information flow as a safety problem. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 352–367. Springer, Heidelberg (2005)
26. Toda, S.: PP is as hard as the polynomial-time hierarchy. SIAM J. Comput. 20(5), 865–877 (1991)
27. Unno, H., Kobayashi, N., Yonezawa, A.: Combining type-based analysis and model checking for finding counterexamples against non-interference. In: PLAS, pp. 17–26 (2006)
28. Yasuoka, H., Terauchi, T.: On bounding problems of quantitative information flow (2010), http://www.kb.ecei.tohoku.ac.jp/~yasuoka
29. Yasuoka, H., Terauchi, T.: Quantitative information flow - verification hardness and possibilities. In: CSF (2010)

On E-Vote Integrity in the Case of Malicious Voter Computers

Sven Heiberg[1], Helger Lipmaa[1,2], and Filip van Laenen[3]

[1] Cybernetica AS, Estonia
[2] Tallinn University, Estonia
[3] Computas AS, Norway

Abstract. Norway has started to implement e-voting (over the Internet, and by using voters' own computers) within the next few years. The vulnerability of voter's computers was identified as a serious threat to e-voting. In this paper, we study the vote integrity of e-voting when the voter computers cannot be trusted. First, we make a number of assumptions about the available infrastructure. In particular, we assume the existence of two out-of-band channels that do not depend on the voter computers. The first channel is used to transmit integrity check codes to the voters prior the election, and the second channel is used to transmit a check code, that corresponds to her vote, back to a voter just after his or her e-vote vast cast. For this we also introduce a new cryptographic protocol. We present the new protocol with enough details to facilitate an implementation, and also present the timings of an actual implementation.

Keywords: Implementation, integrity, malicious voter computers, nationwide e-voting, proxy oblivious transfer, zero-knowledge proofs.

1 Introduction

The first e-voting pilot (that is, voting over the Internet by using voters' own computers) pilot in Norway is currently scheduled for 2011, with plans to have nation-wide e-voting by 2017. As it should be in all democratic countries, Norway aims the electronic elections to be both as accessible/usable and as secure as possible. It is not always easy to reach a sensible compromise. In this paper, we describe our e-voting solution that was proposed to the Norwegian election officials in Summer of 2009. The proposed e-voting protocol tries to find a good compromise between various security and usability.

A nationwide implementation of e-voting has to be secure against as many attacks as possible, and in presence of as many malicious parties as possible without seriously hurting usability or the ease of verifiably correct implementation. Abundant research has been done on the security of e-voting in the presence of malicious voting servers. Thus, this part of e-voting can be considered to be solved to at least certain degree, and thus in this paper, we will not focus on this aspect of e-voting. (The real e-voting will implement additional means to guarantee security against malicious voting servers.)

On the other hand, it is even more difficult to guarantee security in the case when voter computers cannot be trusted. The seeming impossibility of guaranteeing vote privacy and integrity in the presence of malicious voter computers has been one of the main

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 373–388, 2010.

obstacles that has delayed the real-world implementation of e-voting. Moreover, achieving vote privacy in the case of malicious voter computers seems to hurt usability [2], since the value input by a voter to the computer should not reveal voter's preferred candidate. In practice, this amounts to inputing a pseudorandom code (or something similar), unknown to the voter computer but yet securely delivered to the voter himself or herself. Due to both the impossibility of implementing secure yet guaranteed code delivery and to the usability concerns, solutions where a voter is required to enter a random code to the computer are definitely out of the question. In Norway, solutions where the voter could obtain the used random values, and then use her own program to verify the correctness of ciphertexts were not even considered.

Our Contributions. We show that it is possible to guarantee e-vote integrity in the presence of malicious voter computers without drastically changing the user experience, and without the necessity of 100% delivery of random codes (or say, secure hardware tokens). More precisely, we construct a cryptographic protocol at the end of which, after she has entered her vote to the computer, the voter obtains a relatively short integrity check code. Given this check code (and/or the absence or presence of the message itself), the voter can verify the integrity of her vote. This easy verification is the only change in her voting experience as compared to a similar non-secure system: she is not required to enter long codes, nor has the user interface to be particularly clunky. Moreover, in our case, the delivery of the check codes and the subsequent verification is not obligatory: voters who are paranoid enough or just have a reason not to trust either the idea of e-voting, or the security of their own computers, can take additional measures to first obtain the codes and then to perform verification.

We first introduce some organizational assumptions that seem to be necessary and yet practical enough to be implemented. We emphasize that these assumptions ("the necessary evil") have been approved by the Norwegian e-voting project officials. First, Norway has an ongoing parallel process to implement a national public-key infrastructure. This infrastructure will make it possible for the e-voting project to use eID-cards for the authentication of the voters, but not yet for signing the ballots digitally by 2011. This means that for authentication, the same scheme as the one used on the eID-card has to be used, but otherwise, the pilot project is free to use non-standard public-key cryptosystems. It has to be mentioned though that there are some commercial alternatives available that offer digital signature functionality, but it is unclear whether the public will be willing to trust commercial vendors to sign their ballots.

Second, we require the existence of two secure and authenticated channels $\mathsf{prechannel}$ and $\mathsf{postchannel}$. Briefly, before the elections, every voter v gets a list of candidates cnd together with integrity check codes $\mathsf{Code}_v[\mathsf{cnd}]$, where the voter-dependent codes are random and independent. The codes are transfered to all voters over a secure and authenticated $\mathsf{prechannel}$ that is unlikely to be controlled by the same attacker that controls her computer. This is not restrictive in Norway, where voter registration cards are mailed to all voters in advance (and people trust the postal system). Once more, the delivery of check codes to *all* voters is not necessary: we just assume that a large majority of voters have access to the $\mathsf{prechannel}$ by default, and other voters (who are still sufficiently interested in e-voting security) must take a special action to obtain the codes. In principle, there are several alternative ways to build $\mathsf{prechannel}$, but the important requirement is

that the check codes should not be known by the voter's computer. Alternatives include using secure Web pages (available only when accessed by using say a smartphone for which the real e-voting client is not available), or SMSs from a fixed mobile number.

Moreover, a real-time channel postchannel (say, SMS, or a Web page that can be checked by using a smartphone) is used to inform the voter about the success of her actions. More precisely, every time she has voted, an integrity check code is sent to her by using postchannel. Note that in Norway, virtually every voter has a mobile phone with the mobile number known to the government—namely, they are extensively used for tax payment—, and thus there exists an efficient postchannel. Those voters whose mobiles have not been registered yet, but who are interested in e-voting security, have to take additional action. However, voters can choose not to do it. Also, a message from postchannel makes sense even if the voter has not received the original codes from the prechannel: in this case, she at least knows that her vote has been recorded.

In addition, the Norwegian e-voting procedure will allow the voters to revote either electronically—such that later e-vote takes precedence over an earlier e-vote—or by (later) participating in conventional paper voting (p-voting), which will take precedence over e-votes. This will provide at least some (though not complete) protection against vote buying and coercion: if either of these has happened, the voter can choose to revote later by using either an e-vote or a p-vote. (The p-voting period will start several days after the e-voting period has ended.) Clearly, if the voter can be both physically coerced (to the extent where she cannot go and participate in p-voting) and she cannot trust her computer, then she cannot be completely protected against all frauds. However, the revoting procedure, which is already implemented in Estonian national e-voting procedure, offers at least some protection against vote buying and coercion. Moreover, due to the existence of the postchannel, a voter will get a timely notification when her vote was altered by her computer. In this case, she can use a different computer to revote, or when necessary, participate in p-voting. Therefore, the combination of a quick-response postchannel and revoting not only guarantees fraud detection but also allows the voters to act on it.

On the flip side, every voter can legally use the same PC to vote many times for (not necessarily) different candidates. This limits the choice of postchannel in our case significantly. In particular, it is not secure to use the (possibly malicious) PC itself as the postchannel. Namely, assume that the voter votes for candidate A, then is coerced to vote for B, and then votes again for A. The PC, already knowing the integrity check codes of A and B, can submit a vote for B but display the integrity check code for A.

Given those organizational assumptions, we consider the next setting. Voter's ballot (vote) is encrypted and signed (possibly by the attacker), and then sent to the vote collector. (Without loss of generality, in this paper we will assume that there is a single vote collector. In practice, there will be more, but all our protocols will naturally generalize. We will not mention this important point anymore.) The vote collector computes, given an encrypted and signed vote, a ciphertext of the integrity check code $\mathsf{Code}_v[\mathsf{cnd}]$ and sends it to another server (called the messenger). The messenger decrypts the code, and then sends an SMS alert of the type "You, [name], voted at [time], the check code is $\mathsf{Code}_v[\mathsf{cnd}]$ " to the voter over postchannel. The voter verifies the correctness: she complains when she got a wrong message over postchannel (which say contains a wrong

check code), or did not get it all when she voted (in particular when her computer tells her that the vote collector is unavailable), or gets a message when she did not vote. Here, we need that the messenger, who can be behind a firewall, is unaware of the correspondence between the candidates and the corresponding check codes. I.e., a malicious messenger should not collaborate with a malicious vote collector.

In Sect. 4, we propose a cryptographic protocol by which the messenger obtains $\mathsf{Code}_v[\mathsf{cnd}]$. The basic idea of the protocol is as follows. Voter's computer sends to the vote collector two ciphertexts that "encrypt" cnd, one with tallier's public key, and another one with messenger's public key. This is accompanied by a non-interactive zero-knowledge (NIZK) proof of knowledge that the two encrypted values are equal and belong to the correct range (i.e., correspond to a valid candidate). The corresponding full NIZK proof of knowledge is presented in Sect. 3.2, and its full security proof is given in an appendix. When the NIZK proof of knowledge is correct, the vote collector cryptocomputes, based on the second ciphertext, a ciphertext of $\mathsf{Code}_v[\mathsf{cnd}]$ that is encrypted by messenger's public key. This is done by using a "proxy oblivious transfer" protocol [13] with the additional requirement that the proxy should not get to know the index used by the chooser even when he knows the whole unordered database. The vote collector then sends an encryption of cnd (under tallier's public key) to the tallier, and an encryption of $\mathsf{Code}_v[\mathsf{cnd}]$ (under messenger's public key) to the messenger. In Sect. 4, the new protocol is presented in sufficient details to facilitate an implementation.

We then give an informal security assessment of the full integrity check protocol, and explain our choice of underlying cryptographic primitives and protocols. In this paper, we are *not* going to discuss the operation of tallier since there is a decent amount of literature on this part of the e-voting process. However, we stress that the full e-voting solution in Norway must use additional cryptographic protocols to guarantee better security against malicious voting servers (i.e., vote collectors, talliers, and messengers).

We finish the paper by describing an implementation of the new integrity check protocol, and by giving the timings in the case where there is both a small and a large number of voters and candidates. For example, if there are 80 candidates, the vote collector's throughput is around $2\,000$ votes per hour on our test machine. The throughput can be increased dramatically by using several vote collectors, better (faster and multicore) CPUs, or even hardware acceleration. In particular, our next task consists of implementing the described protocol in a commercial Hardware Security Module.

Risk Assessment: Avoided Attacks Versus New Attacks. Without the use of the new protocol (or something similar), the voters will not be informed at all whether their e-votes reached the voting servers. Thus, a malicious entity (say some foreign government, or a terrorist organization) can mount a full-scale attack (by writing malicious software that covertly takes over many of voter computers) on the e-voting process and stay undetected. Alternatively, they may reveal themselves after the end of the elections and prove that they in fact manipulated the elections — even that case would be quite devastating. If the integrity protocol of this paper is implemented, such attacks will all be at least detected—given that sufficiently many voters verify the codes—, and the voters can also react by revoting on paper if necessary.

The new protocol also creates some genuinely new attacks. For example, an attacker can take over the prechannel (for example, by distributing fake voter registration cards) or the postchannel (by massively distributing fake SMSs). Both attacks are arguably much more difficult to perform without detection than the takeover of voter computers, since they at least require some physical presence. Attacks on only the postchannel basically amount to the voters receiving bogus messages with (very high probability) wrong check codes. In this case the voters will be alerted, and can revote. Even if both channels are successfully attacked (which is quite difficult by an outsider in the case the prechannel is implemented by using "snail mail" and the postchannel is implemented by using SMSs), there is no more harm done than by attacking voter computers: the attacker can then both break correctness (by just reordering codes sent by the prechannel) and anonymity, but both can done trivially by just a malicious computer.

Finally, there are some genuinely new attacks which more hinge on human psychology than cryptography or computer security in general. As an example, voters can falsely claim that they received wrong codes, and thus cause alarm and distrust in elections. Here we emphasize, that the new protocol makes it possible for voters to detect attacks (so that they can revote) but in most of the cases, not to prove their presence. (With some exceptions, such as when they receve incorrectly formatted SMSs from the correct mobile number.) In our own opinion, due to this attack, voter complaints should thus always taken with a grain of salt: if such a complaint occurs, then clearly either there was an attack by an outsider or the voter herself. This should be explained to the voters before the e-voting. Moreover, without such a protocol, any voter can (legitimately) claim that she does not trust e-voting since she may have a virus — and that the government has done nothing to protect her in such a case. We think that the latter complaint is much more valid.

Due to the lack of space, many details have been omitted. They can be found in the full version [9].

2 Cryptographic Preliminaries

Notation. All logarithms are on basis 2. k is the security parameter, we assume that $k = 80$. $x \leftarrow X$ denotes assignment; if X is a set or a randomized algorithm, then $x \leftarrow X$ denotes a random selection of x from the set or from the possible outputs of X as specified by the algorithm. In the case of integer operations, we will explicitly mention the modulus, like in $z \leftarrow a + b \mod q$. On the other hand, we will omit modular reduction in the case of group operations (like $h \leftarrow g^r$), since in this case depending on the group, reduction may or may not make sense.

Hash Functions, Random Oracle Model and Key Derivation Functions. A function $H : A \rightarrow B$ is a hash function if $|B| < |A|$. Within this paper, we usually need to assume that H is a random oracle. I.e., the value of $H(x)$ is completely unpredictable if one has not seen $H(x)$ before. Random oracles are useful in many cryptographic applications, by making it possible to design efficient cryptographic protocols. In practice, one would instantiate H with a strong cryptographic hash function like SHA2 or the future winner of the SHA3 competition. While there exist schemes which are secure in the random oracle model but which are insecure given any "real" function [5], all

known examples are quite contrived. A *key derivation function* $\mathsf{Kdf} : A \to B$ takes a random element from set A and outputs a pseudorandom element in set B. If $|B| < |A|$ then Kdf is a pseudorandom function, but if $|B| \geq |A|$ then Kdf can be constructed without any cryptographic assumptions. See, e.g., [6]. For the sake of simplicity, we think of Kdf as a random oracle.

Signature Schemes. A signature scheme $\mathsf{SC} = (\mathsf{Gen}^{\mathsf{sc}}, \mathsf{Sign}, \mathsf{Ver})$ is a triple of efficient algorithms, where $\mathsf{Gen}^{\mathsf{sc}}$ is a randomized key generation function, Sign is a (possibly randomized) signing algorithm and Ver is a verification algorithm. A signature scheme is EUF-CMA (existentially unforgeable against chosen message attacks) secure, if it is computationally infeasible to generate a new signature (i.e., a signature to a message that was not queried from the oracle), given an access to an oracle who signs messages chosen by the adversary. For the purpose of this paper, any of the well-known EUF-CMA secure signature schemes can be used. However, since e-voting is most probably going to use the existing PKI infrastructure of the relevant country, the most prudent approach is to rely on whatever signature scheme has been implemented in the corresponding ID-cards.

Public-Key Cryptosystems. Let $\mathsf{PKC} = (\mathsf{Gen}^{\mathsf{pkc}}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key cryptosystem, where $\mathsf{Gen}^{\mathsf{pkc}}$ is a randomized key generation algorithm that on input $(1^k; r)$, for some random string r, outputs a new secret/public key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}^{\mathsf{pkc}}(1^k; r)$, Enc is a randomized encryption algorithm with $c = \mathsf{Enc}_{\mathsf{pk}}(m; r')$, and Dec is a decryption algorithm with $\mathsf{Dec}_{\mathsf{sk}}(c) = m'$. It is required that if $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}^{\mathsf{pkc}}(1^k; r)$ then $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m; r')) = m$ for all valid m, r and r'. We denote $\mathsf{Enc}_{\mathsf{pk}}(m; r)$ (resp., $\mathsf{Gen}^{\mathsf{pkc}}(1^k; r)$) for a randomly chosen r also just as $\mathsf{Enc}_{\mathsf{pk}}(m)$ (resp., $\mathsf{Gen}^{\mathsf{pkc}}(1^k)$).

In the case of the Elgamal cryptosystem [7], one fixes a cyclic group $\mathbb{G}$ of a prime order $2^{2k+1} > q > 2^{2k}$, together with a generator g of $\mathbb{G}$. Then, $\mathsf{Gen}^{\mathsf{pkc}}(1^k)$ generates a random $\mathsf{sk} \leftarrow \mathbb{Z}_q$, and sets $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. On input $m \in \mathbb{G}$, the encryption algorithm generates a new random $r \leftarrow \mathbb{Z}_q$, and sets $\mathsf{Enc}_{\mathsf{pk}}(m; r) := (m \cdot \mathsf{pk}^r, g^r)$. On input $c = (c_1, c_2) \in \mathbb{G}^2$, the decryption algorithm outputs $m' \leftarrow c_1 / c_2^{\mathsf{sk}}$. Elgamal is multiplicatively homomorphic. I.e., $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1; r_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2; r_2)) = m_1 \cdot m_2$ for $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{Gen}^{\mathsf{pkc}}(1^k)$. Further discussion is provided in the full version [9].

Non-Interactive Zero-Knowledge Proof of Knowledge. Let L be an arbitrary NP-language, and let $R = \{(x, y)\}$ where $x \in L$ and y is the corresponding NP-witness. A Σ-protocol (P_1, V_1, P_2, V_2) for a relation R is a three-message protocol between a prover and a verifier (both stateful), such that (1) the prover and verifier have a common input x, and the prover has a private input y, (2) the prover sends the first (P_1) and the third (P_2) message, and the verifier sends the second message V_1, after which the verifier either rejects or accepts (by using V_2), (3) the protocol is public-coin: i.e., the verifier chooses her response V_1 completely randomly from some predefined set, (4) the protocol satisfies the security properties of correctness, special honest-verifier zero-knowledge (SHVZK), and special soundness. We identify a protocol run with the tuple $(x; \mathfrak{i}, \mathfrak{c}, \mathfrak{r})$ where $(\mathfrak{i}, \mathfrak{c}, \mathfrak{r})$ are the three messages of this protocol. A protocol run is *accepting*, if an honest verifier accepts this run, i.e., on having input x and seeing the messages $\mathfrak{i}$, $\mathfrak{c}$, and $\mathfrak{r}$.

Based on an arbitrary Σ-protocol, one can build a non-interactive zero-knowledge (NIZK) proof of knowledge in the random oracle model, by using the Fiat-Shamir heuristic. I.e., given $(x, y) \in R$ and a random oracle H [3], the corresponding NIZK proof of knowledge π consists of $(\mathfrak{i}, \mathfrak{c}, \mathfrak{r})$, where $\mathfrak{i} \leftarrow P_1(x, y)$, $\mathfrak{c} \leftarrow H(param, x, \mathfrak{i})$, and $\mathfrak{r} \leftarrow P_2(x, y, \mathfrak{c})$, where $param$ is the set of public parameters (like the description of the underlying group, etc).

We use the next common notation. A NIZK proof of knowledge $\mathrm{PK}(R(\dots))$ is for relation R, where the prover has to prove the knowledge of variables denoted by Greek letters. All other variables are known to both the prover and the verifier. For example, $\mathrm{PK}(y = \mathsf{Enc}_{\mathsf{pk}}(\mu; \rho) \wedge \mu \in \{0, 1\})$ denotes a NIZK proof of knowledge that the prover knows a Boolean μ and some ρ such that $y = \mathsf{Enc}_{\mathsf{pk}}(\mu; \rho)$.

NIZK Proof of Equality of Plaintexts. Let $\mathsf{PKC} = (\mathsf{Gen}^{\mathsf{pkc}}, \mathsf{Enc}, \mathsf{Dec})$ be the Elgamal cryptosystem. Fix $\mathbb{G}$, g, and two key pairs $(\mathsf{sk}_1, \mathsf{pk}_1) \in \mathsf{Gen}^{\mathsf{pkc}}(1^k)$ and $(\mathsf{sk}_2, \mathsf{pk}_2) \in \mathsf{Gen}^{\mathsf{pkc}}(1^k)$. Let H be a random oracle. The NIZK proof of equality of plaintext is a NIZK proof of knowledge $\mathrm{PK}(e_1 = \mathsf{Enc}_{\mathsf{pk}_1}(g^\mu; \rho_1) \wedge e_2 = \mathsf{Enc}_{\mathsf{pk}_2}(g^\mu; \rho_2))$, that e_1 and e_2 encrypt the same plaintext under a different key.

Range Proof in Exponents. In the following we need a range proof in exponents, i.e., a NIZK proof of knowledge $\mathrm{PK}(e = \mathsf{Enc}_{\mathsf{pk}}(g^\mu; \rho) \wedge \mu \in [0, \mathsf{CC}])$ for some positive integer CC. In the discrete logarithm setting the most efficient known range proof in exponents was proposed in [12]. (Another range proof in exponents that is comparably communication-efficient, was recently proposed in [4]. However, the latter proof uses pairings and is thus computationally less efficient.) The communication complexity of the range proof in exponents from [12] is logarithmic in CC. In the general case (when assuming stronger assumptions), there exist range proofs in exponents with communication that is essentially independent of CC. However, if the value of CC is relatively small, the latter proofs actually are less efficient than the proof of [12].

We specify this proof fully in Sect. 3.1, where we present a NIZK proof of knowledge that uses this range proof in exponents as a subproof.

3 Cryptographic Tools

3.1 Strong Proxy Oblivious Transfer

In a 1-out-of-n proxy oblivious transfer protocol, $(n, 1)$-POT [13], for ℓ-bit strings, the chooser has an index $x \in \{0, \dots, n-1\}$ and a public key pk, the sender has pk and a database $f = (f_0, \dots, f_{n-1})$ with $f_i \in \{0,1\}^\ell$, and the proxy has a decryption key. At the end of the protocol, the proxy obtains f_x. A two-message $(n, 1)$-POT protocol $\Gamma = (\mathsf{Gcpir}, \mathsf{Query}, \mathsf{Reply}, \mathsf{Answer})$ is a quadruple of polynomial-time algorithms, with Gcpir and Query being randomized, such that for any r, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gcpir}(1^k; r)$, x, f and r', $\mathsf{Answer}_{\mathsf{sk}}(x, \mathsf{Reply}_{\mathsf{pk}}(f, \mathsf{Query}_{\mathsf{pk}}(x; r'))) = f_x$. As before, we denote $\mathsf{Gcpir}(1^k) := \mathsf{Gcpir}(1^k; r)$ and $\mathsf{Query}_{\mathsf{pk}}(x) := \mathsf{Query}_{\mathsf{pk}}(x; r')$ for randomly chosen r and r'. Here, the proxy generates the key pair $(\mathsf{sk}, \mathsf{pk})$ and sends pk to the chooser and to the sender. The chooser then sends $\mathsf{Query}_{\mathsf{pk}}(x)$ to the sender, who sends $\mathsf{Reply}_{\mathsf{pk}}(f, \mathsf{Query}_{\mathsf{pk}}(x))$ to the proxy. The proxy obtains f_x by applying $\mathsf{Answer}_{\mathsf{sk}}$.

Semisimulatable Privacy for Strong Proxy Oblivious Transfer. Let $\Gamma = (\mathsf{Gcpir}, \mathsf{Query}, \mathsf{Reply}, \mathsf{Answer})$ be a 2-message $(n, 1)$-POT protocol. Within this work we use the convention of many previous papers on oblivious transfer protocols to only require (semisimulatable) privacy in the malicious model. I.e., chooser's privacy is guaranteed in the sense of indistinguishability (CPA-security), while sender's privacy is guaranteed in the sense of simulatability. We note that POT's privacy definition is a simple modification of the standard OT's semisimulatable privacy definition.

We give an informal definition of semisimulatable privacy. For the *CPA-security* (i.e., the privacy) *of the chooser*, (1) no malicious nonuniform probabilistic polynomial-time sender should be able to distinguish, with non-negligible probability, between the distributions $(\mathsf{pk}, \mathsf{Query}_{\mathsf{pk}}(x_0))$ and $(\mathsf{pk}, \mathsf{Query}_{\mathsf{pk}}(x_1))$ that correspond to any two of chooser's inputs x_0 and x_1 that are chosen by the sender, and (2) no malicious nonuniform probabilistic polynomial-time proxy should be able to distinguish, with non-negligible probability, between the distributions $(\{f\}, \mathsf{sk}, \mathsf{pk}, \mathsf{Reply}_{\mathsf{pk}}(f, \mathsf{Query}_{\mathsf{pk}}(x_0)))$ and $(\{f\}, \mathsf{sk}, \mathsf{pk}, \mathsf{Reply}_{\mathsf{pk}}(f, \mathsf{Query}_{\mathsf{pk}}(x_1)))$ that correspond to any two of chooser's inputs x_0 and x_1 that are chosen by the sender. (Here, $\{f\}$ denotes an unordered version of f.) For *sender-privacy*, we require the existence of an *unbounded* simulator that, given pk, chooser's message Q^*_{pk} and proxy's legitimate output corresponding to this message, generates sender's message that is *statistically* indistinguishable from honest sender's message $\mathsf{Reply}_{\mathsf{pk}}$ in the real protocol; here Q^*_{pk} does not have to be correctly computed. As in earlier papers that use semisimulatable privacy, unboundedness is required mostly so that the simulator could "decrypt" chooser's first message. A protocol is *private* if it is both chooser-private and sender-private.

Instantiation. In the proposed e-voting protocol, the database size n corresponds to the number of candidates, and therefore it is usually small (say $n \leq 64$). This means that it is sufficient to use a POT protocol with linear-in-n communication. (In the case when n is larger, one could consider relying on an underlying oblivious transfer protocol with small polylogarithmic communication like those of [10,8].) On the other hand, it is important to minimize sender's computation. Given those considerations, we base the new POT protocol on the AIR oblivious transfer protocol [1]. The result has (in the case of a small n) good communication and computation, is based on a well-known security assumption (Decisional Diffie-Hellman), and allows one to construct efficient NIZK proofs of knowledge.

Let $\mathsf{PKC} = (\mathsf{Gen}^{\mathsf{pkc}}, \mathsf{Enc}, \mathsf{Dec})$ be the Elgamal cryptosystem, and let $g \in \mathbb{G}$ be a fixed generator of the plaintext group. Chooser's private input is $x \in \{0, \ldots, n-1\}$, and sender's private input is $f = (f_0, \ldots, f_{n-1})$ for $f_i \in \{0,1\}^\ell$ with (relatively) small ℓ. The new $(n, 1)$-strong POT protocol consists of the next steps:

1. The proxy sets $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}^{\mathsf{pkc}}(1^k)$, and sends pk to the chooser and the sender.
2. For $\rho \leftarrow \mathbb{Z}_q$, the chooser sets $e \leftarrow \mathsf{Enc}_{\mathsf{pk}}(g^x; \rho)$, and sends $\mathsf{Query}_{\mathsf{pk}}(x) \leftarrow e$ to the sender.
3. The sender does on input pk and $\mathsf{Query}_{\mathsf{pk}}(x) = e$:
   (a) For every $i \in \{0, \ldots, n-1\}$: generate new random values $r_i, r'_i \leftarrow \mathbb{Z}_q$, set $e_i \leftarrow (\mathsf{Enc}_{\mathsf{pk}}(g^i; 1)/e)^{r_i} \cdot \mathsf{Enc}_{\mathsf{pk}}(g^{f_i}; r'_i)$.
   (b) Send $\mathsf{Reply} = \mathsf{Reply}_{\mathsf{pk}}(f, (\mathsf{pk}, e)) \leftarrow \{e_0, \ldots, e_{n-1}\}$ to the proxy, where the set elements in Reply are given in a random order.

4. For all elements e' in the set Reply, the proxy computes $y \leftarrow \mathsf{Dec}_{\mathsf{sk}}(e')$. He finds an y, such that the discrete logarithm z of y on basis g is small. He outputs z as $\mathsf{Answer}_{\mathsf{sk}}(x, \mathsf{Reply})$.

Note that the sender can precompute the values $\mathsf{Enc}_{\mathsf{pk}}(g^i; 1)$ and $\mathsf{Enc}_{\mathsf{pk}}(g^{f_i}; 1)$, and therefore her online computation is dominated by $2n$ exponentiations in $\mathbb{G}$. (Note that in the actual implementation, this protocol will also be accompanied with a NIZK proof that x is in the correct range.)

Computing discrete logarithm is efficient when all database elements are small, say $\ell \leq 5$, and can be just done by table-lookup by comparing all values y with values g^i for small i. (Discrete logarithm step could be avoided by using an additively homomorphic cryptosystem. However, known additively homomorphic cryptosystems are otherwise considerably less efficient than Elgamal.) Moreover, with an overwhelming probability, there is exactly one element e_j such that the discrete logarithm of $\mathsf{Dec}_{\mathsf{sk}}(e_j)$ is small. Thus, the proxy can just decrypt all values e', and then check them against a precomputed table lookup of g^i for small values of i; the comparison step will take $\Theta(n \cdot \log n)$ elementary operations. Since n is very small, this part is considerably faster than decrypting n different ciphertexts. When using say Lipmaa's [10] oblivious transfer-protocol based POT, the messenger will only have to decrypt a single element and then make $\Theta(\log n)$ comparisons by using binary search. However, the cost of computing Answer will be higher. Our choice is supported by implementation timings (Sect. 7) that show that proxy's time load is much smaller than that of sender. Finally, note that the messenger has to decrypt in average 50% of the elements, and thus his online cost is dominated by $\approx n/2$ exponentiations.

This protocol is clearly both correct and private, given that Elgamal is CPA-secure [1].

Weak POT for Large Database Elements. We also need to use proxy oblivious transfer in a situation, where the database elements are significantly longer, such that computing discrete logarithm (as in the proposed strong POT protocol) will not anymore possible. However, in our application, the proxy is allowed to know an *unordered* version $\{f\}$ of the database f. More precisely, the proxy knows an unordered tuple $F := \{g^{f_0}, \ldots, g^{f_{n-1}}\}$, and for efficiency reasons, we assume that this tuple is sorted. After the end of the POT protocol, he obtains g^{f_x} for some unknown x, and he can verify whether g^{f_x} is equal to some element of F by using binary search, in time $\Theta(\log n)$. However, that does not help him in determining x since F does not contain any information about indexes. We call this protocol a weak oblivious transfer protocol.

3.2 New NIZK Proof of Knowledge

We need a NIZK proof of knowledge $\mathrm{PK}(e = \mathsf{Enc}_{\mathsf{pkt}}(g^\mu; \rho) \wedge e' = \mathsf{Query}_{\mathsf{pkm}}(\mu; \rho') \wedge \mu \in [0, \mathsf{CC}])$, where we use the Elgamal cryptosystem and the new proxy oblivious transfer protocol. Since in the new POT protocol, the first message is just $\mathsf{Enc}_{\mathsf{pkt}}(g^\mu)$, we need to prove an AND of two statements, that e and e' "encrypt" the same value g^μ (under different keys), and that e' encrypts a value g^μ where $\mu \in [0, \mathsf{CC}]$. We already presented both proofs separately. For the sake of completeness, the full interactive version of this zero-knowledge proof is given in Prot. 1. We need actually a NIZK proof of knowledge version of it, which is presented later as Prot. 2.

System parameters: $\mathbb{G}$, q, g.
Common inputs: CC and $\lambda := \lfloor \log_2 \mathsf{CC} \rfloor$, pkt, pkm, e'.
Prover's input: μ, ρ'.

1. Prover does:
   (a) Compute the values $\mu_j \in \{0,1\}$ such that $\mu = \sum_{j=0}^{\lambda} \mu_j \mathsf{CC}_j$ with $\mathsf{CC}_j \leftarrow \lfloor (\mathsf{CC} + 2^j)/2^{j+1} \rfloor$.
   (b) For $j \in \{0,\ldots,\lambda\}$ do:
      i. Generate random $\rho_j, \rho'_j \leftarrow \mathbb{Z}_q$, set $e_j \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_j}; \rho_j)$.
      ii. If $\mu_j = 0$ then: Set $\mathfrak{i}_{0,j} \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(1; \rho'_j)$, $\mathfrak{c}_{1,j} \leftarrow \mathbb{Z}_{2^k}$, $\mathfrak{r}_{1,j} \leftarrow \mathbb{Z}_q$, $\mathfrak{i}_{1,j} \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(1; \mathfrak{r}_{1,j}) \cdot (\mathsf{Enc}_{\mathsf{pkt}}(g; 0)/e_j)^{\mathfrak{c}_{1,j}}$.
      iii. Else if $\mu_j = 1$ then: Set $\mathfrak{i}_{1,j} \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(1; \rho'_j)$, $\mathfrak{c}_{0,j} \leftarrow \mathbb{Z}_{2^k}$, $\mathfrak{r}_{0,j} \leftarrow \mathbb{Z}_q$, $\mathfrak{i}_{0,j} \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(1; \mathfrak{r}_{0,j})/e_j^{\mathfrak{c}_{0,j}}$.
   (c) Generate random $\mu_{\mathsf{and}}, \rho_{\mathsf{and},1}, \rho_{\mathsf{and},2} \leftarrow \mathbb{Z}_q$. Set $\mathfrak{i}_{2,1} \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_{\mathsf{and}}}; \rho_{\mathsf{and},1})$, $\mathfrak{i}_{2,2} \leftarrow \mathsf{Enc}_{\mathsf{pkm}}(g^{\mu_{\mathsf{and}}}; \rho_{\mathsf{and},2})$.
   Send $\mathfrak{i} \leftarrow (e_0,\ldots,e_\lambda, (\mathfrak{i}_{0,0},\mathfrak{i}_{1,0}),\ldots,(\mathfrak{i}_{0,\lambda},\mathfrak{i}_{1,\lambda}), \mathfrak{i}_{2,1}, \mathfrak{i}_{2,2})$ to the verifier.
2. Verifier does: Set $\mathfrak{c} \leftarrow \mathbb{Z}_{2^k}$, send $\mathfrak{c}$ to the prover.
3. Prover does for $j \in \{0,\ldots,\lambda\}$:
   (a) If $\mu_j = 0$ then: Set $\mathfrak{c}_{0,j} \leftarrow \mathfrak{c} - \mathfrak{c}_{1,j} \mod 2^k$, $\mathfrak{r}_{0,j} \leftarrow \rho'_j + \mathfrak{c}_{0,j} \cdot \rho_j \mod q$.
   (b) Else if $\mu_j = 1$ then: Set $\mathfrak{c}_{1,j} \leftarrow \mathfrak{c} - \mathfrak{c}_{0,j} \mod 2^k$, $\mathfrak{r}_{1,j} \leftarrow \rho'_j + \mathfrak{c}_{1,j} \cdot \rho_j \mod q$.
   Let $\rho' \leftarrow \sum \rho_j \mathsf{CC}_j \mod q$ (i.e., $e \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu}; \rho')$). Set $\mathfrak{r}_3 \leftarrow \mu_{\mathsf{and}} + \mathfrak{c} \cdot \mu \mod q$, $\mathfrak{r}_{4,1} \leftarrow \rho_{\mathsf{and},1} + \mathfrak{c} \cdot \rho \mod q$, $\mathfrak{r}_{4,2} \leftarrow \rho_{\mathsf{and},2} + \mathfrak{c} \cdot \rho' \mod q$. Send $\mathfrak{r} \leftarrow (\mathfrak{c}_{0,0},\ldots,\mathfrak{c}_{0,\lambda}, (\mathfrak{r}_{0,0},\mathfrak{r}_{1,0}),\ldots,(\mathfrak{r}_{0,\lambda},\mathfrak{r}_{1,\lambda}), \mathfrak{r}_3, \mathfrak{r}_{4,1}, \mathfrak{r}_{4,2})$ to the verifier.
4. Verifier does:
   (a) Let $e \leftarrow \prod_{j=0}^{\lambda} e_j^{\mathsf{CC}_j}$.
   (b) For $j \in \{0,\ldots,\lambda\}$:
      i. Set $\mathfrak{c}_{1,j} \leftarrow \mathfrak{c} - \mathfrak{c}_{0,j} \pmod{2^k}$.
      ii. If $\mathsf{Enc}_{\mathsf{pkt}}(1; \mathfrak{r}_{0,j}) \neq \mathfrak{i}_{0,j} \cdot e_j^{\mathfrak{c}_{0,j}}$ or $\mathsf{Enc}_{\mathsf{pkt}}(1; \mathfrak{r}_{1,j}) \neq \mathfrak{i}_{1,j} \cdot (e_j/\mathsf{Enc}_{\mathsf{pkt}}(g; 0))^{\mathfrak{c}_{1,j}}$ then: reject.
   (c) If $\mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_3}; \mathfrak{r}_{4,1}) \neq \mathfrak{i}_{2,1} \cdot e^{\mathfrak{c}}$ or $\mathsf{Enc}_{\mathsf{pkm}}(g^{\mathfrak{r}_3}; \mathfrak{r}_{4,2}) \neq \mathfrak{i}_{2,2} \cdot (e')^{\mathfrak{c}}$ then: reject.
   Otherwise: accept.

Protocol 1. Interactive version of the required zero-knowledge proof

Complexity. In Prot. 1, prover's computation is dominated by (at most) $3\lambda + 4$ public-key encryptions and λ exponentiations. Since Elgamal is used, if necessary most of the prover's computation can be done beforehand. However, this should not be necessary in our application, where it is perfectly fine that it takes a minute for the voter's computer to finish computation. Verifier's computation is dominated by $2\lambda + 3$ encryptions, λ of which can be precomputed, and $2\lambda + 2$ exponentiations. In real-world voting, we can in most cases assume that $\lambda \leq 6$, thus verifier's computation is dominated by ≤ 15 encryptions and ≤ 14 exponentiations.

Security. The security of Prot. 1 is a straightforward corollary of known results. However, for the sake of completeness we provide a complete proof.

Theorem 1. *Prot. 1 is a correct, specially sound and SHVZK proof of knowledge for* $\mathrm{PK}(e = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu}; \rho) \wedge e' = \mathsf{Enc}_{\mathsf{pkm}}(g^{\mu}; \rho') \wedge \mu \in [0, \mathsf{CC}])$.

1. Prover has inputs $(descr(\mathbb{G}), g, \mathsf{CC}, \mathsf{pkt}, \mathsf{pkm}, e')$. He computes $\mathfrak{i}$ as in Prot. 1, but then he sets $\mathfrak{c} \leftarrow H(descr(\mathbb{G}), g, \mathsf{CC}, \mathsf{pkt}, \mathsf{pkm}, e', \mathfrak{i})$, and computes $\mathfrak{r}$ that corresponds to this value of $\mathfrak{c}$. The NIZK proof of knowledge is equal to $\pi \leftarrow (e_0, \ldots, e_\lambda, \mathfrak{c}, \mathfrak{r})$.
2. Verifier has inputs $(descr(\mathbb{G}), g, \mathsf{CC}, \mathsf{pkt}, \mathsf{pkm}, e', \pi)$. On input π, she computes the missing elements of $\mathfrak{i}$ exactly as in the proof of the SHVZK property of Prot. 1. Verifier accepts if and only if $\mathfrak{c} = H(descr(\mathbb{G}), g, \mathsf{CC}, \mathsf{pkt}, \mathsf{pkm}, e', \mathfrak{i})$.

Protocol 2. NIZK proof of knowledge version of Prot. 1

NIZK Proof of Knowledge Version. Since Prot. 1 is correct, specially sound and SHVZK, we can now use the Fiat-Shamir heuristic to construct a secure NIZK proof of knowledge. This version is depicted by Prot. 2. Note that when Elgamal in the subgroups of $\mathbb{Z}_p$ is used then $descr(\mathbb{G}) = (p, q)$ and thus $\mathfrak{c} \leftarrow H(p, q, g, \ldots)$.

4 Cryptographic Protocol for E-Vote Integrity

The voting process consists of a number of voters $\mathcal{V}$, their PCs, one or more messengers (Messenger), one or more vote collectors (VC) and one or more talliers (Tallier). A voter enters her preferred candidate number—by using a user-friendly GUI—to her PC, that then runs a vote registration protocol with the vote collectors. Vote collectors collect the votes, and send their collection to the talliers after the voting period has finished. Within this paper, we are not going to specify most of the internal working of the vote collectors or the vote talliers since there exists already an extensive literature on that.

In this paper, we focus on the case when the voter's PC is dishonest. Clearly, if voters would only have access to their PCs, no security could be achieved at all. Therefore, in addition we need the presence of some independent channels accessible by the voters. As an example, in many countries, before any elections the voters will anyway receive a paper voter registration card. We can make use of this channel (prechannel), by adding extra information on this acknowledgment. In addition, most of the voters have access to more than one connected device. The second device (postchannel) may be something simple, like a mobile phone, even if it cannot perform any complex cryptographic operations, but can still guarantee real-time reception of messages.

Description of Protocol. Assume that we have $\mathsf{CC} + 1 > 0$ candidates, and every candidate has been assigned a number $\mathsf{cnd} \in \{0, \ldots, \mathsf{CC}\}$. Since CC is small, we are going to use the AIR-based proxy oblivious transfer protocol $(\mathsf{Gcpir}, \mathsf{Query}, \mathsf{Reply}, \mathsf{Answer})$ and the Elgamal cryptosystem $(\mathsf{Gen}^{\mathsf{pkc}}, \mathsf{Enc}, \mathsf{Dec})$. In particular since Elgamal is multiplicatively homomorphic, instead of the candidate cnd we encrypt g^{cnd}, where g is a fixed generator of Elgamal's plaintext group. (If an additively homomorphic cryptosystem were used, one could instead just encrypt cnd. However, such cryptosystems tend to be less efficient in practice.) The protocol is depicted by Prot. 3.

Complexity. Vote collector's computation is dominated by the verification of the NIZK proof of knowledge (which takes at most $2\lambda + 3$ encryptions and $2\lambda + 2$ exponentiations), and by the execution of the sender's part in the POT protocol that is dominated

System parameters: $\mathbb{G}, g, q, H$.
Voter's inputs: encryption keys of tallier, messenger, her own private signature key, voter collector's signature verification key.
Vote collector's inputs: encryption key of messenger, his own private signature key, voters' signature verification keys.
Tallier's inputs: his own private decryption key, vote collector's signature verification key.
Common inputs: $\mathsf{CC}+1$ candidates $c \in [0, \mathsf{CC}]$, $\lambda := \lfloor \log_2 \mathsf{CC} \rfloor$.

1. Before elections:
   (a) $(\mathbb{G}, q, g)$ and H are fixed and published by a trusted server.
   (b) Some server (be it vote collector or a separate server) generates for every voter-candidate pair (v, cnd) a uniformly random string $R_v[\mathsf{cnd}] \leftarrow \mathbb{Z}_q$, and sets $\mathsf{Code}_v[\mathsf{cnd}] \leftarrow \mathsf{Kdf}(g^{R_v[\mathsf{cnd}]})$ where Kdf is a key derivation function. It sends signed codes $\mathsf{Code}_v[\mathsf{cnd}]$ to corresponding voters (by using $\mathsf{prechannel}$) and to the messengers (in numerically sorted order), and signed values $R_v[\mathsf{cnd}]$ to the vote collectors. // In practice, only the first few, say 25 bits of $\mathsf{Code}_v[\mathsf{cnd}]$ are sent.
2. When voter v enters a candidate number cnd (by using favorite UI) to voter's PC:
   (a) Voter's PC does:
      i. He generates the first message $e' \leftarrow \mathsf{Query}_{\mathsf{pkm}}(\mathsf{cnd})$ of the new weak proxy oblivious transfer protocol.
      ii. He generates a non-interactive zero-knowledge proof $\pi = \mathrm{PK}(e = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu}; \rho) \wedge e' = \mathsf{Query}_{\mathsf{pkm}}(\mu; \rho') \wedge \mu \in [0, \mathsf{CC}])$ that both e and e' correspond to the same *valid* candidate (see Prot. 2).
      iii. He signs (e', π) by using his secret signing key sk_v, $s \leftarrow \mathsf{Sign}_{\mathsf{sk}_v}(e, e', \pi)$.
      iv. He then sends (e', π, s) to the vote collector. (Note that π contains the list $(e_0, \ldots, e_\lambda)$ with $e_j = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_j})$ and $\mu_j \in \{0, 1\}$.)
   (b) After receiving a ballot from the PC, the vote collector does:
      i. He verifies both the signature and the zero-knowledge proof (as specified in Prot. 2). If both verifications are fine, it computes the second message $r \leftarrow \mathsf{Reply}_{\mathsf{pkm}}(e', \mathsf{Code}_v)$ of the POT protocol. Recall here that r consists of a number of randomly-reordered ciphertexts.
      ii. He sends to the voter's PC a signed message accept or reject.
      iii. He signs r and sends it to the messenger.
   (c) After receiving a message from the VC, the messenger does:
      – She verifies the signature on r. She complains when it does not verify.
      – Otherwise, she "decrypts" $g^{R_v[\mathsf{cnd}]} \leftarrow \mathsf{Answer}_{\mathsf{skm}}(\mathsf{cnd}, \mathsf{Reply})$, where skm is messenger's secret key, and obtains $\mathsf{Code}_v[\mathsf{cnd}] \leftarrow \mathsf{Kdf}(g^{R_v[\mathsf{cnd}]})$. (The procedure for this is specified in Sect. 3.1.) It also alerts the voter by using $\mathsf{postchannel}$ with the value of $\mathsf{Code}_v[\mathsf{cnd}]$.
   (d) When receiving a message from $\mathsf{postchannel}$, the voter checks that $\mathsf{Code}_v[\mathsf{cnd}]$ is correct, as in Step 5 if the ideal-world vote registration protocol. The voter also checks that her legitimate voting acts are accompanied by a $\mathsf{postchannel}$ message, and that she receives no spurious messages.
3. After the election period has ended, the vote collector sends all values $e = \prod e_j^{\mathsf{CC}_j}$, signed with his own private key, to the tallier. The tallier operates by using a suitable e-voting procedure to deduce the winner.

Protocol 3. The new protocol between a voter, her computer, vote collector, and messenger

by $2(\mathsf{CC}+1)$ encryptions ($\mathsf{CC}+1$ of which can be precomputed) and $\mathsf{CC}+1$ exponentiations. On top of that, the vote collector has to verify a signature, and sign her message to the messenger. Given say $\mathsf{CC}+1 = 63$ candidates (then $\lambda = 5$), her computation is thus dominated by $2\lambda + 3 + 2(\mathsf{CC}+1) = 139$ encryptions and $2\lambda + 2 + \mathsf{CC} + 1 = 75$ exponentiations, some of which can be precomputed. Note that the bulk of vote collector's computation goes to computing her part of the POT protocol. This seems to be inevitable since most of the known oblivious transfer protocols (the only exception is [11]) requite linear computation. On the other hand, while the description of the NIZK proof of knowledge is seemingly more complex, it is considerably more efficient than the POT protocol.

Discussion. If $R_v[\mathsf{cnd}]$ is long (say $\geq$ 20 bits) then computing Answer requires the computation of discrete logarithm with time complexity of $\geq 2^{10}$ steps by using Pollard's ρ algorithm. Our solution to this is that instead of $R_v[\mathsf{cnd}]$, the check code is $\mathsf{Code}_v[\mathsf{cnd}] = \mathsf{Kdf}(g^{R_v[\mathsf{cnd}]})$. This means that the values $\mathsf{Code}_v[\mathsf{cnd}]$ will be sent over $\mathsf{prechannel}$, too. On the other hand, this step is done by client's computer only once in a while and thus is not a bottleneck, and it may even be desirable to prevent DDoS attacks, by forcing client's computer to perform some work per every cast vote. Also, note that the tallier obtains a ciphertext of g^{cnd}. Here, computing of discrete logarithm is again simple since cnd is small (it can be done by using table-lookup).

5 Security of Integrity Protocol

We now state the security of the e-voting process, given the new integrity protocol. We will give informal security arguments, leaving formal proofs for further work. In all following paragraphs, we consider the case when one party is dishonest, but all other parties are honest. This assumption is not necessary when one additionally implements protocols that guarantee security against malicious servers. For example, one can use standard mixnets, but as said, this is not the topic of the current paper. Note that all parties can blindly refuse accept votes, claiming to have troubles with connection, but this is unavoidable.

Security against Voter Computer. There are no privacy guarantees against malicious voter's PC. However, by doing proper checks, a voter can clearly verify that the voter's PC has voted for a wrong candidate, or did not vote at all. In the case the verification fails, voters can participate in later paper voting that overrides the results of the e-voting.

Security against Vote Collector. Vote collector only sees encrypted data, and thus here privacy is guaranteed. She cannot change votes (since they are signed).

Security against Messenger. Messenger only sees the codes, and which code the voter is voting for right now, but nothing else. Thus, privacy is covered except in the next sense: the messenger can test, in the case of a revote, whether this time the voter is voting for a new or an old candidate. The messenger can also not send a $\mathsf{postchannel}$ message based on such tests. The messenger can also send back a message that corresponds to an earlier vote by the same candidate, but this will be detected by the voter.

Security against Tallier. Tallier only obtains a list of all encrypted ballots, signed by the vote collector. The tallier cannot thus breach the privacy. To guarantee some robustness/integrity while tallying, one can use some well-known cryptographic protocols (for example, mixnets).

6 Discussion

While choosing the underlying primitives and protocols, we considered efficiency to be the most important factor, closely followed by the simplicity of implementation and standardness of security assumptions. Next we will try to motivate our choices.

Public-key Cryptosystem. While Elgamal is only multiplicatively homomorphic, it is several times more efficient than the known additively homomorphic cryptosystems, especially in decryption. In addition, NIZK proofs of knowledge based on known additively homomorphic cryptosystems tend to be less efficient. Slower encryption, decryption and NIZK verifications would make vote collector's computations much more costly. On the other hand, by using standard tricks, we were able to minimize the drawbacks of Elgamal public-key cryptosystem, i.e., the need to compute discrete logarithms. Moreover, Elgamal encryption (and in particular, Elgamal encryption based on elliptic curves) is implemented by several commercially available Hardware Security Modules, which cannot be said about the known additively homomorphic cryptosystems.

Voter Education. For the added two channels and the new protocol to be useful in practice, the voters must be educated. They must be told that they should never enter the check codes to their computer, and that they should actively react to the messages (or their absence) on the postchannel. This will add extra costs, but the costs will be hopefully amortized over several elections. Moreover, the Internet and computers are ubiquitous in the developed world already now, with average people performing much more complex operations in a daily basis. Thus, after some years we can reasonably expect the voters to know how to guarantee their own vote privacy (and security in general case).

7 Implementation Data

We implemented a (slightly optimized) sandbox version of the new e-voting protocol. We tested it thoroughly, and measured its efficiency by using a personal computer that runs `Linux 2.6.18-6-686`, has a Pentium 4 CPU that runs at 2.80GHz and has 512 KB of cache, and has 2 GB of main memory. The code was compiled by using `gcc 4.1.2` with the option `-O2`. For generating the Elgamal parameters, we used the `openssl 0.9.8c` library, while other number-theoretic operations were implemented by using Victor Shoup's `NTL 5.5.1` library.

We measured the time that was spent during the election setup, and during the election itself. In the tallying, one can use any of the standard mixnet-based solutions, and thus we did not measure this part. For the time measurement, we used the standard `Unix` command `time`, and took the average over 100 different runs. The results are summarized in the next two tables, for $v = \{100, 1000, 10\,000\}$ voters, and

$c \in \{8, 32, 80\}$ candidates. In all cases, $|p| = 1024$, $|q| = 160$, and $k = 80$. We used SHA2-256 as the hash function. The first table contains the one-time election setup cost (codecard generation and Elgamal system parameter value generation) which depends linearly on the product $v \cdot c$. More precisely, it is dominated by $v \cdot c$ random number generations and exponentiations modulo p.

	$v = 100$			$v = 1\,000$			$v = 10\,000$		
	$c = 8$	$c = 32$	$c = 80$	$c = 8$	$c = 32$	$c = 80$	$c = 8$	$c = 32$	$c = 80$
Setup	3.875s	15.40s	38.48s	38.58s	2m 34s	6m 25s	6m 25s	25m 38s	1h 4m 20s

The next table summarizes the online computation time of voter's PC, vote collector and messenger, both with and without the zero-knowledge proofs. The costs are given per one vote, and do not significantly depend on the number of the voters. The total row is the sum of the time spent by voter's PC, vote collector and messenger, and gives a (loose) lower bound on time that must elapse before the voter receives back a message on the postchannel.

	With ZK			Without ZK		
	$c = 8$	$c = 32$	$c = 80$	$c = 8$	$c = 32$	$c = 80$
Voter's PC	0.21s	0.30s	0.34s	0.02s	0.02s	0.02s
Vote collector	0.40s	1.07s	2.27s	0.20s	0.78s	1.95s
Messenger	0.02s	0.08s	0.22s	0.02s	0.08s	0.20s
Total	0.63s	1.45s	2.83s	0.24s	0.88s	2.17s

We also note that a single exponentiation on this machine took about 0.0048s. Moreover, the timings of the parties include also the precomputation time. In particular, vote collector's online computation in the POT protocol requires twice less time than her total computation in POT.

As seen from these tables, the computation time of the voter's PC and messenger is quite insignificant even in the case of 80 candidates. On the other hand, if there are 80 candidates, then the vote collector spends (on average) 2.27 seconds per vote and cannot process more than about 1 500 votes per hour even under ideal conditions. Assuming that the vote collector precomputes in the POT protocol, the throughput increases to 3 000 votes per hour. In the case of real e-voting, the cryptographic protocol is obviously only a part of what the vote collector is busy with, and thus the maximum throughput is probably around 2 000 votes per hour. In smaller countries, this is sufficient under normal conditions, but not during the first or the last few hours of the e-voting. However, this can be alleviated by using either fast (and multicore) processors, parallel processing by many vote collectors, or even by using hardware acceleration. (In particular, we are currently considering a Hardware Security Module implementation based on elliptic curves.) The use of such (more expensive) alternatives is reasonable, given the importance of elections in a democratic society. Moreover, in the case of most elections, the number of candidates is not larger than 10.

Acknowledgments. We would like to thank Kristian Gjøsteen for useful comments. The second author was supported by Estonian Science Foundation, grant #8058, and European Union through the European Regional Development Fund.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced Oblivious Transfer: How to Sell Digital Goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
2. Ansper, A., Heiberg, S., Lipmaa, H., Øverland, T.A., van Laenen, F.: Security and Trust for the Norwegian E-voting Pilot Project E-valg 2011. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec 2009. LNCS, vol. 5838, pp. 207–222. Springer, Heidelberg (2009)
3. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Ashby, V. (ed.) ACM CCS 1993, November 3-5, pp. 62–73. ACM Press, Fairfax (1993)
4. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient Protocols for Set Membership and Range Proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
5. Canetti, R., Goldreich, O., Halevi, S.: The Random Oracle Methodology, Revisited. In: STOC 1998, New York, May 23-26, pp. 209–218 (1998)
6. Chevassut, O., Fouque, P.A., Gaudry, P., Pointcheval, D.: The Twist-AUgmented Technique for Key Exchange. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 410–426. Springer, Heidelberg (2006)
7. Elgamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
8. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)
9. Heiberg, S., Lipmaa, H., Van Laenen, F.: On E-Vote Integrity in the Case of Malicious Voter Computers. Tech. Rep. 2010/195, International Association for Cryptologic Research, (April 8, 2010), http://eprint.iacr.org/2010/195
10. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 314–328. Springer, Heidelberg (2005)
11. Lipmaa, H.: First CPIR Protocol with Data-Dependent Computation. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 193–210. Springer, Heidelberg (2010)
12. Lipmaa, H., Asokan, N., Niemi, V.: Secure Vickrey Auctions without Threshold Trust. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 87–101. Springer, Heidelberg (2003)
13. Naor, M., Pinkas, B., Sumner, R.: Privacy Preserving Auctions and Mechanism Design. In: ACM EC 1999, Denver, Colorado (November 1999)

Election Verifiability in Electronic Voting Protocols*,**

Steve Kremer[1], Mark Ryan[2], and Ben Smyth[2,3]

[1] LSV, ENS Cachan & CNRS & INRIA, France
[2] School of Computer Science, University of Birmingham, UK
[3] École Normale Supérieure & CNRS & INRIA, France

Abstract. We present a formal, symbolic definition of election verifiability for electronic voting protocols in the context of the applied pi calculus. Our definition is given in terms of boolean tests which can be performed on the data produced by an election. The definition distinguishes three aspects of verifiability: individual, universal and eligibility verifiability. It also allows us to determine precisely which aspects of the system's hardware and software must be trusted for the purpose of election verifiability. In contrast with earlier work our definition is compatible with a large class of electronic voting schemes, including those based on blind signatures, homomorphic encryption and mixnets. We demonstrate the applicability of our formalism by analysing three protocols: FOO, Helios 2.0, and Civitas (the latter two have been deployed).

1 Introduction

Electronic voting systems are being introduced, or trialled, in several countries to provide more efficient voting procedures. However, the security of electronic elections has been seriously questioned [9,20,8,25]. A major difference with traditional paper based elections is the lack of transparency. In paper elections it is often possible to observe the whole process from ballot casting to tallying, and to rely on robustness characteristics of the physical world (such as the impossibility of altering the markings on a paper ballot sealed inside a locked ballot box). By comparison, it is not possible to observe the electronic operations performed on data. Computer systems may alter voting records in a way that cannot be detected by either voters or election observers. A voting terminal's software might be infected by malware which could change the entered vote, or even execute a completely different protocol than the one expected. The situation can be described as *voting on Satan's computer*, analogously with [5].

The concept of *election* or *end-to-end verifiability* that has emerged in the academic literature, *e.g.*, [17,18,10,3,21,2], aims to address this problem. It should allow voters and election observers to verify, independently of the hardware and software running the election, that votes have been recorded, tallied and declared correctly. One generally distinguishes two aspects of verifiability.

* This work has been partly supported by the EPSRC projects *UbiVal* (EP/D076625/2), *Trustworthy Voting Systems* (EP/G02684X/1) and *Verifying Interoperability Requirements in Pervasive Systems* (EP/F033540/1); the *ANR SeSur AVOTÉ* project; and the *Direction Générale pour l'Armement* (DGA).

** A long version containing full proofs is available in [19].

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 389–404, 2010.

- *Individual verifiability:* a voter can check that her own ballot is included in the election's bulletin board.
- *Universal verifiability:* anyone can check that the election outcome corresponds to the ballots published on the bulletin board.

We identify another aspect that is sometimes included in universal verifiability.

- *Eligibility verifiability:* anyone can check that each vote in the election outcome was cast by a registered voter and there is at most one vote per voter.

We explicitly distinguish eligibility verifiability as a distinct property.

Our contribution. We present a definition of election verifiability which captures the three desirable aspects. We model voting protocols in the applied pi calculus and formalise verifiability as a triple of boolean tests $\Phi^{IV}, \Phi^{UV}, \Phi^{EV}$ which are required to satisfy several conditions on all possible executions of the protocol. Φ^{IV} is intended to be checked by the individual voter who instantiates the test with her private information (*e.g.*, her vote and data derived during the execution of the protocol) and the public information available on the bulletin board. Φ^{UV} and Φ^{EV} can be checked by any external observer and only rely on public information, *i.e.*, the contents of the bulletin board.

The consideration of eligibility verifiability is particularly interesting as it provides an assurance that the election outcome corresponds to votes legitimately cast and hence provides a mechanism to detect ballot stuffing. We note that this property has been largely neglected in previous work and our earlier work [23] only provided limited scope for.

A further interesting aspect of our work is the clear identification of which parts of the voting system need to be trusted to achieve verifiability. As it is not reasonable to assume voting systems behave correctly we only model the parts of the protocol that we need to trust for the purpose of verifiability; all the remaining parts of the system will be controlled by the adversarial environment. Ideally, such a process would only model the interaction between a *voter* and the voting terminal; *that is, the messages input by the voter*. In particular, the voter should not need to trust the election hardware or software. However, achieving absolute verifiability in this context is difficult and protocols often need to trust some parts of the voting software or some administrators. Such trust assumptions are motivated by the fact that parts of a protocol can be audited, or can be executed in a distributed manner amongst several different election officials. For instance, in Helios 2.0 [3], the ballot construction can be audited using a cast-or-audit mechanism. Whether trust assumptions are reasonable depends on the context of the given election, but our work makes them explicit.

Tests Φ^{IV}, Φ^{UV} and Φ^{EV} are assumed to be verified in a trusted environment (if a test is checked by malicious software that always evaluates the test to hold, it is useless). However, the verification of these tests, unlike the election, can be repeated on different machines, using different software, provided by different stakeholders of the election. Another possibility to avoid this issue would be to have tests which are human-verifiable as discussed in [2, Chapter 5].

We apply our definition on three case studies: the protocol by Fujioka *et al.* [15]; the Helios 2.0 protocol [4] which was effectively used in recent university elections in Belgium; and the protocol by Juels *et al.* [18], which has been implemented by Clarkson *et al.* as Civitas [13,12]. This demonstrates that our definition is suitable for a large class of protocols; including schemes based on mixnets, homomorphic encryption and blind signatures. (In contrast, our preliminary work presented in [23] only considers blind signature schemes.) We also notice that Helios 2.0 does not guarantee eligibility verifiability and is vulnerable to ballot stuffing by dishonest administrators.

Related work. Juels *et al.* [17,18] present a definition of universal verifiability in the provable security model. Their definition assumes voting protocols produce non-interactive zero-knowledge proofs demonstrating the correctness of tallying. Here we consider definitions in a symbolic model. Universal verifiability was also studied by Chevallier-Mames *et al.* [11]. They show an incompatibility result: protocols cannot satisfy verifiability and vote privacy in an unconditional way (without relying on computational assumptions). But as witnessed by [17,18], weaker versions of these properties can hold simultaneously. Our case studies demonstrate that our definition allows privacy and verifiability to coexist (see [14,6] for a study of privacy properties in the applied pi calculus). Baskar *et al.* [7] and subsequently Talbi *et al.* [24] formalised individual and universal verifiability for the protocol by Fujioka *et al.* [15]. Their definitions are tightly coupled to that particular protocol and cannot easily be generalised. Moreover, their definitions characterise individual executions as verifiable or not; such properties should be considered with respect to every execution.

In our earlier work [23] a preliminary definition of election verifiability was presented with support for automated reasoning. However, that definition is too strong to hold on protocols such as [18,4]. In particular, our earlier definition was only illustrated on a simplified version of [18] which did not satisfy coercion-resistance because we omitted the mixnets. Hence, this is the first general, symbolic definition which can be used to show verifiability for many important protocols, such as the ones studied in this paper.

2 Applied pi Calculus

The calculus assumes an infinite set of names $a, b, c, k, m, n, s, t, \ldots$, an infinite set of variables $v, x, y, z, \ldots$ and a finite signature Σ, that is, a finite set of function symbols each with an associated arity. We also assume an infinite set of *record variables* $r, r_1, \ldots$. A function symbol of arity 0 is a constant. We use metavariables u, w to range over both names and variables. Terms L, M, N, T, U, V are built by applying function symbols to names, variables and other terms. Tuples $u_1, \ldots, u_l$ and $M_1, \ldots, M_l$ are occasionally abbreviated $\tilde{u}$ and $\tilde{M}$. We write $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ for substitutions that replace variables $x_1, \ldots, x_l$ with terms $M_1, \ldots, M_l$.

The applied pi calculus [1,22] relies on a simple sort system. Terms can be of sort Channel for channel names or Base for the payload sent out on these channels. Function symbols can only be applied to, and return, terms of sort Base. A term is ground when it does not contain variables.

The grammar for processes is shown in Figure 1 where u is either a name or variable of channel sort. Plain processes are standard constructs, except for the *record message* $\mathsf{rec}(r, M).P$ construct discussed below. Extended processes introduce *active substitutions* which generalise the classical let construct: the process $\nu\, x.(\{^M/_x\} \mid P)$ corresponds exactly to the process let $x = M$ in P. As usual names and variables have scopes which are delimited by restrictions and by inputs. All substitutions are assumed to be cycle-free.

$P, Q, R ::=$	processes	$A, B, C ::=$	extended processes
0	null process	P	plain process
$P \mid Q$	parallel	$A \mid B$	parallel composition
$!P$	replication	$\nu\, n.A$	name restriction
$\nu\, n.P$	name restriction	$\nu\, x.A$	variable restriction
$u(x).P$	message input	$\{^M/_x\}$	active substitution
$\overline{u}\langle M\rangle.P$	message output		
$\mathsf{rec}(r, M).P$	record message		
if $M = N$ then P else Q	conditional		

Fig. 1. Applied pi calculus grammar

A *frame* φ is an extended process built from 0 and active substitutions $\{^M/_x\}$; which are composed by parallel composition and restriction. The *domain* of a frame φ is the set of variables that φ exports. Every extended process A can be mapped to a frame $\phi(A)$ by replacing every plain process in A with 0.

The record message construct $\mathsf{rec}(r, M).P$ introduces the possibility to enter special entries in frames. We suppose that the sort system ensures that r is a variable of record sort, which may only be used as a first argument of the rec construct or in the domain of the frame. Moreover, we make the global assumption that a record variable has a unique occurrence in each process. Intuitively, this construct will be used to allow a voter to privately record some information which she may later use to verify the election.

The sets of free and bound names and variables in process A are denoted by $\mathrm{fn}(A)$, $\mathrm{bn}(A)$, $\mathrm{fv}(A)$, $\mathrm{bv}(A)$. Similarly, we write $\mathrm{fn}(M)$, $\mathrm{fv}(M)$ for the names and variables in term M and $\mathrm{rv}(A)$ and $\mathrm{rv}(M)$ for the set of record variables in a process and a term. An extended process A is *closed* if $\mathrm{fv}(A) = \emptyset$. A *context* $C[\_]$ is an extended process with a hole. An *evaluation context* is a context whose hole is not under a replication, a conditional, an input, or an output.

The signature Σ is equipped with an equational theory E, that is, a finite set of equations of the form $M = N$. We define $=_E$ as the smallest equivalence relation on terms, that contains E and is closed under application of function symbols, substitution of terms for variables and bijective renaming of names. In this paper we tacitly assume that all signatures and equational theories contain the function symbols $\mathsf{pair}(\cdot,\cdot), \mathsf{fst}(\cdot), \mathsf{snd}(\cdot)$ and equations for pairing:

$$\mathsf{fst}(\mathsf{pair}(x, y)) = x \qquad \mathsf{snd}(\mathsf{pair}(x, y)) = y$$

as well as some constant $\bot$. As a convenient shortcut we then write $(\!|T_1, \ldots T_n|\!)$ for $\mathsf{pair}(T_1, \mathsf{pair}(\ldots, \mathsf{pair}(T_n, \bot)))$ and $\pi_i(T)$ for $\mathsf{fst}(\mathsf{snd}^{i-1}(T))$.

Semantics. The operational semantics of the applied pi calculus are defined with respect to the three relations: structural equivalence ($\equiv$), internal reductions ($\rightarrow$) and labelled reduction ($\xrightarrow{\alpha}$). These semantics are standard and defined in [19]. We only illustrate them on an example (Figure 2). We write $\Longrightarrow$ for $(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*$, that is, the reflexive transitive closure of the labelled reduction.

Let $P = \nu a, b.\mathsf{rec}(r, a).\overline{c}\langle(\!|a, b|\!)\rangle.c(x).\text{if } x = a \text{ then } \overline{c}\langle f(a)\rangle$. Then we have that

$$
\begin{array}{rcl}
P & \rightarrow & \nu a, b.(\overline{c}\langle(\!|a, b|\!)\rangle.c(x).\text{if } x = a \text{ then } \overline{c}\langle f(a)\rangle \mid \{^{a}/_{r}\}) \\
& \equiv & \nu a, b.(\nu y_1.(\overline{c}\langle y\rangle.c(x).\text{if } x = a \text{ then } \overline{c}\langle f(a)\rangle \mid \{^{(\!|a,b|\!)}/_{y_1}\}) \mid \{^{a}/_{r}\}) \\
& \xrightarrow{\nu x.\overline{c}\langle x\rangle} & \nu a, b.(c(x).\text{if } x = a \text{ then } \overline{c}\langle f(a)\rangle \mid \{^{(\!|a,b|\!)}/_{y_1}\} \mid \{^{a}/_{r}\}) \\
& \xrightarrow{\nu x.c(\pi_1(y))} & \nu a, b.(\text{if } a = a \text{ then } \overline{c}\langle f(a)\rangle \mid \{^{(\!|a,b|\!)}/_{y_1}\} \mid \{^{a}/_{r}\}) \\
& \rightarrow & \nu a, b.(\overline{c}\langle f(a)\rangle \mid \{|\, \{^{(\!|a,b|\!)}/_{y_1}\} \mid \{^{a}/_{r}\}) \\
& \xrightarrow{\nu y_2.\overline{c}\langle y_2\rangle} & \nu a, b.(\text{if } a = a \text{ then } \overline{c}\langle f(a)\rangle \mid \{^{(\!|a,b|\!)}/_{y_1}\} \mid \{^{f(a)}/_{y_2}\} \mid \{^{a}/_{r}\}
\end{array}
$$

Observe that labelled outputs are done by reference and extend the domain of the process's frame.

Fig. 2. A sequence of reductions in the applied pi semantics

3 Formalising Voting Protocols

As discussed in the introduction we want to explicitly specify the parts of the election protocol which need to be trusted. Formally the trusted parts of the voting protocol can be captured using a voting process specification.

Definition 1 (Voting process specification). *A* voting process specification *is a tuple* $\langle V, A\rangle$ *where* V *is a plain process without replication and* A *is a closed evaluation context such that* $fv(V) = \{v\}$ *and* $rv(V) = \emptyset$.

For the purposes of individual verifiability the voter may rely on some data derived during the protocol execution. We keep track of all such values using the record construct (Definition 2).

Definition 2. *Let* rv *be an infinite list of distinct* record variables. *We define the function* R *on a finite process* P *without replication as* $\mathsf{R}(P) = \mathsf{R}_{\mathsf{rv}}(P)$ *and, for all lists* rv*:*

$$
\begin{array}{ll}
\mathsf{R}_{rv}(0) & \widehat{=}\ 0 \\
\mathsf{R}_{rv}(P \mid Q) & \widehat{=}\ \mathsf{R}_{odd(rv)}(P) \mid \mathsf{R}_{even(rv)}(Q) \\
\mathsf{R}_{rv}(\nu\, n.P) & \widehat{=}\ \nu\, n.\mathsf{rec}(head(rv), n).\mathsf{R}_{tail(rv)}(P) \\
\mathsf{R}_{rv}(u(x).P) & \widehat{=}\ u(x).\mathsf{rec}(head(rv), x).\mathsf{R}_{tail(rv)}(P) \\
\mathsf{R}_{rv}(\overline{u}\langle M\rangle.P) & \widehat{=}\ \overline{u}\langle M\rangle.\mathsf{R}_{rv}(P) \\
\mathsf{R}_{rv}(\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q) & \widehat{=}\ \mathit{if}\ M = N\ \mathit{then}\ \mathsf{R}_{rv}(P)\ \mathit{else}\ \mathsf{R}_{rv}(Q)
\end{array}
$$

where the functions head and tail are the usual ones for lists, and odd (resp. even) returns the list of elements in odd (resp. even) position.

In the above definition odd and even are used as a convenient way to split an infinite list into two infinite lists.

Given a sequence of record variables $\tilde{r}$, we denote by $\tilde{r}_i$ the sequence of variables obtained by indexing each variable in $\tilde{r}$ with i. A voting process can now be constructed such that the voter V records the values constructed and input during execution.

Definition 3. *Given a voting process specification* $\langle V, A\rangle$*, integer* $n \in \mathbb{N}$*, and names* $s_1, \ldots, s_n$*, we build the augmented voting process* $\mathsf{VP}_n^+(s_1, \ldots, s_n) = A[V_1^+ \mid \cdots \mid V_n^+]$ *where* $V_i^+ = \mathsf{R}(V)\{^{s_i}/_v\}\{^{r_i}/_r \mid r \in rv(\mathsf{R}(V))\}$.

The process $\mathsf{VP}_n^+(s_1, \ldots, s_n)$ models the voting protocol for n voters casting votes $s_1, \ldots, s_n$, who privately record the data that may be needed for verification using record variables $\tilde{r}_i$.

4 Election Verifiability

We formalise election verifiability using three tests Φ^{IV}, Φ^{UV}, Φ^{EV}. Formally, a test is built from conjunctions and disjunctions of *atomic tests* of the form $(M =_E N)$ where M, N are terms. Tests may contain variables and will need to hold on frames arising from arbitrary protocol executions. We now recall the purpose of each test and assume some naming conventions about variables.

Individual verifiability: The test Φ^{IV} allows a voter to identify her ballot in the bulletin board. The test has:

- a variable v referring to a voter's vote.
- a variable w referring to a voter's public credential.
- some variables $x, \bar{x}, \hat{x}, \ldots$ expected to refer to global public values pertaining to the election, *e.g.*, public keys belonging to election administrators.
- a variable y expected to refer to the voter's ballot on the bulletin board.
- some record variables $r_1, \ldots, r_k$ referring to the voter's private data.

Universal verifiability: The test Φ^{UV} allows an observer to check that the election outcome corresponds to the ballots in the bulletin board. The test has:

- a tuple of variables $\tilde{v} = (v_1, \ldots, v_n)$ referring to the declared outcome.
- some variables $x, \bar{x}, \hat{x}, \ldots$ as above.
- a tuple $\tilde{y} = (y_1, \ldots, y_n)$ expected to refer to all the voters' ballots on the bulletin board.
- some variables $z, \bar{z}, \hat{z}, \ldots$ expected to refer to outputs generated during the protocol used for the purposes of universal and eligibility verification.

Eligibility verifiability: The test Φ^{EV} allows an observer to check that each ballot in the bulletin board was cast by a unique registered voter. The test has:

- a tuple $\tilde{w} = (w_1, \ldots, w_n)$ referring to public credentials of eligible voters.
- a tuple $\tilde{y}$, variables $x, \bar{x}, \hat{x}, \ldots$ and variables $z, \bar{z}, \hat{z}, \ldots$ as above.

The remainder of this section will focus on the individual and universal aspects of our definition; eligibility verifiability will be discussed in Section 5.

4.1 Individual and Universal Verifiability

The tests suitable for the purposes of election verifiability have to satisfy certain conditions: if the tests succeed, then the data output by the election is indeed valid (*soundness*); and there is a behaviour of the election authority which produces election data satisfying the tests (*effectiveness*). Formally these requirements are captured by the definition below. We write $\tilde{T} \simeq \tilde{T}'$ to denote that the tuples $\tilde{T}$ and $\tilde{T}'$ are a permutation of each other modulo the equational theory, that is, we have $\tilde{T} = T_1, \ldots T_n$, $\tilde{T}' = T'_1, \ldots T'_n$ and there exists a permutation χ on $\{1, \ldots, n\}$ such that for all $1 \leq i \leq n$ we have $T_i =_E T'_{\chi(i)}$.

Definition 4 (Individual and universal verifiability). *A voting specification* $\langle V, A \rangle$ *satisfies* individual and universal verifiability *if for all* $n \in \mathbb{N}$ *there exist tests* Φ^{IV}, Φ^{UV} *such that* $fn(\Phi^{IV}) = fn(\Phi^{UV}) = rv(\Phi^{UV}) = \emptyset$, $rv(\Phi^{IV}) \subseteq rv(\mathsf{R}(V))$, *and for all names* $\tilde{s} = (s_1, \ldots, s_n)$ *the conditions below hold. Let* $\tilde{r} = rv(\Phi^{IV})$ *and* $\Phi^{IV}_i = \Phi^{IV}\{^{s_i}/_v, ^{\tilde{r}_i}/_{\tilde{r}}\}$.

Soundness. For all contexts C *and processes* B *such that* $C[\mathsf{VP}^+_n(s_1, \ldots, s_n)] \Longrightarrow B$ *and* $\phi(B) \equiv \nu\tilde{n}.\sigma$, *we have:*

$$\forall i, j. \quad \Phi^{IV}_i \sigma \wedge \Phi^{IV}_j \sigma \Rightarrow i = j \tag{1}$$

$$\Phi^{UV} \sigma \wedge \Phi^{UV}\{^{\tilde{v}'}/_{\tilde{v}}\}\sigma \Rightarrow \tilde{v}\sigma \simeq \tilde{v}'\sigma \tag{2}$$

$$\bigwedge_{1 \leq i \leq n} \Phi^{IV}_i\{^{y_i}/_y\}\sigma \wedge \Phi^{UV}\sigma \;\Rightarrow\; \tilde{s} \simeq \tilde{v}\sigma \tag{3}$$

Effectiveness. There exists a context C *and a process* B, *such that* $C[\mathsf{VP}^+_n(s_1, \ldots, s_n)] \Longrightarrow B$, $\phi(B) \equiv \nu\tilde{n}.\sigma$ *and*

$$\bigwedge_{1 \leq i \leq n} \Phi^{IV}_i\{^{y_i}/_y\}\sigma \wedge \Phi^{UV}\sigma \tag{4}$$

An individual voter should verify that the test Φ^{IV} holds when instantiated with her vote s_i, the information $\tilde{r}_i$ recorded during the execution of the protocol and some bulletin board entry. Indeed, Condition (1) ensures that the test will hold for at most one bulletin board entry. (Note that Φ^{IV}_i and Φ^{IV}_j are evaluated with the same ballot $y\sigma$ provided by $C[\_]$.) The fact that her ballot is counted will be ensured by Φ^{UV} which should also be tested by the voter. An observer will instantiate the test Φ^{UV} with the bulletin board entries $\tilde{y}$ and the declared outcome $\tilde{v}$. Condition (2) ensures the observer that Φ^{UV} only holds for a single outcome. Condition (3) ensures that if a bulletin board contains the ballots of voters who voted $s_1, \ldots, s_n$ then Φ^{UV} only holds if the declared outcome is (a permutation of) these votes. Finally, Condition (4) ensures that there exists an execution where the tests hold. In particular this allows us to verify whether the protocol can satisfy the tests when executed as expected. This also avoids tests which are always false and would make Conditions (1)-(3) vacuously hold.

4.2 Case Study: FOO

The FOO protocol, by Fujioka, Okamoto & Ohta [15], is an early scheme based on blind signatures and has been influential for the design of later protocols.

How FOO works. The voter first computes her ballot as a commitment to her vote $m' = \mathsf{commit}(rnd, v)$ and sends the signed blinded ballot $\mathsf{sign}(sk_V, \mathsf{blind}(rnd', m'))$ to the registrar. The registrar checks that the signature belongs to an eligible voter and returns $\mathsf{sign}(sk_R, \mathsf{blind}(rnd', m'))$, the blind signed ballot. The voter verifies the registrar's signature and unblinds the message to recover her ballot signed by the registrar $m = \mathsf{sign}(sk_R, m')$. The voter then posts her signed ballot to the bulletin board. Once all votes have been cast the tallier verifies all the entries and appends an identifier ℓ to each valid entry. The voter then checks the bulletin board for her entry, the triple (ℓ, m', m), and appends the commitment factor rnd. Using rnd the tallier opens all of the ballots and announces the declared outcome.

Equational theory. We model blind signatures and commitment as follows.

$$\begin{aligned} \mathsf{checksign}(\mathsf{pk}(x), \mathsf{sign}(x, y)) &= \mathsf{true} & \mathsf{getmsg}(\mathsf{sign}(x, y)) &= y \\ \mathsf{unblind}(y, \mathsf{sign}(x, \mathsf{blind}(y, z))) &= \mathsf{sign}(x, z) & \mathsf{unblind}(x, \mathsf{blind}(x, y)) &= y \\ \mathsf{open}(x, \mathsf{commit}(x, y)) &= y \end{aligned}$$

Model in applied pi. The parts of the protocol that need to be trusted for achieving verifiability are surprisingly simple (Definition 5). The name rnd models the randomness that is supposed to be used to compute the commitment of the vote. All a voter needs to ensure is that the randomness used for the commitment is fresh. To ensure verifiability, all other operations such as computing the commitment, blinding and signing can be performed by the untrusted terminal.

Definition 5. *The voting process specification* $\langle V_{\mathsf{foo}}, A_{\mathsf{foo}} \rangle$ *is defined as*

$$V_{\mathsf{foo}} \mathrel{\hat{=}} \nu rnd.\overline{c}\langle v \rangle.\overline{c}\langle rnd \rangle \quad \textit{and} \quad A_{\mathsf{foo}}[\_] \mathrel{\hat{=}} \_.$$

Individual and universal verifiability. We define the tests

$$\Phi^{IV} \mathrel{\hat{=}} y =_E (r, \mathsf{commit}(r, v)) \qquad \Phi^{UV} \mathrel{\hat{=}} \bigwedge_{1 \le i \le n} v_i =_E \mathsf{open}(\pi_1(y), \pi_2(y))$$

Intuitively, a bulletin board entry y should correspond to the pair formed of the random generated by voter i and commitment to her vote.

Theorem 1. $\langle V_{\mathsf{foo}}, A_{\mathsf{foo}} \rangle$ *satisfies individual and universal verifiability.*

4.3 Case Study: Helios 2.0

Helios 2.0 [4] is an open-source web-based election system, based on homomorphic tallying of encrypted votes. It allows the secret election key to be distributed amongst several trustees, and supports distributed decryption of the election result. It also allows independent verification by voters and observers of election results. Helios 2.0 was successfully used in March 2009 to elect the president of the Catholic University of Louvain, an election that had 25,000 eligible voters.

How Helios works. An election is created by naming a set of trustees and running a protocol that provides each of them with a share of the secret part of a public key pair. The public part of the key is published. Each of the eligible voters is also provided with a private pseudo-identity. The steps that participants take during a run of Helios are as follows.

1. To cast a vote, the user runs a browser script that inputs her vote and creates a ballot that is encrypted with the public key of the election. The ballot includes a ZKP that the ballot represents an allowed vote (this is needed because the ballots are never decrypted individually).
2. The user can audit the ballot to check if it really represents a vote for her chosen candidate; if she elects to do this, the script provides her with the random data used in the ballot creation. She can then independently verify that the ballot was correctly constructed, but the ballot is now invalid and she has to create another one.
3. When the voter has decided to cast her ballot, the voter's browser submits it along with her pseudo-identity to the server. The server checks the ZKPs of the ballots, and publishes them on a bulletin board.
4. Individual voters can check that their ballots appear on the bulletin board. Any observer can check that the ballots that appear on the bulletin board represent allowed votes, by checking the ZKPs.
5. The server homomorphically combines the ballots, and publishes the encrypted tally. Anyone can check that this tally is done correctly.
6. The server submits the encrypted tally to each of the trustees, and obtains their share of the decryption key for that particular ciphertext, together with a proof that the key share is well-formed. The server publishes these key shares along with the proofs. Anyone can check the proofs.
7. The server decrypts the tally and publishes the result. Anyone can check this decryption.

Equational theory. We use a signature in which $\mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{text}})$ denotes the encryption with key x_{pk} and random x_{rand} of the plaintext x_{text}, and $x_{\mathsf{ciph}} * y_{\mathsf{ciph}}$ denotes the homomorphic combination of ciphertexts x_{ciph} and y_{ciph} (the corresponding operation on plaintexts is written $+$ and on randoms $\circ$). The term $\mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s, x_{\mathsf{ballot}})$ represents a proof that the ballot x_{ballot} contains some name s and random x_{rand} with respect to key x_{pk}; $\mathsf{decKey}(x_{\mathsf{sk}}, x_{\mathsf{ciph}})$ is a decryption key for x_{ciph} w.r.t. public key $\mathsf{pk}(x_{\mathsf{sk}})$; and $\mathsf{decKeyPf}(x_{\mathsf{sk}}, x_{\mathsf{ciph}}, x_{\mathsf{dk}})$ is a proof that x_{dk} is a decryption key for x_{ciph} w.r.t. public key $\mathsf{pk}(x_{\mathsf{sk}})$. We use the equational theory that asserts that $+, *, \circ$ are commutative and associative, and includes the equations:

$$\mathsf{dec}(x_{\mathsf{sk}}, \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{text}})) = x_{\mathsf{text}}$$

$$\mathsf{dec}(\mathsf{decKey}(x_{\mathsf{sk}}, ciph), ciph) = x_{\mathsf{plain}} \quad \text{where } ciph = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})$$

$$\mathsf{penc}(x_{\mathsf{pk}}, y_{\mathsf{rand}}, y_{\mathsf{text}}) * \mathsf{penc}(x_{\mathsf{pk}}, z_{\mathsf{rand}}, z_{\mathsf{text}}) = \mathsf{penc}(x_{\mathsf{pk}}, y_{\mathsf{rand}} \circ z_{\mathsf{rand}}, y_{\mathsf{text}} + z_{\mathsf{text}})$$

$$\mathsf{checkBallotPf}(x_{\mathsf{pk}}, ballot, \mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s, ballot)) = \mathsf{true} \quad \text{where } ballot = \mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s)$$

$$\mathsf{checkDecKeyPf}(\mathsf{pk}(x_{\mathsf{sk}}), ciph, dk, \mathsf{decKeyPf}(x_{\mathsf{sk}}, ciph, dk)) = \mathsf{true}$$
$$\text{where } ciph = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}}) \text{and } dk = \mathsf{decKey}(x_{\mathsf{sk}}, ciph)$$

Note that in the equation for $\mathsf{checkBallotPf}$ s is a name and not a variable. As the equational theory is closed under bijective renaming of names this equation holds for any name, but fails if one replaces the name by a term, *e.g.*, $s + s$. We suppose that all names are possible votes but give the possibility to check that a voter does not include a term $s + s$ which would add a vote to the outcome.

Model in applied pi. The parts of the system that are not verifiable are:

- The script that constructs the ballot. Although the voter cannot verify it, the trust in this script is motivated by the fact that she is able to audit it.
- The trustees. Although the trustees' behaviour cannot be verified, voters and observers may want to trust them because trust is distributed among them.

Hence, we include these two components in the context A_{helios} of our voting process specification.

Definition 6. *The voting process specification* $\langle V_{\mathsf{helios}}, A_{\mathsf{helios}}\rangle$ *is defined where*

$$\begin{aligned}
V_{\mathsf{helios}} &\hat{=} d(x_{\mathsf{pid}}).\ \overline{d}\langle v\rangle.\ d(x_{\mathsf{ballot}}).\ d(x_{\mathsf{ballotpf}}).\overline{c}\langle(\!|w, x_{\mathsf{ballot}}, x_{\mathsf{ballotpf}}|\!)\rangle\\
A_{\mathsf{helios}}[_] &\hat{=} \nu sk, d.\ \left(\overline{c}\langle\mathsf{pk}(sk)\rangle \mid (!\nu pid.\ \overline{d}\langle pid\rangle) \mid (!B) \mid T \mid _\right)\\
B &\hat{=} \nu m.\ d(x_{\mathsf{vote}}).\overline{d}\langle\mathsf{penc}(\mathsf{pk}(sk), m, x_{\mathsf{vote}})\rangle.\\
&\qquad \overline{d}\langle\mathsf{ballotPf}(\mathsf{pk}(sk), m, x_{\mathsf{vote}}, \mathsf{penc}(\mathsf{pk}(sk), m, x_{\mathsf{vote}}))\rangle\\
T &\hat{=} c(x_{\mathsf{tally}}).\ \overline{c}\langle(\!|\mathsf{decKey}(sk, x_{\mathsf{tally}}), \mathsf{decKeyPf}(sk, x_{\mathsf{tally}}, \mathsf{decKey}(sk, x_{\mathsf{tally}}))|\!)\rangle
\end{aligned}$$

We suppose that the inputs of x_{pid}, x_{ballot} and x_{ballotpf} are stored in record variables r_{pid}, r_{ballot} and $r_{ballotpf}$ respectively. The voter V_{helios} receives her voter id pid on a private channel. She sends her vote on the channel to A_{helios}, which creates the ballot for her. She receives the ballot and sends it (paired with pid) to the server. A_{helios} represents the parts of the system that are required to be trusted. It publishes the election key and issues voter ids. It includes the ballot creation script B, which receives a voter's vote, creates a random m and forms the ballot, along with its proof, and returns it to the voter. A_{helios} also contains the trustee T, which accepts a tally ciphertext and returns a decryption key for it, along with the proof that the decryption key is correct. We assume the trustee will decrypt any ciphertext (but only one).

The untrusted server is assumed to publish the election data. We expect the frame to define the election public key as x_{pk} and the individual pid's and ballots as y_i for each voter i. It also contains the homomorphic tally z_{tally} of the encrypted ballots, and the decryption key z_{decKey} and its proof of correctness z_{decKeyPf} obtained from the trustees. When the protocol is executed as expected the resulting frame should have substitution σ such that

$$\begin{aligned}
y_i\sigma &= (\!|pid_i, \mathsf{penc}(\mathsf{pk}(sk), m_i, v_i), \mathsf{ballotPf}(\mathsf{pk}(sk), m_i, v_i, \mathsf{penc}(\mathsf{pk}(sk), m_i, v_i))|\!)\\
x_{pk}\sigma &= \mathsf{pk}(sk) \qquad z_{\mathsf{tally}}\sigma = \pi_2(y_1) * \cdots * \pi_2(y_n)\sigma\\
z_{\mathsf{decKey}}\sigma &= \mathsf{decKey}(sk, z_{\mathsf{tally}})\sigma \quad z_{\mathsf{decKeyPf}}\sigma = \mathsf{decKeyPf}(sk, z_{\mathsf{tally}}, z_{\mathsf{decKey}})\sigma
\end{aligned}$$

Individual and universal verifiability. The tests Φ^{IV} and Φ^{UV} are introduced for verifiability purposes. Accordingly, given $n \in \mathbb{N}$ we define:

$$\begin{array}{l}\Phi^{IV} \mathrel{\widehat{=}} y =_E (r_{pid}, r_{ballot}, r_{ballotpf}) \\ \Phi^{UV} \mathrel{\widehat{=}} z_{\mathsf{tally}} =_E \pi_2(y_1) * \cdots * \pi_2(y_n) \\ \qquad \wedge \bigwedge_{i=1}^{n} (\mathsf{checkBallotPf}(x_{\mathsf{pk}}, \pi_2(y_i), \pi_3(y_i)) =_E \mathsf{true}) \\ \qquad \wedge \mathsf{checkDecKeyPf}(x_{\mathsf{pk}}, z_{\mathsf{tally}}, z_{\mathsf{decKey}}, z_{\mathsf{decKeyPf}}) =_E \mathsf{true} \\ \qquad \wedge v_1 + \cdots + v_n =_E \mathsf{dec}(z_{\mathsf{decKey}}, z_{\mathsf{tally}})\end{array}$$

Theorem 2. *$\langle V_{\mathsf{helios}}, A_{\mathsf{helios}} \rangle$ satisfies individual and universal verifiability.*

5 Eligibility Verifiability

To fully capture *election verifiability*, the tests Φ^{IV} and Φ^{UV} must be supplemented by a test Φ^{EV} that checks eligibility of the voters whose votes have been counted. We suppose that the public credentials of eligible voters appear on the bulletin board. Φ^{EV} allows an observer to check that only these individuals (that is, those in posession of credentials) cast votes, and at most one vote each.

Definition 7 (Election verifiability). *A voting specification $\langle V, A \rangle$ satisfies* election verifiability *if for all $n \in \mathbb{N}$ there exist tests $\Phi^{IV}, \Phi^{UV}, \Phi^{EV}$ such that $fn(\Phi^{IV}) = fn(\Phi^{UV}) = fn(\Phi^{EV}) = rv(\Phi^{UV}) = rv(\Phi^{EV}) = \emptyset$, $rv(\Phi^{IV}) \subseteq rv(\mathsf{R}(V))$, and for all names $\tilde{s} = (s_1, \ldots, s_n)$ we have:*

1. *The tests Φ^{IV} and Φ^{UV} satisfy each of the conditions of Definition 4;*
2. *The additional conditions 5, 6, 7 and 8 below hold.*

Let $\tilde{r} = rv(\Phi^{IV})$, $\Phi_i^{IV} = \Phi^{IV}\{{}^{s_i}/_{v}, {}^{\tilde{r}_i}/_{\tilde{r}}, {}^{y_i}/_{y}\}$, $X = fv(\Phi^{EV}) \backslash dom(\mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n))$

Soundness. For all contexts C and processes B such that $C[\mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n)] \Longrightarrow B$ and $\phi(B) \equiv \nu\tilde{n}.\sigma$, we have:

$$\Phi^{EV}\sigma \wedge \Phi^{EV}\{{}^{x'}/_{x} \mid x \in X \backslash \tilde{y}\}\sigma \Rightarrow \tilde{w}\sigma \simeq \tilde{w}'\sigma \tag{5}$$

$$\bigwedge_{1 \le i \le n} \Phi_i^{IV}\sigma \wedge \Phi^{EV}\{{}^{\tilde{w}'}/_{\tilde{w}}\}\sigma \quad \Rightarrow \quad \tilde{w}\sigma \simeq \tilde{w}'\sigma \tag{6}$$

$$\Phi^{EV}\sigma \wedge \Phi^{EV}\{{}^{x'}/_{x} \mid x \in X \backslash \tilde{w}\}\sigma \Rightarrow \tilde{y}\sigma \simeq \tilde{y}'\sigma \tag{7}$$

Effectiveness. There exists a context C and a process B such that $C[\mathsf{VP}_{\mathsf{n}}^{+}(s_1, \ldots, s_n)] \Longrightarrow B$, $\phi(B) \equiv \nu\tilde{n}.\sigma$ and

$$\bigwedge_{1 \le i \le n} \Phi_i^{IV}\sigma \wedge \Phi^{UV}\sigma \wedge \Phi^{EV}\sigma \tag{8}$$

The test Φ^{EV} is instantiated by an observer with the bulletin board. Condition (5) ensures that, given a set of ballots $\tilde{y}\sigma$, provided by the environment, Φ^{EV} succeeds only for one list of voter public credentials. Condition (6) ensures that if a bulletin board contains the ballots of voters with public credentials $\tilde{w}\sigma$ then Φ^{EV} only holds on a permutation of these credentials. Condition (7) ensures that, given a set of credentials $\tilde{w}$, only one set of bulletin board entries $\tilde{y}$ are accepted by Φ^{EV} (observe that for such a strong requirement to hold we expect the voting specification's frame to contain a public key, to root trust). Finally, the effectiveness condition is similar to Condition (4) of the previous section.

5.1 Case Study: JCJ-Civitas

The protocol due to Juels *et al.* [18] is based on mixnets and was implemented by Clarkson *et al.* [13,12] as an open-source voting system called Civitas.

How JCJ-Civitas works. An election is created by naming a set of registrars and talliers. The protocol is divided into four phases: *setup*, *registration*, *voting* and *tallying*. We now detail the steps of the protocol, starting with the setup phase.

1. The registrars (resp. talliers) run a protocol which constructs a public key pair and distributes a share of the secret part amongst the registrars' (resp. talliers'). The public part $\mathsf{pk}(sk_T)$ (resp. $\mathsf{pk}(sk_R)$) of the key is published. The registrars also construct a distributed signing key pair ssk_R, $\mathsf{pk}(ssk_R)$.

The registration phase then proceeds as follows.

2. The registrars generate and distribute voter credentials: a private part d and a public part $\mathsf{penc}(\mathsf{pk}(sk_R), m'', d)$ (the probabilistic encryption of d under the registrars' public key $\mathsf{pk}(sk_R)$). This is done in a distributed manner, so that no individual registrar learns the value of any private credential d.
3. The registrars publish the signed public voter credentials.
4. The registrars announce the candidate list $\tilde{t} = (t_1, \ldots, t_l)$.

The protocol then enters the voting phase.

5. Each voter selects her vote $s \in \tilde{t}$ and computes two ciphertexts $M = \mathsf{penc}(\mathsf{pk}(sk_T),$ $, m)s$ and $M' = \mathsf{penc}(\mathsf{pk}(sk_R), m', d)$ where m, m' are nonces. M contains her vote and M' her credential. In addition, the voter constructs a non-interactive zero-knowledge proof of knowledge demonstrating the correct construction of her ciphertexts and validity of the candidate ($s \in \tilde{t}$). (The ZKP provides protection against coercion resistance, by preventing forced abstention attacks via a *write in*, and binds the two ciphertexts for eligibility verifiability.) The voter derives her ballot as the triple consisting of her ciphertexts and zero-knowledge proof and posts it to the bulletin board.

After some predefined deadline the tallying phase commences.

6. The talliers read the n' ballots posted to the bulletin board by voters (that is, the triples consisting of the two ciphertexts and the zero-knowledge proof) and discards any entries for which the zero-knowledge proof does not hold.
7. The elimination of re-votes is performed on the ballots using pairwise *plaintext equality tests* (PET) on the ciphertexts containing private voter credentials. (A PET [16] is a cryptographic predicate which allows a keyholder to provide a proof that two ciphertexts contain the same plaintext.) Re-vote elimination is performed in a verifiable manner with respect to some publicly defined policy, *e.g.*, by the order of ballots on the bulletin board.
8. The talliers perform a verifiable re-encryption mix on the ballots (ballots consist of a vote ciphertext and a public credential ciphertext; the link between both is preserved by the mix.) The mix ensures that a voter cannot trace her vote, allowing the protocol to achieve coercion-resistance.
9. The talliers perform a verifiable re-encryption mix on the list of public credentials published by the registrars. This mix anonymises public voter credentials, breaking any link with the voter for privacy purposes.
10. Ballots based on invalid credentials are weeded using PETs between the mixed ballots and the mixed public credentials. Both have been posted to the bulletin board. (Using PETs the correctness of weeding is verifiable.)
11. Finally, the talliers perform a verifiable decryption and publish the result.

Equational theory. The protocol uses a variant of the ElGamal encryption scheme [18]. Accordingly we adopt the signature and associated equational theory from the Helios case study. We model the ZK proof demonstrating correct construction of the voter's ciphertexts, re-encryption and PETs by the equations

$$\begin{aligned}
&\mathsf{checkBallot}(\mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{text}}, x'_{\mathsf{pk}}, x'_{\mathsf{rand}}, x'_{\mathsf{text}}),\\
&\qquad\qquad \mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{text}}), \mathsf{penc}(x'_{\mathsf{pk}}, x'_{\mathsf{rand}}, x'_{\mathsf{text}})) = \mathsf{true}\\
&\mathsf{renc}(y_{\mathsf{rand}}, \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{text}})) = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), f(x_{\mathsf{rand}}, y_{\mathsf{rand}}), x_{\mathsf{text}})\\
&\mathsf{pet}(\mathsf{petPf}(x_{\mathsf{sk}}, ciph, ciph'), ciph, ciph') = \mathsf{true}
\end{aligned}$$

where $ciph \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{text}})$ and $ciph' \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x'_{\mathsf{rand}}, x_{\mathsf{text}})$. In addition we consider verifiable re-encryption mixnets and introduce for each permutation χ on $\{1, \ldots, n\}$ the equation:

$$\begin{aligned}
&\mathsf{checkMix}(\mathsf{mixPf}(x_{\mathsf{ciph},1}, \ldots, x_{\mathsf{ciph},n}, ciph_1, \ldots, ciph_n, z_{\mathsf{rand},1}, \ldots, z_{\mathsf{rand},n}),\\
&\qquad\qquad x_{\mathsf{ciph},1}, \ldots, x_{\mathsf{ciph},n}, ciph_1, \ldots, ciph_n) = \mathsf{true}
\end{aligned}$$

where $ciph_i \mathrel{\hat{=}} \mathsf{renc}(z_{\mathsf{rand},i}, x_{\mathsf{ciph},\chi(i)})$. We also define re-encryption of pairs of ciphertexts and introduce for each permutation χ on $\{1, \ldots, n\}$ the equation

$$\begin{aligned}
&\mathsf{checkMixPair}(\mathsf{mixPairPf}(\langle x_1, x'_1\rangle, \ldots, \langle x_n, x'_n\rangle, \langle c_1, c'_1\rangle, \ldots, \langle c_n, c'_n\rangle,\\
&\quad \langle z_1, z'_1\rangle, \ldots, \langle z_n, z'_n\rangle), \langle x_1, x'_1\rangle, \ldots, \langle x_n, x'_n\rangle, \langle c_1, c'_1\rangle, \ldots, \langle c_n, c'_n\rangle) = \mathsf{true}
\end{aligned}$$

where $c_i \mathrel{\hat{=}} \mathsf{renc}(z_i, x_{\chi(i)})$ and $c'_i \mathrel{\hat{=}} \mathsf{renc}(z'_i, x'_{\chi(i)})$.

Model in applied pi. We make the following trust assumptions for verifiability

- The voter is able to construct her ballot; that is, she is able to generate nonces m, m', construct her ciphertexts and generate a zero-knowledge proof.
- The registrars construct distinct credentials d for each voter and construct the voter's public credential correctly. (The latter assumption can be dropped if the registrars provides a proof that the public credential is correctly formed [18].) The registrars keep the private part of the signing key secret.

Although neither voters nor observers can verify that the registrars adhere to such expectations, they trust them because trust is distributed. The trusted components are modelled by the voting process specification $\langle A_{\mathsf{jcj}}, V_{\mathsf{jcj}}\rangle$ (Definition 8). The context A_{jcj} distributes private keys on a private channel, launches an unbounded number of registrar processes and publishes the public keys of both the registrars and talliers. The registrar R constructs a fresh private credential d and sends the private credential along with the signed public part (that is, $\mathsf{sign}(ssk_R, \mathsf{penc}(x_{pk_R}, m'', d))$) to the voter; the registrar also publishes the signed public credential on the bulletin board. The voter V_{jcj} receives the private and public credentials from the registrar and constructs her ballot; that is, the pair of ciphertexts and a zero-knowledge proof demonstrating their correct construction.

Definition 8. *The voting process specification $A_{\mathsf{jcj}}, V_{\mathsf{jcj}}$ is defined where:*

$$
\begin{aligned}
A_{\mathsf{jcj}} &\mathrel{\hat{=}} \nu\, a, ssk_R.(!R \mid \{\mathsf{pk}(sk_R)/_{x_{\mathsf{pk_R}}}, \mathsf{pk}(ssk_R)/_{x_{spk_R}}, \mathsf{pk}(sk_T)/_{x_{\mathsf{pk_T}}}\} \mid \_)\\
V_{\mathsf{jcj}} &\mathrel{\hat{=}} \nu\, m, m'.a(x_{cred}).\mathit{let}\ ciph = \mathsf{penc}(x_{pk_T}, m, v)\ \mathit{in}\\
&\quad \mathit{let}\ ciph' = \mathsf{penc}(x_{pk_R}, m', \pi_1(x_{cred}))\ \mathit{in}\\
&\quad \mathit{let}\ zkp = \mathsf{ballotPf}(x_{pk_T}, m, v, x_{pk_R}, m', \pi_1(x_{cred}))\ \mathit{in}\\
&\quad \overline{c}\langle(\!|ciph, ciph', zkp|\!)\rangle\\
R &\mathrel{\hat{=}} \nu\, d, m''.\mathit{let}\ sig = \mathsf{sign}(ssk_R, \mathsf{penc}(x_{pk_R}, m'', d))\ \mathit{in}\ \overline{a}\langle(\!|d, sig|\!)\rangle.\overline{c}\langle sig\rangle
\end{aligned}
$$

Election verifiability. We suppose the recording function uses record variables $\tilde{r} = (r_{cred}, r_m, r_{m'}) = \mathrm{rv}(\mathsf{R}(V))$ (corresponding to the variable x_{cred} and names m, m' in the process V). Accordingly, given $n \in \mathbb{N}$ we define:

$$
\begin{aligned}
\Phi^{IV} &\mathrel{\hat{=}} y =_E (\!|\mathsf{penc}(x_{\mathsf{pk_T}}, r_m, v), \mathsf{penc}(x_{\mathsf{pk_R}}, r_{m'}, \pi_1(r_{cred})),\\
&\qquad \mathsf{ballotPf}(x_{\mathsf{pk_T}}, r_m, v, x_{\mathsf{pk_R}}, r_{m'}, \pi_1(r_{cred}))|\!) \wedge w = \pi_2(r_{cred})\\
\Phi^{UV} &\mathrel{\hat{=}} \mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(y_1), \pi_2(y_1)|\!), \ldots, (\!|\pi_1(y_n), \pi_2(y_n)|\!),\\
&\qquad z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},n}) =_E \mathsf{true}\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{n} \mathsf{dec}(z_{\mathsf{decKey},i}, \pi_1(z_{\mathsf{bal},i})) =_E v_i\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{n} \mathsf{checkDecKeyPf}(x_{\mathsf{pk_T}}, \pi_1(z_{\mathsf{bal},i}), z_{\mathsf{decKey},i}, z_{\mathsf{decPf},i}) =_E \mathsf{true}\\
\Phi^{EV} &\mathrel{\hat{=}} \textstyle\bigwedge_{i=1}^{n} \mathsf{checkBallot}(\pi_3(y_i), \pi_1(y_i), \pi_2(y_i))\\
&\quad \wedge \mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(y_1), \pi_2(y_1)|\!), \ldots, (\!|\pi_1(y_n), \pi_2(y_n)|\!),\\
&\qquad z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},n}) =_E \mathsf{true}\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{n} \mathsf{pet}(z_{\mathsf{petPf},i}, \pi_2(z_{\mathsf{bal},i}), \hat{z}_{\mathsf{cred},i}) =_E \mathsf{true}\\
&\quad \wedge (\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},n}|\!) \simeq (\!|\hat{z}_{\mathsf{cred},1}, \ldots, \hat{z}_{\mathsf{cred},n}|\!)\\
&\quad \wedge \mathsf{checkMix}(z_{\mathsf{mixPf}}, \mathsf{getmsg}(w_1), \ldots, \mathsf{getmsg}(w_n), z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},n}) =_E \mathsf{true}\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{n} \mathsf{checksign}(x_{spk_R}, w_i)
\end{aligned}
$$

Theorem 3. $\langle A_{\mathsf{jcj}}, V_{\mathsf{jcj}}\rangle$ *satisfies election verifiability.*

6 Conclusion

We present a symbolic definition of election verifiability which allows us to precisely identify which parts of a voting system need to be trusted for verifiability. The suitability of systems can then be evaluated and compared on the basis of trust assumptions. We also consider eligibility verifiability, an aspect of verifiability that is often neglected and satisfied by only a few protocols, but nonetheless an essential mechanism to detect ballot stuffing. We have applied our definition to three protocols: FOO, which uses blind signatures; Helios 2.0, which is based on homomorphic encryption, and JCJ-Civitas, which uses mixnets and anonymous credentials. For each of these protocols we discuss the trust assumptions that a voter or an observer needs to make for the protocol to be verifiable. Since Helios 2.0 and JCJ-Civitas have been implemented and deployed, we believe our formalisation is suitable for analysing real world election systems.

Acknowledgements

We are particularly grateful to Michael Clarkson for careful reading of our preliminary formal definition of election verifiability. His comments provided useful guidance for the definition we present here.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL 2001: Proc. 28th ACM Symposium on Principles of Programming Languages, pp. 104–115. ACM, New York (2001)
2. Adida, B.: Advances in Cryptographic Voting Systems. PhD thesis, MIT (2006)
3. Adida, B.: Helios: Web-based open-audit voting. In: Proc. 17th Usenix Security Symposium, pp. 335–348. USENIX Association (2008)
4. 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: Electronic Voting Technology/Workshop on Trustworthy Elections, EVT/WOTE (2009)
5. Anderson, R., Needham, R.: Programming Satan's Computer. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 426–440. Springer, Heidelberg (1995)
6. Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: CSF 2008: Proc. 21st IEEE Computer Security Foundations Symposium, Washington, USA, pp. 195–209. IEEE, Los Alamitos (2008)
7. Baskar, A., Ramanujam, R., Suresh, S.P.: Knowledge-based modelling of voting protocols. In: TARK 2007: Proc. 11th International Conference on Theoretical Aspects of Rationality and Knowledge, pp. 62–71. ACM, New York (2007)
8. Bowen, D.: Secretary of State Debra Bowen Moves to Strengthen Voter Confidence in Election Security Following Top-to-Bottom Review of Voting Systems. California Secretary of State, press release DB07:042 (August 2007), http://www.sos.ca.gov/elections/voting_systems/ttbr/db07_042_ttbr_system_decisions_release.pdf
9. Bundesverfassungsgericht (Germany's Federal Constitutional Court). Use of voting computers in 2005 Bundestag election unconstitutional. Press release 19/2009 (March 2009), http://www.bundesverfassungsgericht.de/en/press/bvg09-019en.html

10. Chaum, D., Ryan, P.Y.A., Schneider, S.: A practical, voter-verifiable election scheme. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005)
11. Chevallier-Mames, B., Fouque, P.-A., Pointcheval, D., Stern, J., Traore, J.: On Some Incompatible Properties of Voting Schemes. In: WOTE 2006: Proc. Workshop on Trustworthy Elections (2006)
12. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. Technical Report 2007-2081, Cornell University (May 2007), http://hdl.handle.net/1813/7875 (revised March 2008)
13. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: S&P 2008: Proc. Symposium on Security and Privacy, pp. 354–368. IEEE, Los Alamitos (2008)
14. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17(4), 435–487 (2009)
15. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: ASIACRYPT 1992: Proc. Workshop on the Theory and Application of Cryptographic Techniques, pp. 244–251. Springer, Heidelberg (1992)
16. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 162–177. Springer, Heidelberg (2000)
17. Juels, A., Catalano, D., Jakobsson, M.: Coercion-Resistant Electronic Elections. Cryptology ePrint Archive, Report 2002/165 (2002)
18. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: WPES 2005: Proc. Workshop on Privacy in the Electronic Society, pp. 61–70. ACM, New York (2005)
19. Kremer, S., Smyth, B., Ryan, M.D.: Election verifiability in electronic voting protocols. Technical Report CSR-10-06, University of Birmingham, School of Computer Science (2010), http://www.bensmyth.com/publications/10tech/CSR-10-06.pdf
20. Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (Netherland's Ministry of the Interior and Kingdom Relations). Stemmen met potlood en papier (Voting with pencil and paper). Press release (May 2008), http://www.minbzk.nl/onderwerpen/grondwet-en/verkiezingen/nieuws--en/112441/stemmen-met-potlood
21. Participants of the Dagstuhl Conference on Frontiers of E-Voting. Dagstuhl accord (2007), http://www.dagstuhlaccord.org/
22. Ryan, M.D., Smyth, B.: Applied pi calculus. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols, ch. 6. IOS Press, Amsterdam (2010)
23. Smyth, B., Ryan, M.D., Kremer, S., Kourjieh, M.: Towards automatic analysis of election verifiability properties. In: Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS 2010). LNCS. Springer, Heidelberg (2010)
24. Talbi, M., Morin, B., Tong, V.V.T., Bouhoula, A., Mejri, M.: Specification of electronic voting protocol properties using ADM logic: FOO case study. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 403–418. Springer, Heidelberg (2008)
25. UK Electoral Commission. Key issues and conclusions: electoral pilot schemes (May 2007), http://www.electoralcommission.org.uk/elections/pilots/May2007

Pretty Good Democracy for More Expressive Voting Schemes

James Heather[1], Peter Y.A. Ryan[2], and Vanessa Teague[3]

[1] Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH, UK
j.heather@surrey.ac.uk
[2] Dept. Computer Science and Communications, University of Luxembourg
peter.ryan@uni.lu
[3] Dept. Computer Science and Software Engineering, University of Melbourne
vteague@csse.unimelb.edu.au

Abstract. In this paper we revisit *Pretty Good Democracy*, a scheme for verifiable Internet voting from untrusted client machines. The original scheme was designed for first-past-the-post elections. Here, we show how Pretty Good Democracy can be extended to voting schemes in which the voter lists the candidates in their order of preference. Our scheme applies to elections using STV, IRV, Borda, or any other tallying scheme in which a vote is a list of candidates in preference order. We also describe an extension to cover Approval or Range voting.

1 Introduction

Secure Internet voting wouldn't be difficult at all, if only the authorities tallying the election were perfectly trustworthy, nobody ever attempted to influence another person's vote, and every home PC was perfectly secure. Unfortunately, we have in general no grounds to make such assumptions. There are various schemes for Internet voting [JCJ05, Adi08], which use cryptography to weaken or eliminate (some of) these assumptions. Here we concentrate on *Pretty Good Democracy* (PGD) [RT09], which provides a proof of correct tallying and guarantees of ballot privacy that do not depend on any assumptions about the device used to cast the vote.

PGD is a form of *code voting*, which uses a separate channel (such as the postal service) to deliver a *code sheet* to each voter. The voter casts their vote by selecting the appropriate code(s) from their sheet. They then check that they receive an acknowledgement code that matches the one on their sheet. This provides mutual authentication between the voter and the authorities. In PGD [RT09], we added a proof of correct tallying on a web bulletin board. Although obviously not secure against a coercer who observes both the code sheet and the voter's communications, PGD is receipt-free as long as *either* the code sheet *or* the voter's messages to the voting authorities remain secret. As far as we know, it is the only scheme to provide both receipt-freeness and verifiability on an untrusted voting client. The main disadvantage of PGD relative to other schemes is that the election's integrity depends on correct behaviour of a threshold of

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 405–423, 2010.

trustees. This is not as onerous a requirement as it may first seem, however: even in national governmental elections, this can be solved by giving threshold shares to each political party, along with various neutral organisations such as the UN, so that one can have confidence that a threshold set will not rig the election because there is no result that all its members would be content with. Nevertheless, this arguably makes PGD inappropriate for elections of significant political importance, though there are many other hard-fought elections in which privacy and evidence of correctness are important, such as elections in trade unions and professional societies.

The first version of PGD [RT09] was designed for elections in which the voter chose a single favourite candidate. However, many countries and many other organisations use voting schemes requiring the voter to list several (or all) candidates in their order of preference. For example, the Single Transferable Vote scheme (STV) is used in national elections in Australia, Ireland, Malta and Scotland. Instant Runoff Voting (IRV), also known as the Alternative Vote (AV), which is the single-vacancy version of STV, is used in some local elections in the USA, the UK, Australia, and many other countries, and it is widely expected that there will soon be a referendum on using it for UK parliamentary elections. The Borda Count is used in certain political elections in Slovenia, and also in many other organisations, such as the Eurovision Song Contest. In this paper, we extend PGD to allow voters to express their vote as a list of candidates in preference order. Any method could then be used to tally the votes, including existing solutions for the secure tallying of Borda [CM05] or STV/IRV votes [BMN+09, Hea07, WB09]. We present three different protocols, creatively named A, B and C. Protocol C also encompasses Approval Voting or Range Voting, in which the voter gives a score to all candidates.

Like PGD 1.0, all the protocols in this paper are receipt-free but not coercion-resistant: a voter can sell her code sheet before voting, but cannot prove after voting what vote she cast. The protocols with a single ack (A and C) are receipt-free even against a coercer who directly observes the ack return. Protocol B is receipt-free only if the voter has a chance to generate a fake ack code list before the coercer observes it.

In all cases, it takes either a leak of the printed code sheet or a collusion of a threshold number of trustees (which, by assumption, cannot occur) to derive an appropriate Ack Code without the vote being properly registered on the bulletin board. The assumptions behind integrity are described in more detail in Section 1.2.

In Section 1.1 we review PGD 1.0 and explain why the obvious extension to more complex voting schemes fails, then in 1.2 we give an overview and comparison of the extensions to more expressive voting schemes. The next three sections each contain a different extension, with a discussion of pros and cons. In Section 2, the simplest method (Protocol A) is described, which is secure but suffers from the disadvantage that each preference requires a separate interaction with the authorities. Protocol B, in Section 3, has the simplest voting experience, but somewhat complicated acknowledgement checking. Protocol C, in Section 4,

is an approach based on a two-dimensional table, which allows votes that are ordered lists or approval or range votes.

1.1 Review of PGD 1.0

Like other forms of Code Voting [Cha01], PGD assumes that each voter receives a *Code Sheet*, which is a list of candidate names and corresponding Vote Codes. We assume that the vote codes are kept secret and revealed only to the voter; mechanisms and procedures to support this assumption are discussed in detail in [RT09]. An example of a Code Sheet is given in Figure 1.

Candidate	Vote Code
Chequered Red	3772
Cross	4909
Fuzzy	9521
Green	7387
Red	2531
Ack Code: 8243	
Ballot ID: 3884092844	

Fig. 1. Example Vote Codes

Each voter sends the Vote Code for their chosen candidate to the central Vote Server. They could use any networked device for the transmission, including a home PC or mobile phone. Even a corrupted device is unable to substitute a different choice of candidate, because it does not learn the other codes.

After sending the Vote Code, the voter waits to receive an *acknowledgement code*. In the original Code Voting scheme [Cha01], the printed code sheet contained a separate Ack Code for each candidate. In PGD [RT09] we argued that one Ack code per code sheet sufficed. Either way, the purpose of the Ack is to demonstrate to the voter that they communicated with the correct server and that it received and registered the correct Vote Code. In PGD there was then a computer-verifiable proof of correct tallying, which could be publicised on a bulletin board.

The key innovation of the PGD scheme is that, in order to access the correct ack code, the voter server must invoke the co-operation of a threshold set of Trustees. The revealing of the correct ack code is thus a side-effect of the correct registration on the bulletin board of a valid code.

Cryptographic tools. This protocol relies on two main cryptographic tools:

Verifiable re-encryption shuffles (for example, [FS01, Gro03, Nef01, JJR02]) allow an authority to permute encrypted values (votes) by a secret permutation, while providing a publicly verifiable proof on the bulletin board that some shuffle has been correctly performed.

Plaintext Equivalence Tests (PETs) ([JJ00, TH08]), allow a threshold set of authorities who share a public key to compare two values encrypted with that key, and determine whether the two values are equal or not, without learning any other information.

Overview of ballot construction. The roles of the authorities in PGD are:

- A *Voting Authority* VA who generates the requisite number of vote codes and ack codes encrypted under the Trustees' public key, PK_T.
- A set of *Clerks*, who generate encrypted Vote Codes for each ballot, one version for the Bulletin Board and one for the printed code sheets.
- A *Registrar* who decrypts the ballots provided by the Clerks and prints the code sheets.
- A *Returning Officer* who distributes the code sheets to the voters.
- A *Voting Server*, who receives the votes, then posts the ballot ID and the encrypted vote code on the Bulletin Board along with a Zero Knowledge proof of knowledge of the plaintext.
- A set of *Trustees*, who work with the Voting Server to register the votes on the Bulletin Board (BB) and reveal the ack codes. They have shares of the secret key corresponding to the threshold public key, PK_T. "Registration" means recording the vote on the BB. (In the extension protocols in Sections 2, 3 and 4, the Trustees will be split into several authorities with slightly different roles.)
- A set of Auditors responsible for performing various types of audit, on the initial set-up, on the information posted to the Bulletin Board, *e.g.* the zero knowledge proofs, and verifying the anonymising shuffles and final decryption steps. The auditors are not privileged or trusted, and do not receive secret information—any member of the public can be an auditor.

Full details of PGD 1.0 can be found in [RT09]. Here, for completeness, we give a brief outline. First, a sufficiently large set of voting and ack codes of the appropriate form are generated. These are encrypted under the Teller's threshold public key and put through a sequence of re-encryption shuffles. These codes are then assembled into a table of the appropriate dimensions: each row will have n encrypted vote codes and an encrypted ack code and will correspond to a code sheet. This table is referred to as the P-table. This construction ensures that only certain sets of entities acting in collusion could compromise the secrecy of the codes. The Registrar decrypts the information of the P-table and prints the vote codes to the code sheets.

All of this is done on the bulletin board, except obviously the Registrar's decryption of the ballots and the distribution of the code sheets to voters.

In order to ensure receipt-freeness, the P-table must be transformed to conceal the correspondence between the position in the row of an (encrypted) code and the candidate it represents. Each row of the P-table is therefore subjected to a further sequence of re-encryption permutations to create the Q-table. The Q-table is used to register the votes.

When votes are cast, the encrypted Vote Code supplied by the Vote Server is matched by a threshold set of Trustees via plaintext equivalence tests against the encrypted codes in the appropriate row of the Q-table. If a match is found, the matching term is flagged and the threshold set of trustees decrypt the ack code. The fact that the terms of the Q-table are permuted ensures that flagging one does not reveal what candidate this represents. The information defining the permutation is stored in an encrypted *onion* in the usual Prêt à Voter fashion and this is later used during tabulation to interpret the flagged term to identify the chosen candidate.

The rest of the tallying is similar to Prêt à Voter. In this paper we present two protocols with single ack codes that can be published on the bulletin board, and one protocol with an ordered list of ack codes that must be decrypted and returned to the voter secretly.

An obvious extension to preference voting that fails. The simplest extension would be for the voter simply to list their vote codes in preference order, and wait for the (single) return Ack. However, this is insecure because a cheating client or VS could simply rearrange the codes undetectably.

1.2 New Protocol Comparison

Figure 2 contains a functional comparison of the three new protocols presented in this paper. "Single-step voting" means that casting a vote requires only one interaction with the authorities. "Single ack" means that there is only one Ack code—this is important because it means that the protocol is receipt-free even against a coercer who observes the ack return directly. "Number of preferences hidden on BB" means that observers cannot tell from the bulletin board how many preferences each voter cast. This is sometimes important, because different jurisdictions have very different rules about how many preferences may or must be cast. Being able to check via the bulletin board is a useful feature for demonstrating vote validity, though it may make voters vulnerable to being coerced into casting fewer (or more) peferences than they wished.

Protocol	A	B	C
Single-step voting	×	✓	✓
Single Ack	✓	×	✓
Number of preferences hidden on BB	✓	×	×
Approval or Range Votes	×	×	✓

Fig. 2. Comparison of protocol features

Security properties are compared in Figure 3, which also includes PGD 1.0. It shows under what circumstances a corrupt device or Voting Server can manipulate the vote. "No" is good and "yes" is bad. "Knowing" a code means knowing its meaning, *i.e.* its candidate or preference.

Cheating client or VS can undetectably:	PGD 1.0	A	B	C
Truncate the vote knowing only the Vote Codes	n/a	Yes	No	Yes
Truncate the vote knowing only the Ack Codes	n/a	Yes	Yes	Yes
Otherwise manipulate the vote knowing only the Vote Codes	Yes	Yes	No	Yes
Otherwise manipulate the vote knowing only the Ack Codes	No	Yes	Yes	No
Otherwise manipulate the vote knowing only the order of candidates on the code sheet	n/a	n/a	Y	n/a
Manipulate the vote knowing neither Vote nor Ack Codes	No	No	No	No

Fig. 3. Comparison of protocol security properties

2 Protocol A: The Simple Solution

Another possibility is to use distinct Ack codes for each candidate, sent secretly to the voter in addition to the public one that is posted on the bulletin board (see Section 9.3 of [RT09]). The voter would have to send in each code in turn, then wait for the appropriate Ack to be received before sending in the next code, and so on.

2.1 Discussion

This is a secure and simple solution—it is impossible for a cheating client to switch vote codes or candidate acks undetectably, and it is easy for the voter to understand why. Its only shortcoming is that it could take some time for the authorities to generate and return the acks, during which time the voter has to wait. Furthermore, the security is undermined if a malicious client machine successfully persuades the voter to enter all their vote codes in one go without waiting for the intermediate acks, because the client could then apply the same rearrangement to both the vote codes and the ack codes.

3 Protocol B: Returning the Acknowledgement Codes in Ballot Order

The idea is to give each voter a code sheet with two lists of codes:

- a list of candidate codes in a random, secret order, and
- a list of preference codes in preference order.

Voting is a simple matter of sending in the vote codes in order of preference. The return acknowledgement should be a list of preference codes in the order the candidates appear on the code sheet, which is computed by the authorities without requiring any voter interaction. The voting protocol is thus a simple two-pass process: the voter sends her list of vote codes and then checks the sequence of preference codes. The main drawbacks with this are that it may be difficult for voters to understand how to check their preference codes, and that the integrity guarantee is not as strong as that offered by Protocol A.

3.1 Security Properties

The integrity guarantee for this protocol depends on the candidates being listed in a secret, random order. Our main security claim is:

Claim. A cheating client or VS (who doesn't know the meaning of the preference codes) can swap two preferences undetectably only if it knows which two positions on the code sheet they correspond to.

Proof. See Section 3.4.

Furthermore, the protocol is receipt-free if the voter keeps secret either their code sheet or their communications with the Vote Server. This is stated more precisely in Section 3.4 and proved in Appendix A.

3.2 Voter Interface Details

The idea is that the voter submits their candidate codes in their order of preference, and receives as acknowledgement a list of preference codes in the order the candidates appear on their code sheet.

For example, for the code sheet given in Figure 4(a), the voter might wish to vote "Chequered, Fuzzy, Green, Red, Cross", so they would enter codes $9521, 7387, 4909, 3772, 2531$ in sequence. At this point they have finished casting their vote, and if they are not interested in verifying their vote, they do not need to interact with the system any further (though obviously a cheating client or VS must not know in advance that this voter will not check their returned preference codes, or they could rearrange the vote).

Candidate	Vote Code
Red	3772
Green	4909
Chequered	9521
Fuzzy	7387
Cross	2531
Ack Code: 8243	
Ballot ID: 3884092844	

(a) Vote Codes

Preference	Ack Code
1st	K
2nd	T
3rd	C
4th	W
5th	M
Ballot ID: 3884092844	

(b) Preference Codes

Fig. 4. Example Code sheets

If the voter does take the trouble to verify the registration of her vote, she should expect as acknowledgement a list of preference codes given in the order the candidates are printed on the code sheet. For the example preference codes in Figure 4(b), the first would be code W, (because Red is the first candidate in the order printed on the ballot paper, and the preference given to it was 4th), then C, K, T, M. Thus the voter should expect to receive the acknowledgement: $WCKTM$.

To assist the voter, we could provide a blank column alongside the candidate list. The voter writes the appropriate preference code for each candidate alongside the candidate, as shown in Figure 5. Then the acknowledgement code will be the sequence of letters read down the column, thus:

Candidate	Vote Code	Pref code
Red	3772	W
Green	4909	C
Chequered	9521	K
Fuzzy	7387	T
Cross	2531	M
Ack Code: 8243		
Ballot ID: 3884092844		

Fig. 5. Example code sheet with preference codes filled in

Note that it would be possible to provide the voter with three sheets:

- A conventional code sheet showing the voting codes against the candidates with the candidates in canonical (*e.g.* lexical) order.
- The sheet showing the preference codes, as before.
- A sheet showing the candidates in the permuted order with the blank column alongside for the voter to fill in the preference codes.

Indeed, all three sheets could be generated and distributed via different processes and channels for added security. Whether this would help the voter and add sufficiently to the security to justify the additional costs would depend on the individual case.

3.3 Details of Ballot Construction, Acknowledgement and Tallying

Notation. If σ and π are permutations on n items, then $\sigma \circ \pi$ is the permutation defined by $(\sigma \circ \pi)(i) = \sigma(\pi(i))$. If L is a list, then L_i denotes the i-th element of L. Denote by $\pi(L)$ the idea of "applying" a permutation π to a list L, which means taking each element L_i in turn and copying it into position $\pi(i)$ in the new list. The result is $\pi(L) = L_{\pi^{-1}(1)}, \ldots, L_{\pi^{-1}(n)}$. It follows that the result of applying π and then σ to L is $(\sigma \circ \pi)(L) = L_{\pi^{-1}(\sigma^{-1}(1))}, \ldots, L_{\pi^{-1}(\sigma^{-1}(n))}$.

In what follows, we will use '$[x]$' to denote the encryption of x. (Almost everything is encrypted, so the notation is just a reminder.)

Building Blocks. Numerous protocols exist for proving a shuffle of a list of ciphertexts. In [RT10], efficient protocols are given for proving that the *same* shuffle has been applied to several lists, even if they are encrypted under different public keys. We will call this protocol *Shuf-par*.

Ballot Construction: The Bulletin-Board part. We use a distributed ballot construction similar to that of PGD. Obviously we need full permutations rather than cyclic shifts. For each vote ID, we need to produce a printed code sheet as described above. There are five different authorities, each of which could be performed by a single (trustworthy) individual, or (preferably) distributed among several.

1. The *ballot-construction authorities* produce the codes and a randomly-arranged encrypted version of each code sheet, on the bulletin board.
2. the *code-sheet authority* randomly reshuffles and then prints the code sheets. (The shuffling and decrypting can be distributed using standard techniques, but the printing is more difficult to distribute.)
3. the *PET authorities* share the key with which the Vote Codes are encrypted. They perform distributed PET tests on the bulletin board to register each vote.
4. the *Output shuffle authorities* transform the votes into the correct order for tallying and the preference codes into the correct order for returning to voters.
5. the *decryption authorities* share the key for decrypting the candidate names in each vote.

Each row corresponds to one code sheet, *i.e.* one vote. To avoid cluttering the text we drop the indices that indicate the row, and just describe the set up w.r.t. a typical row. Let c_i be the i-th candidate, and VC_i the i-th vote code. The *ballot-construction authorities* begin by constructing, for each vote, a table similar to the P-table described above and displaying on the BB. Alongside each encrypted vote code we add the encryption of the corresponding candidate. We thus have a table in which each row comprises:

1. A list $\mathcal{VC}$ of encrypted (c_i, VC_i) pairs in a canonical order.

   Now, for each row, the pairs are subjected to a sequence of re-encryption shuffles to yield:
2. A re-encrypted version of $\mathcal{VC}$ with each row shuffled by a secret random order ρ.

$$\rho(\mathcal{VC}) = ([c_{\rho^{-1}(1)}], [VC_{\rho^{-1}(1)}]), ([c_{\rho^{-1}(2)}], [VC_{\rho^{-1}(2)}]), \ldots, ([c_{\rho^{-1}(n)}], [VC_{\rho^{-1}(n)}])$$

   This table we will again refer to as the Q-table.

   Each row of this table has to be decrypted and the information printed on a code sheet. Note that the candidates will be printed in the in the order given, *i.e.* according to the ρ permutation encoded in this sequence.
3. A table $\mathcal{PC}$ of encrypted preference codes in order is also posted to the Bulletin Board. Each row will correspond to a code sheet and will have the form:

$$\mathcal{PC} = [PC_1], \ldots, [PC_n]$$

The $\mathcal{VC}^*$-Table. Now we generate the S-table that will serve to register the votes. We need to introduce further permutations to the rows in order to ensure that the scheme is receipt free. First, in order to keep track of this permutation, we add to the ith pair an encryption of i. This each row is a list of triples of the form:

$$\mathcal{VC}' = ([1], [c_{\rho^{-1}(1)}], [VC_{\rho^{-1}(1)}]), \ldots, ([n], [c_{\rho^{-1}(n)}], [VC_{\rho^{-1}(n)}])$$

Another set of authorities called the *Code Sheet authorities* then perform further shuffles within each row of the Vote Codes, by another secret, parallel, random permutation σ_i, where i indexes the row in question. The protocol of [RT10] is used here to ensure that the triples are preserved in these shuffles. The output of this is posted to the Bulletin Board.

The result of this will be a new table, $\mathcal{VC}^*$, in which each row has the form:

$$\mathcal{VC}^* = ([\sigma^{-1}(1)], [c_{\rho^{-1}\circ\sigma^{-1}(1)}], [VC_{\rho^{-1}\circ\sigma^{-1}(1)}]), \ldots, ([\sigma^{-1}(n)], [c_{\rho^{-1}\circ\sigma^{-1}(n)}], [VC_{\rho^{-1}\circ\sigma^{-1}(n)}])$$

This table will be posted to the Bulletin Board and used to register the votes. Notice that the order in which the candidates, and the vote codes, appear is different to that that appears on the code sheets, in fact differs by the secret σ permutation. This is crucial to ensure that the scheme is receipt-free.

The authorities are also required to show their workings on the Bulletin Board to allow for auditing.

Ack computation and return. When a vote $\mathcal{V}$ arrives with the Trustees (from the VS) it's an encrypted list of vote codes in preference order:

$$\mathcal{V} = ([VC_{\pi^{-1}(1)}], \ldots, [VC_{\pi^{-1}(n)}])$$

For convenience we will assume throughout this section that each vote is a *complete* list of preferences (that is, it includes every candidate). However, partial lists could easily be accommodated, though the tallying would reveal how many preferences had been expressed. This issue is discussed further in Section 4.

The authorities construct the *tallyable vote* T and the *acknowledgement list* A on the BB as follows:

1. The PET authorities perform PET tests comparing the terms of the vote $\mathcal{V}$ with the list $\mathcal{VC}^*$ from the bulletin board, until they have found all possible matches.[1][2] When $[\mathcal{V}_j]$ matches $[\mathcal{VC}^*]_i$, this means that $\pi^{-1}(j) = \rho^{-1}\circ\sigma^{-1}(i)$, so candidate $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ gets preference j. The following transformations can be perfomed (and verified) by anyone:

[1] Note that $\sigma \circ \rho$ is secret, *i.e.* not the permutation that's printed on the code sheets, so this does not reveal anything about the vote. If a party knows $\sigma \circ \rho$, or knows σ and has the code sheet, they can learn the vote from this step, which is a good reason to have ρ and σ generated by a series of shufflers.

[2] This could require n^2 PETs.

(a) **Vote Updating:** Put $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ into the vote T at preference j. (For example, T could just be a list of candidate names in order, in which case all we do is add $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ into the list T in the j-th place.) Since $[c_{\rho^{-1}\circ\sigma^{-1}(i)}]$ is still encrypted, nobody knows which candidate actually received preference j.

(b) **Ack code updating:** To construct the correct acknowledgement code, extract $[\sigma^{-1}(i)]$ from $[\mathcal{VC}^*{}_i]$ and append to A the term

$$([\sigma^{-1}(i)], [PC_j])$$

2. Once all the terms in the row have been registered and ranked, we have a sequence of pairs of the form:

$$([\sigma^{-1}(i)], [PC_j]), \text{ for } i = 1, \ldots, n$$

in which $\pi^{-1}(j) = \rho^{-1} \circ \sigma^{-1}(i)$. Now the preference codes must be arranged in the correct order, corresponding to the order shown on the code sheet. We want to do this in a way that does not result in the authorities, or anyone, learning the σ shuffle. We can accomplish this as follows: the output shuffle authorities each perform a parallel shuffle on the sequence, preserving the pairings. Once this is done, a threshold set of the decryption authorities decrypt all the terms. The preference codes are now arranged into the order of the first terms. It is clear by construction that this puts the preference codes into the candidate order of the code sheet.

Tallying. Since the votes are simply lists of encrypted candidate names in preference order, there are many possible tallying options depending on the voting scheme and on the degree of privacy required. Any of the secure tallying protocols for STV/IRV or Borda mentioned in the introduction could be implemented here.

3.4 Proofs of Correctness

Basic proof of correctness. It should already be clear, but is important to state, that when everyone follows the protocol the votes are cast and counted as the voter intended.

Lemma 1. *When all authorities follow the protocol correctly, the vote registered is the same as the permutation applied by the voter to the Vote Codes, which is also the same as the vote implied by the acknowledging preference codes.*

Proof. By construction. □

Proof of security against a cheating client. A malicious client can not undetectably cast a modified vote. Specifically, it cannot swap two candidates or preferences unless it knows the position of the corresponding candidates on

the code sheet, or the meanings of the relevant preference codes. Here we restate the claim and sketch a proof. Of course, if there are only two candidates then a swap can indeed be performed undetectably. The probability of successfully guessing the candidates positions is 1 in $\binom{n}{2}$.

Claim. A cheating client or VS (who doesn't know the codes) can swap two preferences undetectably only if it knows which two positions on the code sheet they correspond to.

Proof. Starting assumptions:

1. Each ballot ID gets only one registered vote and ack code list.
2. The VS can derive no information from the Ack Codes.

In the worst case the client knows exactly what vote the voter wants to cast. We will assume this worst-case adversary and show that it can rearrange the preference acks correctly only if it knows the corresponding positions on the ballot.

Suppose the voter intends to cast vote V, a permutation of the candidate names. The cheating client swaps preferences i and j, which means swapping the i-th and j-th items in the list of Vote Codes (or candidate names), and submits the modified vote instead. It receives from the trustees a (cleartext) list of preference codes P arranged in the order the candidates appear on the code sheet. This list differs from what the voter is expecting only in that the codes for the i-th and j-th preferences must be swapped. Since the cheating VS knows which candidate names these correspond to, swapping them correctly implies knowing which (unordered) two locations on the code sheet they occupy. □

Proof of privacy. We wish to show that the protocol is receipt-free. Obviously only computational privacy is achieved, because both the vote codes and the ordered candidate names are shown, encrypted, on the bulletin board (that, is it does not achieve "everlasting privacy"). Equally obviously, our protocol is not receipt-free against a coercer who can observe both the (properly authenticated) code sheet and also the voter's communications with the Vote Server. We show that the protocol is receipt free if either the code sheet or the voter's communications with the VS remain secret.

We prove receipt freeness according to the definition of Moran and Naor [MN06]. The basic idea is that the voter, when requested to vote in a certain way by the coercer, should have a "coercion resistance" strategy which allows them to vote in the way they wished while providing a view to the coercer that is indistinguishable from obedience.

The protocol presented here goes just up to the point of having a list of encrypted votes on the bulletin board, and hence is not supposed to reveal any information, so we can prove directly that the coercion resistance strategy produces a coercer view indistinguishable from obedience. (Moran and Naor's definition must be more complex to deal with the case that the coercer learns useful information from the public tally itself, which isn't relevant here.)

Whether the subsequent tallying step preserves receipt freeness is a separate question, which is outside the scope of this paper. As described in the Introduction, expressive voting schemes can be vulnerable to pattern-matching ("Italian") attacks, and the choice of tallying protocol should protect against this.

The weakest point for maintaining voter privacy is in the printing and distribution of the code sheets. If we assume that that phase doesn't leak information, the distributed ballot construction implies that ρ and σ remain secret if:

1. At least one of the ballot construction authorities keeps their component permutation secret, and
2. At least one of the code sheet authorities keeps their component permutation secret, and
3. At least one of the output shuffle authorities keeps their component permutation secret, and
4. Fewer than a threshold number of decryption trustees collude.

We will refer to this list as the *Authority secrecy assumptions.*

The following theorem shows that an adversary who is ignorant of either the code sheet or the voter's communications with the VS learns nothing (more) about the vote from observing the bulletin board.

Theorem 1. *Given the authority secrecy assumptions above, Protocol B is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. The proof is in Appendix A. The main idea is that the voter can lie freely to the coercer about either ρ (if the coercer does not see the code sheet) or $\rho \circ \sigma$ (if the coercer does not observe communications with the VS) and hence produce a plausible claim to have cast any vote with the same number of candidates as the one they truly cast.

4 Protocol C: Two-Dimensional Tables

In this section each voter receives a two-dimensional table. Each row represents a candidate, each column a number. The numbers could be ranks for STV, Borda or IRV votes, as shown in Figure 6, or they could be scores for Range or Approval voting, as shown in Figure 7. Compared with Protocol B, this has more complicated vote casting but much simpler Ack checking.

For each candidate, the voter selects the code in the appropriate column, which the client then sends to the vote server. As in PGD 1.0, each voter receives a single ack, and the security of the scheme is dependent upon the secrecy of the Vote Codes and Ack code.

Candidate	1st	2nd	3rd	4th	5th
Red	37	90	12	08	72
Green	14	46	88	49	09
Chequered	95	10	21	83	20
Fuzzy	33	99	21	73	87
Cross	39	25	31	11	92

Ack Code: 8243
Ballot ID: 3884092844

Fig. 6. Example of Candidate and Preference Codes

Candidate	Approve	Disapprove
Red	37	72
Green	49	09
Chequered	95	21
Fuzzy	73	87
Cross	25	31

Ack Code: 8243
Ballot ID: 3884092844

Fig. 7. Example of Candidate and Approval/Disapproval Codes

4.1 Details of Ballot Construction, Ack Return and Tallying

Ballot construction Ballot construction and ack return are much simpler than the corresponding construction in Protocol C. On the code sheets and on the Bulletin Board, the candidates can remain in canonical order throughout. For each ballot, for each candidate, the authorities post to the Bulletin Board

- an encrypted Ack Code, and
- for each canonically ordered candidate, a list of encrypted (Vote Code, number) pairs in a secret, random order.

There are two slightly different versions depending on the kind of voting.

- For Range or Approval Voting, each vote code list is shuffled independently. This makes it impossible to tell how many candidates received the same number.
- for STV, IRV, or Borda, the same shuffle is applied to the code list of every candidate on the same ballot. This makes it easy to check the validity of each vote: anything with at most one PET match in each column is valid, because it has no repeated preferences.[3]

In either case, the table should be printed on the code sheet in canonical order, while the order(s) on the bulletin board remain secret.

[3] We are assuming here that votes are valid if they skip preferences, but not if they repeat a preference. If another rule were applied then an appropriate validity checking step would have to be added later.

Tallying. Again, Plaintext Equivalence Tests are used to match each Voter's encrypted Vote Codes with those on the Bulletin Board. When the submitted Vote Code matches $(VC_{ij}, number_j)$, this implies that candidate i (who is known from the canonical order) "gets" number $number_j$ (which is still encrypted). The correct interpretation of this depends on the voting scheme.

Approval or Range Voting, or Borda Count. For voting schemes that simply accumulate a score for each candidate, the tallying is simple. Using an encryption scheme with homomorphic addition, $number_j$ can simply be added to candidate i's total without being decrypted. Of course the scores have to be set up correctly in advance, with, for example, 1 and 0 for approval and disapproval respectively in AV, and $n - j$ for the j-th preference in Borda. This is straightforward.

Lists of preferences: STV or IRV. If the straightforward PET matching is done on the bulletin board, it reveals how many preferences each voter expressed. This protects against a cheating client or VS who submits only a subset of the complete preference list, but unfortunately it also violates each voter's privacy to some extent. In many instances, this would be a serious problem because it could allow a coercer to demand that a voter restrict the number of preferences they expressed. However, in the case where everyone must list the same number of preferences, all valid votes would be indistinguishable. This is fairly common in Australia, where often a permuation has to be complete to be valid, and it also occurs in the United States, where IRV with (typically) three compulsory preferences is sometimes used.

Tallying for IRV or STV is complex. So far, for each vote, we have produced a list of candidate names (in canonical order) with their corresponding (encrypted) rank. There are (at least) two possible options:

- Shuffle all the votes in this form and then decrypt them at the end. This would give the correct answer but possibly expose the voters to pattern-matching attacks (a.k.a. "Italian" Attacks) as described by Heather [Hea07] (and others).
- Apply a privacy-preserving STV or IRV tallying protocol [BMN+09], [Hea07], [WB09], possibly with a preprocessing step to deal with votes that skip some preferences.

4.2 Proofs of Correctness for Protocol C

This protocol is considerably simpler than Protocol B, which is reflected in the relative simplicity of the assumptions and proofs.

Basic proof of correctness. Again, when everyone follows the protocol the votes are cast and counted as the voter intended.

Lemma 2. *When all authorities follow Protocol C correctly, the vote registered corresponds to the rows and columns chosen on the code sheet.*

Proof. By construction. □

Proof of security against a cheating client. We would like to argue that a cheating client or VS cannot alter a vote undetectably, but it is important to clarify "undetectably." So far in this paper the voter has been able to detect vote manipulation by the absence of the expected ack code(s). The same will be true here, unless the cheating client or VS submits a subset of the $(VC_{ij}, number_j$ pairs, which is detectable only if the voter checks the bulletin board (presumably via an independent device). As explained above, this is not a problem in schemes in which the number of pairs is specified, such as AV with compulsory explicit approval or disapproval of each candidate, or IRV with exactly three preferences.

Claim. A cheating client or VS (who doesn't know the codes) cannot add valid (candidate, number) pairs.

Proof. Achieving a successful PET test requires either knowledge of the relevant code or collusion of a threshold number of decryption authorities. □

Claim. A cheating client or VS (who doesn't know the codes) cannot remove (candidate, number) pairs without this being observable on the bulletin board.

Proof. The bulletin board reveals how many pairs were registered for each vote. □

Proof of privacy. As in Section 3.4 we wish to show that the data on the bulletin board preserve (computational) vote privacy. Again we assume that that code sheet printing phase doesn't leak information, that at least one of the ballot construction authorities keeps their component permutations secret, and that fewer than a threshold number of decryption trustees collude.

Theorem 2. *Protocol C is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. Omitted, but very similar to that of Theorem 1.

5 Discussion

These protocols are designed so that even a completely corrupted device is unable to alter a voter's choices undetectably, *assuming that the voter follows the protocol perfectly.* Since the voter probably votes infrequently, and trusts the computer for voting instructions, the assumption of perfect voter behaviour might be easy to undermine. For example, a virus that presented an appealing window with instructions like, "please enter the candidate names and vote codes in the order they appear on your code sheet", (for Protocol C), or "please enter all the numbers in both tables", (for Protocol B) would probably succeed with many voters. Given that information it would then be able to cast whatever vote it chose and manipulate the returning acknowledgement codes correctly to avoid detection. Although attacks of this kind also work on other versions of code voting, our protocols are considerably more complicated and have more subtle privacy assumptions than the others, and hence are probably more vulnerable.

References

[Adi08] Adida, B.: Helios: Web-based Open-Audit Voting (2008)

[BMN+09] Benaloh, J., Moran, T., Naish, L., Ramchen, K., Teague, V.: Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. In: IEEE Transactions on Information Forensics and Security (2009)

[Cha01] Chaum, D.: SureVote: Technical Overview. In: Proceedings of the Workshop on Trustworthy Elections. In: WOTE 2001 (2001)

[CM05] Clarkson, M.R., Myers, A.C.: Coercion-Resistant Remote Voting using Decryption Mixes. In: Workshop on Frontiers in Electronic Elections. In: FEE 2005 (2005)

[FS01] Furukawa, J., Sako, K.: An Efficient Scheme for Proving a Shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001)

[Gro03] Groth, J.: A Verifiable Secret Shuffle of Homomorphic Encryptions. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 145–160. Springer, Heidelberg (2002), Later Version at www.brics.dk/~jg/JournalShuffle2.ps

[Hea07] Heather, J.A.: Implementing STV Securely in Prêt à Voter. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium, Venice, Italy, pp. 157–169 (July 2007)

[JCJ05] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant Electronic Elections. In: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, vol. 11 (2005)

[JJ00] Jakobsson, M., Juels, A.: Mix and Match: Secure Function Evaluation via Ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 162. Springer, Heidelberg (2000)

[JJR02] Jakobsson, M., Juels, A., Rivest, R.: Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In: USENIX Security Symposium, pp. 339–353 (2002)

[MN06] 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)

[Nef01] Andrew Neff, C.: A Verifiable Secret Shuffle and its Application to E-Voting. In: Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001), pp. 116–125. ACM Press, New York (2001)

[RT09] Ryan, P.Y.A., Teague, V.: Pretty Good Democracy. In: Proceedings of the 17th International Workshop on Security Protocols, Cambridge, UK, April 2009. LNCS. Springer, Heidelberg (2009)

[RT10] Ramchen, K., Teague, V.: Parallel shuffling and its application to Prêt à Voter. In: USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2010), Washington, DC (August 2010)

[TH08] Ting, P.-Y., Huang, X.-W.: Distributed Paillier plaintext equivalence test. International Journal of Network Security 6(3), 258–264 (2008)

[WB09] Wen, R., Buckland, R.: Minimum disclosure counting for the alternative vote. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) VOTE-ID 2009. LNCS, vol. 5767, pp. 122–140. Springer, Heidelberg (2009)

A Proof of Theorem 1

Theorem 1. *Given the authority secrecy assumptions defined in Section 3.4, the protocol is* receipt-free *[MN06] against a coercer who either*

1. *does not observe the order of candidates on the code sheet, or*
2. *does not observe the voter's communications with the Vote Server.*

Proof. This is really two separate results, one for each of conditions 1 and 2. The proofs are in the following two lemmas.

Lemma 3. *Suppose a voter wishes to cast vote v (a full permutation of the candidates) but the coercer instead demands π (also a full permutation). Suppose also that the coercer observes the voter's communication with the Vote Server, but no information about the order of candidates on the code sheet. Then there exists a coercion-resistance strategy CR for the voter such that the coercer's view when the voter obeys (i.e. votes π) is indistinguishable from the coercer's view when the voter disobeys (i.e. votes v) and runs CR.*

Proof. The coercer's view consists of any communication between herself and the voter before the vote, the complete list of messages between the voter and the VS, and the Bulletin Board transcript.

The voter's coercion resistance strategy is to vote v and tell the coercer that he voted π. Then the order of the preference acks, assuming vote π, is consistent with exactly one code sheet permutation ρ', which the voter can easily compute and claim to the coercer. Also the pattern of PET matches in the bulletin-board transcript is consistent with exactly one registration permutation σ'.

Distinguishing the claimed permutations ρ' and σ' from the true ρ and σ reduces to gaining information from the shuffles, which we are assuming to be impossible without sufficient authorities misbehaving. □

Lemma 4. *Suppose a voter wishes to cast vote v (a full permutation of the candidates) but the coercer instead demands π (also a full permutation). Suppose also that the coercer observes the voter's code sheet, but that the voter has an untappable channel to the VS. Then there exists a coercion-resistance strategy CR for the voter such that the coercer's view when the voter obeys (i.e. votes π) is indistinguishable from the coercer's view when the voter disobeys (i.e. votes v) and runs CR.*

Proof. The coercer's view consists of any communication between herself and the voter before the vote, all the information on the code sheet, and the Bulletin Board transcript.

The voter's coercion resistance strategy CR is to vote v and then tell the coercer that he voted π and received acks consistent with that.

The new information revealed on the bulletin board is the pattern of which elements of $(\sigma \circ \rho)(\mathcal{VC})$ match which elements of the vote $\mathcal{V}$. The coercer, having seen the code sheet, knows ρ. Given this information, the voter's claim to have voted π is consistent with exactly one registration permutation. (Specifically

$\sigma' = \sigma \circ \rho \circ \upsilon^{-1} \circ \pi \circ \rho^{-1}$, which the voter does not have to compute). Distinguishing voter obedience from the CR strategy hence reduces to distinguishing σ from σ' based on the shuffles in the construction phase, or distinguishing the true vote or preference return order from the suffled ones in the tally phase, which is infeasible without sufficient authorities misbehaving. □

Efficient Multi-dimensional Key Management in Broadcast Services*

Marina Blanton[1] and Keith B. Frikken[2]

[1] Department of Computer Science and Engineering, University of Notre Dame
mblanton@nd.edu
[2] Computer Science and Software Engineering, Miami University
frikkekb@muohio.edu

Abstract. The prevalent nature of Internet makes it a well suitable medium for many new types of services such as location-based services and streaming content. Subscribers to such services normally receive encrypted content and can obtain access to it if they possess the corresponding decryption key. Furthermore, in location-based services a subscription is normally granted to a geographic area specified by user-specific coordinates (x_1, x_2), (y_1, y_2) and custom time interval (t_1, t_2). Similarly, subscriptions to other services also involve multiple dimensions. The problem of key management is then to assign keys to each point on a D-dimensional grid and to subscribers in such a way as to permit all users to obtain access only to the resources in their subscriptions and minimize the associated overhead. In this work, we develop a novel key management scheme for multi-dimensional subscriptions that both outperforms existing solutions and supports a richer set of access privileges than existing schemes. Our scheme is provably secure under the Decision Linear Diffie-Hellman Assumption.

1 Introduction

The ubiquity of digital communication today allows it to be easily used for a variety of broadcast or streaming services. For instance, location-based services (LBS) have become widely spread and deployed; examples of such services include LOC-AID [2] and Garmin [1]. In these systems, a user typically can subscribe to a geo-spatial area for a specified duration of time and is able to query the system for spatial-temporal information such as traffic conditions, points of interest near a particular geographic location, or receive periodic updates such as the weather forecast. There is obviously a need to ensure that only legitimate subscribers can obtain access to the information within their subscription rights. Similarly, geographic information systems (GIS) collect and store large amounts of geo-spatial data such as satellite images, and there is a need to protect this data from unauthorized access. In particular, the importance of fine-grained access control mechanisms that would permit precise release of geo-spatial information was discussed in the NRC's IT roadmap to geo-spatial future [17] as a major challenge.

In such systems, a user typically subscribes for a fee to an area bounded by coordinates (x_1, y_1) and (x_2, y_2) for a specific time interval $[t_1, t_2]$. A user is then allowed to

* Portions of this work were sponsored by AFOSR grant AFOSR-FA9550-09-1-0223 and NSF grant CNS-0915843.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 424–440, 2010.

access the resource or broadcast associated with the coordinate (x, y) at time t if and only if $x_1 \leq x \leq x_2$, $y_1 \leq y \leq y_2$, and $t_1 \leq t \leq t_2$. The space is modeled as a two-dimensional grid of size $T_1 \times T_2$ cells. Time is also partitioned into small slots and becomes the third dimension of the grid. More generally, a grid of any dimension can be specified and used.

A typical solution in broadcast services is to encrypt content and broadcast the encrypted content. The access control is then enforced by distributing certain secret keys to subscribers and ensuring that the decryption key for a broadcasted resource is available only to the parties who are authorized to access the resource. This setup has several advantages including the ability to outsource the storage and distribution of encrypted content to a third-party provider, which is important in GIS systems or location-based services that deal with large volumes of data. Another advantage is that users can also remain anonymous while accessing an (authorized) resource.

Access control enforcement is therefore implemented via key management, which is a well studied topic. With this enforcement mechanism, the service provider assigns keys to the resources (in our context, all resources associated with a single position in a multi-dimensional space are assigned the same key). When a user subscribes to a set of resources in the system (i.e., a sub-grid in multi-dimensional space), she obtains secret information that will allow her to obtain access to the subscribed resources. This secret information either can directly contain decryption keys for all resources to which the user is entitled to have access or can permit derivation of all such keys. The challenge in designing such solutions is in ensuring that the access control policy is enforced while achieving the best possible performance. Therefore, the overhead of such solutions is measured in terms of the size of user secret keys, work necessary to derive a key, amount of additional information the service provider must maintain, etc.

While key management for dynamic groups or hierarchical systems (such as RBAC) is well-studied (see Section 2), solutions for geo-spatial systems and higher dimensions appeared only in the recent years [5, 20, 24]. In general, solutions that can be applied to a space of an arbitrary dimension D are desirable as they will allow any service or system to be used within this framework. As an example of an application that benefits from a key management scheme that supports any number of dimensions, consider streaming television. When a user subscribes to television, the access policy could be specified over a potentially large number of dimensions such as (i) content being accessed (at the level of station, show, or episode), (ii) quality of channel (e.g, data quality and commercial content), (iii) time of access (e.g., hour, day, month, and year), (iv) location of user (e.g., state and zipcode). Unlike subscribing to a rectangular region in location-based services, this subscription may not consist of a range over the dimensions; that is, a user could subscribe to multiple stations. Furthermore, more flexible subscriptions than continuous range in all dimensions can be desirable for location-based services as well: for example, there might be need to exclude certain geographic regions such as military installations or critical infrastructures from subscriptions.

Our contributions. We propose a novel key management scheme for multi-dimensional grids with attractive performance characteristics. In particular, it has the following properties, which none of the existing schemes can simultaneously achieve:

- Users can subscribe to an arbitrary set of points or intervals in each dimension, i.e., the subscription region does not have to be contiguous.
- The amount of user secret storage and the amount of work a user must perform to derive a key do not exponentially depend on the number of dimensions D; both storage and work are linear in D. This means that the scheme can be efficiently realized for applications where the number of dimensions is high without burdening the user; this substantially improves the performance compared to other schemes.
- Users do not need to access any external (publicly available) data for the purposes of aiding key derivation; broadcast content is all a user receives.
- The service provider needs to store only a constant amount of information associated with the scheme.

A more detailed comparison with prior literature is provided in the next section. We are able to achieve this performance by issuing sub-keys to users for each dimension separately and using a mechanism for tying the sub-keys of each user together to be able to maintain security (i.e., to achieve resilience against collusion). Our scheme enjoys provable security under the standard Decision Linear assumption.

2 Related Work

Related work on key management can be divided into two lines of research that go under the names of key management for access hierarchies and group key management. We give a brief overview of each of them next.

In hierarchical access control schemes, all users are divided into a set of access classes, which are organized in a hierarchy. Resources associated with each access class are encrypted with the corresponding encryption key. A user with access to a specific class is allowed to access resources at her own class and all descendant classes in the hierarchy. In order to lower overhead of such schemes, public information that helps in the key derivation process is used. Users from different classes use different parts of the public information data structure to derive necessary keys efficiently. Performance of such schemes is measured in terms of the number of keys a user stores, the size of public information, work needed to derive a key, and overhead associated with user joins and leaves.

The formal definitions of security in this context were put forward by Atallah et al. [7,3] (the overall literature is very extensive, see, e.g., [7] for an overview), and, in particular, that work defined the notion of *key recovery* and *key indistinguishability* for key management schemes. Consequently, the work of Ateniese et al. [8] extended the definitions to time-based key management for a hierarchy of access classes, where time is partitioned into small slots and a user obtains access to a certain class in the hierarchy (and consequently to all descendant classes) for a certain contiguous interval of time which may differ for each user.[1] The authors also showed that the security notions in the presence of *static* and *adaptive* adversaries are (polynomial time) equivalent for time-based schemes, which means that showing security against static adversaries is sufficient for such schemes. In this extended framework, key derivation is now performed

[1] This problem was studied prior to Ateniese et al. [8] (see, e.g., [21,15,22]), but earlier schemes lack formal proofs of security and some of them are known to have security flaws.

for the purposes of hierarchical access control and time-based (i.e., one-dimensional) access control. Other work on time-based key management for user hierarchies includes [6,12] that improve performance of the initial solutions in [4], lowering the overhead associated with the schemes. These latter publications give a mechanism for performing time-based key management that can be combined with any suitable hierarchical key management scheme, i.e., the mechanisms for achieving two goals are decoupled. More recently, techniques for higher dimensions were proposed as well. In particular, [5] gives an efficient solution for geo-spatial (i.e., two-dimensional) access control, which is further improved and extended to a higher number of dimensions in [24].

Literature on group key management is also concerned with the problem of key assignment to users and resources. No relationship between user classes or groups is assumed (i.e., key management is performed for each group independently), and instead the need to perform key derivation comes only from the dynamic nature of groups, where a user can join and leave a group at any time. The key difference between this line of work and work on hierarchical key management is (i) absence of relationship between the groups, and (ii) inability to use public storage. In particular, the use of public information greatly aids the performance of hierarchical schemes (including extensions to multiple dimensions) resulting in low overheads in terms of the number of user keys and key derivation time. In group key management protocols, it is assumed that some content is broadcast to the users, and the users can derive the necessary decryption key (using the broadcast and stored keys) if and only if they are authorized to access the content.

This problem is well studied with many solutions available (see, e.g., [14,13,23,18]). Srivatsa et al. [20] were first to extend the framework to multiple dimensions, to enable such schemes to be used with location-based services such as spatial-temporal authorizations and subscription services of any dimensionality in general. This work provides a solution which is significantly more efficient than the straightforward use of prior group key management protocols for a single group, and supports access to a contiguous interval in each dimension. Its user overhead (i.e., the number of keys and key derivation time), however, is exponential in the number of dimensions, which makes it less suitable for applications where the number of dimensions is large. Our solution simultaneously removes exponential dependence on the number of dimensions (all overhead is at most linear in the number of dimensions) and improves expressiveness of the scheme by permitting user access to any subset of slots in each dimension.

We summarize performance of other solutions and our scheme in Table 1. In the table, D denotes the number of dimensions, X_i denotes the set of user subscription units in dimension i, where $|X_i|$ is the size of the set, and T_i denotes the maximum number of units in dimension i. The expressiveness column indicates whether a scheme supports only contiguous intervals in each dimension (in which case X_i can be specified as a range $[a, b]$ for $a \leq b$) or any subset of units in each dimension from the available range $[1, T_i]$. The communication overhead column indicates the amount of data that must be made available to permit key derivation for *all* authorized users when encrypted content for to a single point in the D-dimensional space is broadcast. That is, the solution of Yuan-Atallah uses a public data structure of the specified size that allows any user to efficiently derive decryption keys for any subscribed point in the D-dimensional space,

Table 1. Comparison with prior work

Scheme	User's keys	Key derivation	Comm. overhead	Expressiveness
Yuan-Atallah [24]	$O(1)$	$O(1)$	$O(\prod_{i=1}^{D} T_i \cdot (\log^* \log^* (\prod_{i=1}^{D} T_i))^D)$	contiguous interval
Srivatsa et al. [20]	$O(\frac{1}{D}(2^{D-1} \cdot \sum_{i=1}^{D} \log \lvert X_i \rvert))$	$O(\frac{1}{D}(2^D \cdot \sum_{i=1}^{D} \log \lvert X_i \rvert))$	–	contiguous interval
Our scheme	$O(\sum_{i=1}^{D} \lvert X_i \rvert)$	$O(D)$	$O(D)$	any subset

Table 2. Performance of multi-dimensional query over encrypted data schemes

Scheme	Public-key size	Enc. cost	Ciphertext size	Dec. key size	Dec. cost
Boneh-Waters [11]	$O(D \cdot T)$	$O(D \cdot T)$	$O(D \cdot T)$	$O(D)$	$O(D)$
Shi et al. [19]	$O(D \log T)$	$O(D \log T)$	$O(D \log T)$	$O(D \log T)$	$O((\log T)^D)$

but the solution is not well suited for uni-directional broadcast services since different users will need to use different parts of the public data structure. In our case, each encrypted transmission can be easily prepended with D data items which will permit all authorized users to obtain access to the content.

Another direction of research related to this work is queries over encrypted data. In particular, we mention the work of Shi et al. [19] on multi-dimensional range queries and the work of Boneh and Waters [11] that permits multi-dimensional subset and range queries. Since these schemes do not use key derivation (and therefore have different characteristics), but could potentially be used in our context, we provide their performance separately in Table 2. This table uses T as the number of points in each dimension, i.e., $T = T_1 = \ldots = T_D$. It is clear that in our context transmitting ciphertext of size $O(D \cdot T)$ (as in [11]) or having decryption cost of $O((\log T)^D)$ operations (as in [19]) is not acceptable. The higher computational cost in these schemes is dictated by stronger privacy properties (i.e., the inability to determine attributes associated with a ciphertext), which is not needed in our context.

Finally, we mention attribute-based encryption (ABE) as a potential realization of the functionality we seek. With traditional ABE, we will be able to form a ciphertext with D attributes which corresponds to a cell in the multi-dimensional grid. A client who wishes to subscribe to items $X = X_1 \times \cdots \times X_D$ will then have to store $\prod_{i=1}^{D} |X_i|$ keys (i.e, a key per cell of its subscription). Communication cost is $O(D)$ and decryption cost is also $O(D)$. If we employ a hierarchical ABE, the efficiency can potentially be improved through derivation, but the costs are still significant. If, for example, we use Boneh et al. [9] hierarchical identity based encryption (HIBE), which has performance characteristics among the best known for HIBE schemes in that the ciphertext size is independent of the number of elements T_i in each dimension, permitting a user to subscribe to only continuous intervals X_i in each dimension already leads to $O(\prod_{i=1}^{D} (\log |T_i| \log |X_i|))$ private key storage, and supporting arbitrary X_i's results in $O(\prod_{i=1}^{D} (|X_i| \log T_i))$ key material.[2] In such schemes, the size of each decryption key depends on the height of

[2] This can be somewhat decreased with longer ciphertexts (e.g., of size $O(\prod_{i=1}^{D} \log T_i)$).

the hierarchy, and $O(\prod_{i=1}^{D} \log |X_i|)$ ($O(\prod_{i=1}^{D} |X_i|)$) keys are needed to represent X in the case of contiguous intervals (resp., any subsets).

3 Model Description and Definitions

System Model. A service provider has a resource, which is associated with a point in D-dimensional space. We denote the number of items/intervals in jth dimension, for $j = 1, \ldots, D$, by T_j. We will assume that the units are numbered 1 through T_j, i.e., lie in the interval $[1, T_j]$. Then access to a resource with coordinates $(i_1, i_2, \ldots, i_D)$ in D-dimensional space will be secured using a cryptographic key $k_{i_1,\ldots,i_D}$, such that knowledge of the key will imply access to the resource.

Now suppose that a user $\mathcal{U}$ is authorized to have access to units $X = X_1 \times X_2 \times \cdots \times X_D$, where each X_j is an arbitrary subset of T_j units in dimension j (i.e., unlike the prior work, the intervals in each dimension do not have to be contiguous). With such access rights, $\mathcal{U}$ should receive or should be able to compute the keys $k_{i_1,\ldots,i_D}$, where $i_j \in X_j$ for each j. We denote the private information that $\mathcal{U}$ receives by S_X. Obviously, storing $\prod_{j=1}^{D} |X_j|$ keys at the user end is not always practical, and significantly more efficient solutions are possible. A *multi-dimensional key assignment (MDKA) scheme* assigns keys to the units in a multi-dimensional space and users, so that proper access control is enforced in a correct and efficient manner. Such key generation is assumed to be performed by the resource owner, but once a user is issued the keys, there is no interaction with other entities. More formally, we define a MDKA scheme as follows:

Definition 1. *Let $T = T_1 \times T_2 \times \cdots \times T_D$ define a D-dimensional space. A multi-dimensional key assignment scheme consists of algorithms* (Setup, Assign, Derive) *s.t.:*

Setup *is a probabilistic algorithm, which, on input a security parameter 1^κ and D-dimensional grid T, outputs (i) a key $k_{i_1,\ldots,i_D}$ for any $(i_1, \ldots, i_D) \in T$; (ii) secret information* sec *associated with the system; and (iii) public information* pub. *Let* $(K, \mathsf{sec}, \mathsf{pub})$ *denote the output of this algorithm, where K is the set of all keys.*

Assign *is a probabilistic algorithm, which, given specification of access rights $X = X_1 \times \cdots \times X_D \subseteq T$ and secret information* sec, *outputs private information S_X.*

Derive *is a deterministic algorithm, which, on input access rights $X = X_1 \times \cdots \times X_D$, a point $(i_1, \ldots, i_D) \in T$, private information S_X, and public information* pub, *outputs key $k_{i_1,\ldots,i_D}$ if $(i_1, \ldots, i_D) \in X$ and a special failure symbol $\perp$ otherwise. The correctness requirement is such that, for each set of access rights $X \subseteq T$, each point $(i_1, \ldots, i_D) \in X$, each private information S_X, each key $k_{i_1,\ldots,i_D} \in K$, and each public information* pub *that* Setup$(1^\kappa, T)$ *and* Assign(X, sec) *can output,* $\Pr[\mathsf{Derive}(X, (i_1, \ldots, i_D), S_X, \mathsf{pub}) = k_{i_1,\ldots,i_D}] = 1$.

Note that we provide a general specification of such a scheme that can work under different assumptions. As mentioned above, in our solution access to the entire public information is not needed, and instead the key derivation algorithm needs access only to the public information for one point in the D-dimensional space, the key of which is being derived. We will denote public information for point $(i_1, \ldots, i_D)$ as $\mathsf{pub}_{i_1,\ldots,i_D}$, and this is what will be needed for Derive. Also, it is possible that in some schemes all

values that the Assign algorithm outputs (i.e., S_X for every $X \subseteq T$) can be produced at the system initialization time (in which case Assign is deterministic), but it is still desired to separate it from Setup.

Security Model. In prior literature on key management schemes, two security goals have been defined [3]: security against *key recovery*, in which an adversary is unable to compute a key to which it should not have access, and security with respect to *key indistinguishability*, which means that an adversary is unable to learn any information about a key to which it should not have access and thus cannot distinguish it from a random string of the same length. The latter is obviously a stronger notion of security. Also, the literature on one-dimensional (i.e., time-based) KA schemes (e.g., [8]) distinguishes between security in the presence of *static adversaries* and security in the presence of *adaptive adversaries*. Then a static adversary is given a specific unit (i.e., a D-dimensional point in our context) to attack and obtains access to all other keys that do not allow it to access the challenge point. An adaptive adversary, on the other hand, obtains oracle access to the Assign algorithm, can query user keys of its choice, choose a challenge unit, and eventually output its response to the challenge.

In [8] it was shown that the security of key assignment schemes against a static adversary is (polynomial-time) equivalent to the security against an adaptive adversary for both security goals (key recovery and key indistinguishability), which on the surface appears to enable us to consider only static adversaries. There is, however, a difference between prior and our specifications of the key assignment algorithm in that we allow it to be probabilistic. This, in particular, means that two users with exactly the same privileges can obtain different secret information that allows them to access the same resources. From the security point of view, this difference is crucial enough that equivalence between security notions in presence of static and adaptive adversaries no longer holds. That is, a static adversary obtains secret information corresponding to a minimal number of users that ensures coverage of keys for all resources except its challenge, while an adaptive adversary can query any number of keys for possibly the same or overlapping access rights. Thus, there can be schemes that are secure if adversary obtains only one set of key material, but insecure when an adversary has access to multiple versions of the secret information for related access rights. Therefore, in the rest of this work we will concentrate on adaptive adversaries only.

Throughout this work, we use notation $a \stackrel{R}{\leftarrow} A$ to mean that a is chosen uniformly at random from the set A. A function $\epsilon(\kappa)$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ, $\epsilon(\kappa) < \frac{1}{p(\kappa)}$.

Let $\mathcal{A}$ denote an adaptive adversary attacking the security of a MDKA scheme. $\mathcal{A}$ obtains all public information and is given oracle access to Assign algorithm. In the first stage of the attack, $\mathcal{A}$ can query Assign(sec, $\cdot$) and outputs its choice of challenge point $(i_1, \ldots, i_D)$. In the second stage of the attack, $\mathcal{A}$ can further query its oracle and produce its response. Let the set Q denote all queries that $\mathcal{A}$ makes to Assign. $\mathcal{A}$ can query its oracle for any access rights X of its choice with the restriction that the challenge point cannot be contained in X. This in particular does not prevent the adversary from constructing queries that contain all of the i_j's from the challenge across several queries (e.g., querying X_1 and X_2 such that $(i_1, \ldots, i_{D-1}, i'_D) \in X_1$ and $(i'_1, i_2, \ldots, i_D) \in X_2$ is allowed if $i'_D \neq i_D$ and $i'_1 \neq i_1$), which means that the solution must be collusion

resistant. Because the notion of key indistinguishability is strictly stronger than security against key recovery, and it is a widely accepted security model, we concentrate on security with respect to key indistinguishability only. Then after the first stage, $\mathcal{A}$ is given either the real key corresponding to the challenge point or a random value and must correctly guess which one was used. We require that the success probability of $\mathcal{A}$ is negligible in κ. The key indistinguishability experiment is given below.

Experiment $\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T)$
$(K, \mathsf{sec}, \mathsf{pub}) \leftarrow \mathsf{Setup}(1^\kappa, T)$
$((i_1, \ldots, i_D), state) \leftarrow \mathcal{A}_1^{\mathsf{Assign}(\mathsf{sec},\cdot)}(1^\kappa, T, \mathsf{pub})$
$b \overset{R}{\leftarrow} \{0,1\}$
if $b = 0$ then $\alpha \overset{R}{\leftarrow} \{0,1\}^{|k_{i_1,\ldots,i_D}|}$ else $\alpha \leftarrow k_{i_1,\ldots,i_D}$
$b' \leftarrow \mathcal{A}_2^{\mathsf{Assign}(\mathsf{sec},\cdot)}(1^\kappa, T, \mathsf{pub}, (i_1, \ldots, i_D), state, \alpha)$
if $\forall X \in Q, (i_1, \ldots, i_D) \notin X$ and $b = b'$ then return 1 else return 0

Definition 2. *Let $T = T_1 \times \cdots \times T_D$ be a D-dimensional grid of distinct units and $\mathsf{MDKA} = (\mathsf{Setup}, \mathsf{Assign}, \mathsf{Derive})$ be a multi-dimensional key assignment scheme for T and a security parameter κ. Then MDKA is secure with respect to key indistinguishability in the presence of an adaptive adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with oracle access to $\mathsf{Assign}(\mathsf{sec}, \cdot)$ in both stages of the attack if it satisfies the following properties:*

- Completeness: *A user, who is given private information S_X for access rights to $X = X_1 \times \cdots \times X_D \subseteq T$, is able to compute the access key $k_{i_1,\ldots,i_D}$ for each $(i_1, \ldots, i_D) \in X$ using only her knowledge of S_X and public information pub with probability 1.*
- Soundness: *If we let the experiment $\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}$ be specified as above, the advantage of $\mathcal{A}$ is defined as:*

$$\mathsf{Adv}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = \left| \Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - \tfrac{1}{2} \right|$$

We say that MDKA is sound with respect to key indistinguishability if for each $(i_1, \ldots, i_D) \in T$, for all sufficiently large κ, and every positive polynomial $p(\cdot)$, $\mathsf{Adv}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) < 1/p(\kappa)$ for each polynomial-time adversary $\mathcal{A}$.

In addition to the security requirements, an efficient MDKA scheme is evaluated by the following criteria:

- The size of the secret data a user must store;
- The amount of computation for generation of an access key for the target resource;
- The amount of information the service provider must maintain.

Number-Theoretic Preliminaries. The notation $\mathbb{G} = \langle g \rangle$ denotes that g generates the group $\mathbb{G}$. Our solution uses groups with pairings, and we review concepts underlying such groups next.

Definition 3 (Bilinear map). *A map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map if the following conditions hold:*

- *(Efficient) $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same prime order q, and there exists an efficient algorithm for computing e.*

- *(Bilinear) For all $g \in \mathbb{G}$, and $a, b \in \mathbb{Z}_q$, $e(g^a, g^b) = e(g, g)^{ab}$.*
- *(Non-degenerate) If g generates $\mathbb{G}$, then $e(g, g)$ generates $\mathbb{G}_T$.*

Throughout this work, we assume that there is a setup algorithm Set that, on input a security parameter 1^κ, outputs the setup for group $\mathbb{G} = \langle g \rangle$ of prime order q that have a bilinear map e, and $h = e(g, g)$ generates $\mathbb{G}_T$ (which also has order q). That is, $(q, \mathbb{G}, \mathbb{G}_T, e, g, h) \leftarrow Set(1^\kappa)$.

The security of our scheme relies on Decision Linear Diffie-Hellman assumption (DLIN). It was introduced in [10] and is currently widely used; we review it next.

Definition 4 (DLIN). *The Decision Linear problem is, given generator g of $\mathbb{G}$, g^a, g^b, g^{ac}, g^{bd}, and Z, where $a, b, c, d \in \mathbb{Z}_q$ and $Z \in \mathbb{G}$, output 1 if $Z = g^{c+d}$ and 0 otherwise. We say that the Decision Linear assumption holds in $\mathbb{G}$ if any probabilistic polynomial time (in κ) adversary $\mathcal{A}$ has at most negligible probability in solving the Decision Linear problem. More precisely,*

$$\begin{aligned}\mathsf{Adv}_{\mathsf{DLIN},\mathcal{A}}(1^\kappa) = |\Pr[\mathcal{A}(\mathbb{G}, q, g, g^a, g^b, g^{ac}, g^{bd}, g^{c+d}) = 1] - \\ -\Pr[\mathcal{A}(\mathbb{G}, q, g, g^a, g^b, g^{ac}, g^{bd}, g^R) = 1]| \leq \epsilon(\kappa)\end{aligned}$$

for some negligible function $\epsilon(\cdot)$.

4 Description of the Scheme

Overview of the Scheme. Our solution was inspired by work on multi-dimensional range queries [19], where a secret was used to tie multiple dimensions to achieve collusion resilience in a different context. That high-level idea led us to develop a new scheme which is more balanced than all existing key management solutions and improves their performance. Furthermore, our approach supports a richer set of access rights than prior key management work.

At a high level, in our scheme each point j in the ith dimension (for $1 \leq i \leq D$ and $1 \leq j \leq T_i$) is assigned a secret $s_{i,j}$. There is also a system-wide secret w. When a user subscribes to the resources in $X = X_1 \times \cdots \times X_D$, she is issued keys, or private information S_X, that are a function of both $s_{i,j}$'s in her access rights X and w. In particular, w is first split into D random shares w_i such that $\sum_{i=1}^{D} w_i = w$. Then for each point j in the ith dimension of user's subscription (i.e., $j \in X_i$ for $1 \leq i \leq D$), the user obtains a key $k_{i,j}$ computed u sing $s_{i,j}$ and w_i.

When a user receives a broadcast and wants to compute a key associated with a point $(i_1, \ldots, i_D)$, she will be able to derive the encryption key for that point only if $i_j \in X_j$ for each $1 \leq j \leq D$. To compute the encryption key, the user retrieves the key k_{j,i_j} from S_X corresponding to each coordinate i_j of the point $(i_1, \ldots, i_D)$ in dimension j. The point $(i_1, \ldots, i_D)$ will also have publicly available information consisting of $D+1$ values (which is included in the broadcast). The user combines elements of that public data with her keys k_{j,i_j} dimension-wise to compute the necessary encryption key.

Detailed Description. We now present a complete description of the scheme. Security analysis is given in Section 5, and performance analysis in Section 7.

Setup : Run $(q, \mathbb{G}, \mathbb{G}_T, e, g, h) \leftarrow Set(1^\kappa)$ to generate a group with pairings. Choose the master secret $w \overset{R}{\leftarrow} \mathbb{Z}_q$. For each dimension i, for each unit j in dimension i, choose its secret $s_{i,j} \overset{R}{\leftarrow} \mathbb{Z}_q$. For each D-dimensional point with coordinates $(i_1, i_2, \ldots, i_D) \in T$, generate public information by choosing $r_{(i_1,\ldots,i_D)} \overset{R}{\leftarrow} \mathbb{Z}_q$ and setting $\mathsf{pub}_{i_1,\ldots,i_D} = (g^{r_{(i_1,\ldots,i_D)}}, g^{r_{(i_1,\ldots,i_D)} \cdot s_{1,i_1}}, \ldots, g^{r_{(i_1,\ldots,i_D)} \cdot s_{D,i_D}})$. The key for point $(i_1, i_2, \ldots, i_D)$ is $e(g,g)^{r_{(i_1,\ldots,i_D)} w}$ for the value of $r_{(i_1,\ldots,i_D)}$ used in producing the public data.

Assign : Suppose user $\mathcal{U}$ is entitled to access privileges to a D-dimensional structure $X = X_1 \times X_2 \times \cdots \times X_D$, where for each dimension i, $X_i \subseteq 2^{T_i}$ (i.e., X_i can be an arbitrary subset of T_i items). First, randomly choose D random values $w_1, \ldots, w_D$ from $\mathbb{Z}_q$ subject to the constraint $\sum_{i=1}^{D} w_i \bmod q = w$. For each $i = 1, \ldots, D$, for each $j \in X_i$, randomly choose $t \overset{R}{\leftarrow} \mathbb{Z}_q$ and add $k_{i,j} = (g^t, g^{w_i + t \cdot s_{i,j}})$ to the user's S_X.

KeyDer : A user who is entitled to access a D-dimensional point with coordinates $(i_1, \ldots, i_D)$ first retrieves the key associated with each coordinate i_j from her private information S_X. Let $k_{j,i_j} = (g^{t_j}, g^{w_j + t_j \cdot s_{j,i_j}})$ denote such a key. Next, the user retrieves the public information associated with the point $\mathsf{pub}_{i_1,\ldots,i_D} = (g^{r_{(i_1,\ldots,i_D)}}, g^{r_{(i_1,\ldots,i_D)} \cdot s_{1,i_1}}, \ldots, g^{r_{(i_1,\ldots,i_D)} \cdot s_{D,i_D}})$ from the broadcasted content and derives the encryption key as:

$$\prod_{j=1}^{D} e(k_{j,i_j}[2], \mathsf{pub}_{i_1,\ldots,i_D}[1]) e((k_{j,i_j}[1])^{-1}, \mathsf{pub}_{i_1,\ldots,i_D}[j+1])$$

Here $u[i]$ denotes the ith value of tuple u.

It is clear from the above that a system consisting of $\prod_{i=1}^{D} T_i$ points in the D-dimensional space will support $\prod_{i=1}^{D} 2^{T_i}$ types of access privileges. While there is public information associated with each point in T, in Section 6 we show that in practice the service owner needs to have only $O(1)$ storage to maintain the operation of the system.

5 Security Analysis

In this section we show that our scheme satisfies both the completeness and soundness requirements. Efficiency of our solution is evaluated in Section 7.

Theorem 1. *The multi-dimensional key assignment scheme* $\mathsf{MKDA} = (\mathsf{Setup}, \mathsf{Assign}, \mathsf{KeyDir})$ *presented above is complete.*

It is not difficult to show that the result of computation performed at key derivation time for a grid point always equals to the encryption key generated for that point at the setup time. We omit the details due to space considerations.

Theorem 2. *Assuming that the Decision Linear assumption holds, the multi-dimensional key assignment scheme* $\mathsf{MDKA} = (\mathsf{Setup}, \mathsf{Assign}, \mathsf{KeyDir})$ *presented above achieves key indistinguishability in the presence of adaptive adversaries.*

Proof. Suppose there is a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathrm{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) > 1/p(\kappa)$ for some polynomial p. We will show that there exists a PPT adversary $\mathcal{B}$ with black box access to $\mathcal{A}$ that solves the decision linear problem with non-negligible probability.

According to the definition, $\mathcal{B}$ is given $(q, \mathbb{G}, \mathbb{G}_T, e, g, h), g^a$, g^b, g^{ac}, g^{bd}, and Z, and need to decide whether $Z = g^{c+d}$. In our case, $\mathcal{B}$ will be given Z of the form g^{c+d} or g^{R+d} for some $R \in \mathbb{Z}_q$ (where each element of the group can be written as g^{R+d} for some R) and needs to correctly decide whether it was g^{c+d}.

Our algorithm $\mathcal{B}$ first chooses a random point $(\hat{i}_1, \ldots, \hat{i}_D) \in T$. Essentially, $\mathcal{B}$ is guessing the point which $\mathcal{A}$ will use as its challenge. $\mathcal{B}$ then interacts with $\mathcal{A}$ as follows:

Setup: $\mathcal{B}$ performs system setup as follows:

1. It sets the parameters using $(q, \mathbb{G}, \mathbb{G}_T, e, g, h)$.
2. To generate secret information for the cells of the grid, for each dimension $j \in [1, D]$ and each element $t \in [1, T_j]$ in dimension j, $\mathcal{B}$ chooses a random element $q_{j,t} \stackrel{R}{\leftarrow} \mathbb{Z}_q$. $\mathcal{B}$ stores these q values. To finish the setup of secret information sec, we implicitly set $s_{j,t} = q_{j,t}$ when $\hat{i}_j = t$ (i.e., it is part of the challenge) and to $s_{j,t} = q_{j,t} + d$ otherwise. Note that in the latter case $\mathcal{B}$ does not know $s_{j,t}$.
3. To generate the public information for each point $(i_1, \ldots, i_D)$, there are two cases:
   - $(i_1, \ldots, i_D) = (\hat{i}_1, \ldots, \hat{i}_D)$: In this case, we use g^a from $\mathcal{B}$'s challenge to set $\mathsf{pub}_{i_1,\ldots,i_D} = (g^a, (g^a)^{q_{1,i_1}}, \ldots, (g^a)^{q_{D,i_D}})$. This means that $r_{(i_1,\ldots,i_D)} = a$. Notice that $\mathsf{pub}_{i_1,\ldots,i_D} = (g^{r_{(i_1,\ldots,i_D)}}, g^{r_{(i_1,\ldots,i_D)} \cdot s_{1,i_1}}, \ldots, g^{r_{(i_1,\ldots,i_D)} \cdot s_{D,i_D}})$, which is the same as in the real protocol.
   - $(i_1, \ldots, i_D) \neq (\hat{i}_1, \ldots, \hat{i}_D)$: Choose random $u_{(i_1,\ldots,i_D)} \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and use g^b, g^{bd} from $\mathcal{B}$'s challenge to set $\mathsf{pub}_{i_1,\ldots,i_D} = ((g^b)^{u_{(i_1,\ldots,i_D)}}, R_1, \ldots, R_D)$, where

$$R_j = \begin{cases} (g^b)^{u_{(i_1,\ldots,i_D)} \cdot q_{j,i_j}} & \text{if } i_j = \hat{i}_j \\ (g^{bd})^{u_{(i_1,\ldots,i_D)}} (g^b)^{u_{(i_1,\ldots,i_D)} \cdot q_{j,i_j}} & \text{otherwise.} \end{cases}$$

     This means that $\mathcal{B}$ sets $r_{(i_1,\ldots,i_D)} = b \cdot u_{(i_1,\ldots,i_D)}$ and $R_j = g^{r_{(i_1,\ldots,i_D)} \cdot s_{j,i_j}}$, where $s_{j,i_j} = q_{j,i_j}$ if $i_j = \hat{i_j}$ and $s_{j,i_j} = q_{j,i_j} + d$ otherwise (which is consistent with the way secret information was setup). The above guarantees that these tuples are distributed identically to when $\mathcal{A}$ engages in the real protocol.

Assign **queries**: When $\mathcal{A}$ asks for a query $X = X_1 \times X_2 \times \cdots \times X_D$, $\mathcal{B}$ responds as:

1. If $\hat{i}_j \in X_j$ for each $1 \leq j \leq D$ (i.e., X contains the challenge), $\mathcal{B}$ outputs $FAIL$.
2. Otherwise, $\mathcal{B}$ chooses m to be an index such that $\hat{i}_m \notin X_m$ (there must be at least one such index). Next, $\mathcal{B}$ chooses and stores random values $w_1, \ldots, w_{m-1}, w_{m+1}, \ldots, w_D$ from $\mathbb{Z}_q$. Let $w' = \sum_{i \in [1,D], i \neq m} w_i$. $\mathcal{B}$ creates the key material by setting the key information for dimension j and position $t \in X_j$ as follows:
   - If $j = m$, then choose $\ell \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and use g^b, g^{bd}, Z from $\mathcal{B}$'s challenge to compute and return $k_{j,t} = ((g^b)^\ell g,\ Z(g^{bd})^\ell g^{-w'} (g^b)^{\ell q_{j,t}} g^{q_{j,t}}) =$
     $= (g^{b\ell+1}, Z g^{bd\ell - w' + b\ell q_{j,t} + q_{j,t}}) = (g^{b\ell+1}, Z g^{b\ell(d+q_{j,t}) + q_{j,t} - w'})$.
     Note that, because in this case $s_{j,t} = q_{j,t} + d$, when $Z = g^{c+d}$, this tuple is $(g^{b\ell+1}, g^{c - w' + (b\ell+1)(d+q_{j,t})}) = (g^{b\ell+1}, g^{c - w' + (b\ell+1)s_{j,t}})$. Otherwise, when $Z = g^{R+d}$, it is $(g^{b\ell+1}, g^{R - w' + (b\ell+1)s_{j,t}})$.
   - If $j \neq m$ and $\hat{i}_j \neq t$, first choose $\ell \stackrel{R}{\leftarrow} \mathbb{Z}_q$, then compute and return $k_{j,t} = ((g^b)^\ell, g^{w_j} (g^b)^{\ell q_{j,t}} (g^{bd})^\ell) = (g^{b\ell}, g^{w_j + b\ell(q_{j,t}+d)}) = (g^{b\ell}, g^{w_j + b\ell s_{j,t}})$, where $s_{j,t} = q_{j,t} + d$ as required.

– If $j \neq m$ and $\hat{i}_j = t$, choose $\ell \xleftarrow{R} \mathbb{Z}_q$ and return $k_{j,t} = (g^{\ell}, g^{w_j} g^{\ell q_{j,t}}) = (g^{\ell}, g^{w_j + \ell s_{j,t}})$, where now $s_{j,t} = q_{j,t}$ as previously set.

Notice that the keys are consistent with those generated by a real challenger. In particular, for each dimension j and each element $t \in X_j$, $k_{j,t}$ is of the form $(g^r, g^{w_i + r s_{j,t}})$, where r takes the value of $b\ell + 1$, $b\ell$, or ℓ depending on the case, and $w = \sum_{i=1}^{D} w_i$. Note that we have implicitly defined w using randomly chosen w_i for all but one dimension and Z. More specifically, if $Z = g^{c+d}$, then $w = c$ and if $Z = g^{R+d}$, then $w = R$. In all cases $\mathcal{B}$ does not know w. Furthermore, this key assignment implies that applying the key derivation procedure to the key for $(\hat{i}_1, \ldots, \hat{i}_D)$ and $\mathsf{pub}_{\hat{i}_1,\ldots,\hat{i}_D}$ results in the encryption key $e(g,g)^{ac}$ when $Z = g^{c+d}$ and $e(g,g)^{aR}$ when $Z = g^{R+d}$.

Challenge: When $\mathcal{A}$ issues a challenge for $(\bar{i}_1, \ldots, \bar{i}_D)$, if $(\bar{i}_1, \ldots, \bar{i}_D) \neq (\hat{i}_1, \ldots, \hat{i}_D)$, $\mathcal{B}$ outputs $FAIL$. Otherwise, $\mathcal{B}$ returns $e(g^{ac}, g) = e(g,g)^{ac}$. If $Z = g^{c+d}$, then this is the correct key, but if $Z = g^{R+d}$, then this is a independent key from the real one specified by the above parameters.

More Assign **queries**: Same as before.

Output: Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{B}$ returns b'.

Suppose $\mathcal{B}$ does not output $FAIL$. Then if $Z = g^{c+d}$, $\mathcal{A}$ has been given the correct key for the challenge point, and if $Z = g^{R+d}$, $\mathcal{A}$ is given a random key. This means that $\mathcal{A}$'s view is the same as in $\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^{\kappa}, T)$. Furthermore, because $\mathcal{B}$ simply outputs what $\mathcal{A}$'s outputs, if $\mathcal{A}$ can distinguish keys with non-negligible probability, $\mathcal{B}$ will also be able to solve the decision linear problem with non-negligible probability. We next give a more detailed analysis to tie the advantage of $\mathcal{A}$ in experiment $\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^{\kappa}, T)$ with the advantage of $\mathcal{B}$ in solving the decision linear problem.

First observe that:

$$\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^{\kappa}, T) = 1] = \Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^{\kappa}, T) = 1 \wedge \mathsf{ProperQueries}],$$

where the event $\mathsf{ProperQueries}$ means that $\mathcal{A}$ did not query X containing the challenge point during any of its calls to Assign. This equality is true because the experiment always outputs 0 when $\mathcal{A}$ violates this querying constraint. We also use $\mathsf{GoodGuess}$ to denote the event when $\mathcal{A}$ guesses the bit b in the experiment correctly (i.e., when $b = b'$). This in particular implies that $\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^{\kappa}, T) = 1] = \Pr[\mathsf{GoodGuess} \wedge \mathsf{ProperQueries}]$. Next, we have:

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{DLIN},\mathcal{B}}(1^{\kappa}) &= \big|\Pr[\mathcal{B}(\mathbb{G}, q, g, g^a, g^b, g^{ac}, g^{bd}, g^{c+d}) = 1] - && (1)\\
&\quad - \Pr[\mathcal{B}(\mathbb{G}, q, g, g^a, g^b, g^{ac}, g^{bd}, g^{R}) = 1]\big| && (2)\\
&= \big|\Pr[\mathsf{GoodGuess} \wedge \overline{\mathsf{Fail}}] - \Pr[\overline{\mathsf{GoodGuess}} \wedge \overline{\mathsf{Fail}}]\big| && (3)
\end{aligned}$$

where Fail denotes the event that $\mathcal{B}$ outputs $FAIL$ as a result of interaction with $\mathcal{A}$. From the description of the interaction, we know that $\mathcal{B}$ outputs $FAIL$ when (i) $\mathcal{B}$ does not guess the challenge correctly or (ii) when $\mathcal{A}$ attempts to query a key for privileges that contain the point chosen to be the challenge. We formalize this as $\Pr[\overline{\mathsf{Fail}}] = \Pr[\overline{\mathsf{WrongChallenge}} \wedge \mathsf{ProperQueries}]$. Substituting this into equation (3), we obtain:

$$\mathsf{Adv}_{\mathsf{DLIN},\mathcal{B}}(1^{\kappa}) = \big|\Pr[\mathsf{GoodGuess} \wedge \overline{\mathsf{WrongChallenge}} \wedge \mathsf{ProperQueries}] -$$

$$
\begin{aligned}
&\quad -\Pr[\overline{\mathsf{GoodGuess}} \wedge \overline{\mathsf{WrongChallenge}} \wedge \mathsf{ProperQueries}]\big| \\
&= \big|\Pr[\overline{\mathsf{WrongChallenge}} \mid \mathsf{GoodGuess} \wedge \mathsf{ProperQueries}] \times \\
&\quad \times \Pr[\mathsf{GoodGuess} \wedge \mathsf{ProperQueries}] - \\
&\quad -\Pr[\overline{\mathsf{WrongChallenge}} \mid \overline{\mathsf{GoodGuess}} \wedge \mathsf{ProperQueries}] \times \\
&\quad \times \Pr[\overline{\mathsf{GoodGuess}} \wedge \mathsf{ProperQueries}]\big|
\end{aligned}
$$

Now because $\mathcal{B}$ chooses its challenge point uniformly at random regardless of $\mathcal{A}$'s behavior, we rewrite the above as:

$$
\begin{aligned}
\mathsf{Adv}_{\mathsf{DLIN},\mathcal{B}}(1^\kappa) &= \big|\Pr[\overline{\mathsf{WrongChallenge}}]\Pr[\mathsf{GoodGuess} \wedge \mathsf{ProperQueries}] - \\
&\quad -\Pr[\overline{\mathsf{WrongChallenge}}]\Pr[\overline{\mathsf{GoodGuess}} \wedge \mathsf{ProperQueries}]\big| \\
&= \left|\frac{1}{T}\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - \right. \\
&\quad \left. -\frac{1}{T}\left(1 - \Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - \Pr[\overline{\mathsf{ProperQueries}}]\right)\right|
\end{aligned}
$$

since $\Pr[\overline{\mathsf{WrongChallenge}}] = \frac{1}{T}$ and

$$
\begin{aligned}
1 &= \Pr[\mathsf{ProperQueries}] + \Pr[\overline{\mathsf{ProperQueries}}] = \\
&= \Pr[\mathsf{GoodGuess} \wedge \mathsf{ProperQueries}] + \Pr[\overline{\mathsf{GoodGuess}} \wedge \mathsf{ProperQueries}] + \Pr[\overline{\mathsf{ProperQueries}}] = \\
&= \Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] + \Pr[\overline{\mathsf{GoodGuess}} \wedge \mathsf{ProperQueries}] + \Pr[\overline{\mathsf{ProperQueries}}]
\end{aligned}
$$

Finally, we obtain

$$
\begin{aligned}
\mathsf{Adv}_{\mathsf{DLIN},\mathcal{B}}(1^\kappa) &= \frac{1}{T}\left|2\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - 1 + \Pr[\overline{\mathsf{ProperQueries}}]\right| \\
&\geq \frac{1}{T}\left|2\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - 1\right| \\
&= \frac{2}{T}\left|\Pr[\mathbf{Exp}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T) = 1] - \frac{1}{2}\right| = \frac{2}{T}\mathsf{Adv}^{\text{key-ind}}_{\mathsf{MDKA},\mathcal{A}}(1^\kappa, T)
\end{aligned}
$$

This means that if $\mathcal{A}$ succeeds in breaking the security of the MDKA scheme with non-negligible probability δ, $\mathcal{B}$ succeeds in breaking the decisional linear problem with non-negligible probability which is at least $2\delta/T$. □

The above reduction relates the success probabilities of algorithms $\mathcal{A}$ and $\mathcal{B}$ using a factor of $2/T$. This means that it is desirable to set the security parameter of the scheme to be $\kappa + \log(T) - 1$, where κ is the security parameter necessary to ensure the difficulty of solving the decision linear problem. This is likely to increase κ by a few dozen bits (see, e.g., Section 7 for an example application).

Note that our result is consistent with best practices in the literature (e.g., [8]), where security against adaptive adversaries is desired (i.e., the simulator is forced to guess the challenge point). Furthermore, related work on range queries and IBE-based schemes have security proofs in a weaker, so-called selective ID model, where the adversary commits to the challenge point prior to system setup. Under those circumstances, we would achieve a tight reduction with no efficiency loss.

6 Extensions

Reducing Public Storage. In the system the way it was described, the public storage at the server is $O(TD)$. As T could be large, this amount of storage may be problematic. Furthermore, the setup algorithm requires this many modular exponentiations, which is also a bottleneck. We modify the scheme in order to reduce the storage to $O(1)$. The crux of this idea is that since we send the public information to the user on demand, we do not need to have all of the information at once. Furthermore, this information can be derived as it is needed. More specifically, let $F_1 : [1, D] \times T_{\max} \times \{0,1\}^\kappa \to \mathbb{Z}_q$ and $F_2 : T \times \{0,1\}^\kappa \to \mathbb{Z}_q$ be pseudorandom functions, where $T_{\max}$ is the maximum of $T_1, \ldots, T_D$. We make the following changes to the MDKA scheme:

Setup: In this case the public information is now just $(q, \mathbb{G}, \mathbb{G}_T, e, g, h)$. We still chooses the master secret $w \xleftarrow{R} \mathbb{Z}_q$ along with two PRF keys $k_1 \xleftarrow{R} \{0,1\}^\kappa$ and $k_2 \xleftarrow{R} \{0,1\}^\kappa$. The secret information is then (w, k_1, k_2). Implicitly we are setting the secret parameters to be $s_{i,j} = F_1(i, j, k_1)$ and $r_{(i_1,\ldots,i_D)} = F_2((i_1, \ldots, i_D), k_2)$.

Assign and KeyDer: When we need to compute a user's key or public information, we simply compute its values using F_1 and F_2.

One-Time Keys. One concern with key management solutions is that users can distribute access keys to unauthorized parties. This would allow anyone to access the content for free. There are two types of such revelations possible for our system: (i) the user can publish its private S_X or (ii) she can derive the key for a specific cell $(i_1, \ldots, i_D)$ and publish it (i.e., publish $e(g,g)^{r_{(i_1,\ldots,i_D)} \cdot w}$). Of these two types of distributions, the latter is worse, because it reveals no information about the offending party except the ability to access point $(i_1, \ldots, i_D)$, whereas the first type reveals significantly more information about that party, i.e, the complete specification of the access rights. Fortunately, the latter, more damaging attack can be mitigated as follows: The value $r_{(i_1,\ldots,i_D)}$ can be changed each time that cell is used. That is, since we are sending $\mathsf{pub}_{i_1,\ldots,i_D}$ to the users along with the ciphertext, the protocol can simply choose a new values of $r_{(i_1,\ldots,i_D)}$ each time. Thus, the key $e(g,g)^{r_{(i_1,\ldots,i_D)} \cdot w}$ is useful only for the current message, and will not be useful for other messages.

7 Performance

The complexity of our MDKA scheme is as follows:

1. *Size of* pub: This is $O(1)$ as the only values that need to be stored are $(q, \mathbb{G}, \mathbb{G}_T, e, g, h)$.
2. *Size of* sec: This has size $O(1)$ as all that is stored is (w, k_1, k_2).
3. *Size of user key:* A user with access to $X_1 \times \cdots \times X_D$ obtains $\sum_{i=1}^{D} |X_i|$ pairs of values as its private keys and thus maintains $O(\sum_{i=1}^{D} |X_i|)$ values.
4. *Size of an encryption:* To be able to decrypt, the user needs to have the public information associated with the access point, which has size $O(D)$.
5. *Cost to assign key:* This requires $O(\sum_{i=1}^{D} |X_i|)$ work.

6. *Cost to send broadcast to user:* The user will need to receive the public information, which will require $O(D)$ operations from the sender.
7. *Cost to derive key:* This requires $O(D)$ operations.

To demonstrate the applicability of our approach to practical systems, we consider a content streaming application. We give a small example and discuss the performance of our scheme. Suppose that a content streaming system has the following dimensions:

(i) the specific content (i.e., show) being accessed inside the range $[1, 2^{16}]$,
(ii) quality of programming inside the range $[1, 8]$,
(iii) time of access inside the range $[1, 7760]$ (i.e., once for every hour of a year),
(iv) x-coordinate of location of access ranging in the range $[1, 1024]$, and
(v) y-coordinate of location of access ranging in the range $[1, 1024]$.

The motivation for the locations is that the service provider desires to only let the user see the content in certain locations for DRM purposes and location-based content (such as local weather forecast). Consider a user that subscribes to 100 shows, two quality markers (one poor quality for a mobile device and one high quality for home), for access from 6-10PM daily, in a 10 by 10 region. Using our solution, this user would need to store $100 + 2 + 1460 + 10 + 10 = 1582$ keys, which is clearly practical. Furthermore, to derive a specific key, a user would have to perform 10 pairing operations. According to [16], each of these operations take about 11ms on a 1 GHz Pentium III, and thus key derivation would require about 110ms, which is also clearly practical.

We now consider the shortcoming of the solution presented in [20] for this particular problem. First, this scheme provides a weaker notion of security (key recovery instead of key indistinguishability). The main problem, however, is that this scheme does not support arbitrary intervals. This means that the content and time blocks must be separated and multiple sets of keys must be given to the user. That is, for each of the 100 shows and for each day, the user must have $2^3(\frac{3+10+10+13}{4}) = 72$ keys, and therefore store $72 \cdot 365 \cdot 100$ or about 2.6 million keys. Clearly, as the subscription becomes more complex, this will result in the user storing too many keys.

8 Conclusions

In this work we treat the problem of key assignment in multi-dimensional space for subscription-based broadcast and location-based services. In particular, each dimension corresponds to an attribute (such as latitude, longitude, time, or any other attribute) which is partitioned into a number of units, comprising a D-dimensional grid. All resources associated with a point in this grid are assigned a cryptographic key and distributed in encrypted form. A subscriber joining the system obtains access to certain resources specified as a subset of points in each dimension. She is issued key material that allows her to derive cryptographic keys for all D-dimensional points in the subscription privileges. We give a new scheme for key assignment and management with characteristics that favorably compare with existing schemes. In particular, the user acquires overhead only linear in the number of dimensions, is not required to access external data in addition to broadcast content, and can be issued more flexible access privileges than in other schemes. Our solution is provably secure under the standard Decision Linear assumption.

References

1. Garmin (2009), http://www.garmin.com
2. LOC-AID (2009), http://www.loc-aid.net
3. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM Transactions on Information and System Security (TISSEC) 12(3), 1–43 (2009)
4. Atallah, M., Blanton, M., Frikken, K.: Key management for non-tree access hierarchies. In: ACM Symposium on Access Control Models and Technologies, pp. 11–18 (2006)
5. Atallah, M., Blanton, M., Frikken, K.: Efficient techniques for realizing geo-spatial access control. In: ACM Symposium on Information, Computer and Communications Security (ASIACCS 2007), pp. 82–92 (2007)
6. Atallah, M., Blanton, M., Frikken, K.: Incorporating temporal capabilities in existing key management schemes. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 515–530. Springer, Heidelberg (2007)
7. Atallah, M., Frikken, K., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: ACM CCS, pp. 190–201 (2005)
8. Ateniese, G., De Santis, A., Ferrara, A., Masucci, B.: Provably-secure time-bound hierarchical key assignment schemes. In: ACM CCS, pp. 288–297 (2006)
9. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
10. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
11. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
12. De Santis, A., Ferrara, A., Masucci, B.: New constructions for provably-secure time-bound hierarchical key assignment schemes. In: ACM Symposium on Access Control Models and Technologies (SACMAT 2007), pp. 133–138 (2007)
13. Harney, H., Muckenhirn, C.: Group key management protocol (GKMP) architecture. IETF RFC 2094 (1997), http://www.rfc-archive.org/getrfc.php?rfc=2094
14. Harney, H., Muckenhirn, C.: Group key management protocol (GKMP) specification. IETF RFC (2093) (1997), http://www.rfc-archive.org/getrfc.php?rfc=2093
15. Huang, H., Chang, C.: A new cryptographic key assignment scheme with time-constraint access control in a hierarchy. Computer Standards & Interfaces 26, 159–166 (2004)
16. Lynn, B.: The pairing-based cryptography (pbc) library, http://crypto.stanford.edu/pbc
17. Patterson, C., Muntz, R., Pancake, C.: Challenges in location-aware computing. IEEE Pervasive Computing 2(2), 80–89 (2003)
18. Perrig, A., Song, D., Tygar, J.: ELK: A new protocol for efficient large group key distribution. In: IEEE Symposium on Security and Privacy, pp. 247–262 (2001)
19. Shi, E., Bethencourt, J., Chan, H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: IEEE Security and Privacy Symposium, pp. 350–364 (2007)
20. Srivatsa, M., Iyengar, A., Yin, J., Liu, L.: Access control in location-based broadcast services. In: IEEE INFOCOM, pp. 256–260 (2008)
21. Tzeng, W.: A time-bound cryptographic key assignment scheme for access control in a hierarchy. IEEE Transactions on Knowledge and Data Engineering 14(1), 182–188 (2002)

22. Wang, S.-Y., Laih, C.-S.: Merging: An efficient solution for a time-bound hierarchical key assignment scheme. IEEE Transactions on Dependable and Secure Computing 3(1), 91–100 (2006)
23. Wong, C., Gouda, M., Lam, S.: Secure group communication using key graphs. In: ACM SIGCOMM, pp. 68–79 (1998)
24. Yuan, H., Atallah, M.: Efficient and secure distribution of massive geo-spatial data. In: ACM GIS, pp. 440–443 (2009)

Caught in the Maze of Security Standards

Jan Meier and Dieter Gollmann

Security in Distributed Applications
Hamburg University of Technology
Hamburg, Germany
j.meier@tu-harburg.de, diego@tu-harburg.de

Abstract. We analyze the interactions between several national and international standards relevant for eCard applications, noting deficiencies in those standards, or at least deficiencies in the documentation of their dependencies. We show that smart card protocols are currently specified in a way that standard compliant protocol implementations may be vulnerable to attacks. We further show that attempts to upgrade security by increasing the length of cryptographic keys may fail when message formats in protocols are not re-examined at the same time. We argue that the entities responsible for accrediting smart card based applications thus require security expertise beyond the knowledge encoded in security standards and that a purely compliance based certification of eCard applications is insufficient.

1 Introduction

The German government advocates the use of smart cards in its eCard strategy in order to increase efficiency in administration (eGovernment), the health sector (eHealth), and commercial transactions (eBusiness) [1]. Central to this strategy are authentication, qualified electronic signatures, and the use of smart card based tokens. For example, a smart card based identity card could be used to conclude legally binding contracts over the internet. Privacy issues are addressed in eCard applications but remain a key concern for citizens. This is especially true for eCard applications in the health sector. Discussions about such applications, therefore, take place in a politically charged atmosphere and *certificational weaknesses* have to be taken very seriously indeed[1].

The specifications of the IT infrastructure for eCard applications build on various standards, which are also relevant when constructing a security case for the application. A security architecture has to relate the overall security goals of the application to the specific security services provided by the individual components and security protocols deployed. In eCard applications, authentication is one goal. This goal can be reached in multiple ways. The architecture typically references standards to specify the exact method. Those standards define protocols and cryptographic parameters. Having multiple standards frequently causes cross-dependencies, gaps, or conflicts between requirements. These issues could

[1] A certificational weakness is a vulnerability where a real exploitation is impractical.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 441–454, 2010.

be systematized by specifying a hierarchy between standards. However, current standards do not offer strategies assisting designers to identify and resolve these issues.

We will deal with *intentional* underspecification in smart card standards. Often multiple ways exist to achieve the same behavior. In such situations, standards should not constrain possible implementations. Implementation details are therefore left open. We will present cases where protocol analysis at the level of the standard specification would flag a failure, notwithstanding the fact that simple defenses are available at the implementation level. Protocol analysis therefore has to build on more than the given protocol specification and needs access to information about the implementation. This is the dual of the familiar issue where an abstract protocol has been verified to be secure, but an implementation introduces vulnerabilities. Here, the abstract protocol would be insecure and the implementation plugs the gaps.

1.1 Smart Card Details

The way smart cards work implies some specific characteristics of protocol execution. A notable difference compared to devices with a user interface is the inability of smart cards to initiate an action; they can only react. Hence, a card reader has to send commands to the smart card and wait for a response. Protocols can be built from such request/response pairs. Commands can also manipulate the internal state of a smart card. Based on the internal state, the card may reject commands sent by the reader. These features play an important role in our following observations. We will focus on authentication between smart card and reader and do not consider the IT infrastructure behind the reader.

Six smart card commands specified in ISO/IEC 7816-4 [2] suffice to cover all smart card related authentication protocols. The `Manage Security Environment` (`MSE`) command informs the smart card about the cryptographic algorithms and keys to be used in the subsequent command sequence. Keys are selected via key references. Only keys stored on the card can be selected; key references not associated with keys on the card will cause a smart card to cancel command execution. Smart cards store this information given in the `Manage Security Environment` command data in their internal memory. Processing of subsequent commands typically depends on information related to previously sent `Manage Security Environment` commands. The `Get Challenge` (`GetChall`) command requests challenges from the smart card. The latest requested challenge is stored in the smart card's internal memory. When other commands are processed, smart cards can make use of this challenge. The `Read Binary` (`ReadB`) command requests the content of a file given in the command data. Identifiers like the smart card serial number are stored in a file and therefore can be obtained with this command. In situations where a smart card has to authenticate itself, the communication partner sends an `Internal Authenticate` command. The command data is used to generate an authentication token. When communication partners are required to authenticate themselves, the smart card expects an `External Authenticate` (`ExAuth`) command containing an authentication

token. Finally, the `Mutual Authenticate` (`MutAuth`) command combines `Internal Authenticate` and `External Authenticate` into one command.

Smart card commands manipulate the smart card's internal state but also rely heavily on this internal state when certain commands are processed. The cryptographic algorithm specified in the `Manage Security Environment` command determines the kind of protocol the card is running with a communication partner. When access to data on the card is requested, the decision algorithm may refer to this internal state. For example, a cryptographic key may only be used when the successful completion of an earlier protocol run is recorded in the internal state.

The paper is organized as follows. Section 2 describes a set of security standards relevant for eCard applications. Section 3 contains a case study of authentication protocols for use in smart card applications. We use two protocols to demonstrate how protocols are specified for smart card use. We describe potential protocol vulnerabilities of smart card implementations in Section 4. Section 5 summarizes our observations on security standards. Section 6 comments on the certification of eCard applications. Section 7 concludes the paper. Although the application scenario we use as example has a German context, the standardization issues observed are not limited to eCard applications or to Germany.

2 Standards Relevant for eCard Applications

We will paint our observations on authentication in eCard applications with a broad brush, tracking the path from the high level security requirements of an application to precisely formulated requirements on an authentication protocol that could be used in a formal verification step.

Top level specifications of security requirements either take for granted that the meaning of authentication is well defined, or give familiar explanations like "verifying a claimed identity". Closer scrutiny may reveal that it is actually not required to verify the identity but some attribute of an entity. (This attribute might be verified on the basis of a secret key shared by all legitimate entities or on the basis of an attribute certificate.) Hence, there may be a first gap between the requirements stated and the requirements that should actually be met.

Formal analysis of authentication protocols often makes use of *correspondence properties* [3]. Such correspondence properties try to capture the meaning of entity authentication as specified in [4] by insisting that a party successfully completes a protocol run only if the corresponding party has the same view of certain aspects of the protocol run. This view may just capture the identity of the corresponding party, but it might also include session keys, session identifiers, or the other party's view on whom it had communicated with. An attack is then a protocol run violating the chosen correspondence properties. The attacks we discuss later violate such correspondence properties but are not necessarily attacks against a high level authentication goal. Still, it would be desirable for a security architecture to explicitly bridge the gap between the different levels of authentication specifications, so as to give guidance on the selection or certification of protocols suitable for a given application.

The following five standards or standard related documents are relevant for eCard projects. It is important to note that each document resides on its own abstraction layer and addresses different issues.

2.1 Cryptographic Protocols

The ISO/IEC 9798 series describes entity authentication protocols for symmetric key cryptography [5] and asymmetric key cryptography [6]. This standard series is application and technology independent. Protocols are described on an abstract level. Detailed descriptions of the actions communication partners have to perform are given. Message formats, cryptographic algorithms, and key lengths are left unspecified. Hence, these standards do not define direct blueprints for implementation. This standard series is referenced in a standard for smart cards as secure signature creation devices [7] and in the eCard guideline documentation we discuss next.

For eCard projects, the German Federal Office for Information Security maintains recommendations on the strength of cryptographic algorithms and key lengths [8]. In essence, life-spans for encryption mechanisms are given. For example, the use of two key triple DES (2KTDES) is prohibited in eCard projects after 2009 while three key triple DES (3KTDES) may be used until the end of 2013. Migration strategies or ways to adapt existing protocol to eCard requirements are not given. This document does not assist application designers in adapting protocols from ISO/IEC 9798 to smart cards.

2.2 Smart Card Standards

Interactions between smart cards and their environment must be standardized. This includes electrical interfaces, position of connectors, dimensions and commands. All of this is specified in the ISO/IEC 7816 series. We focus on security and on commands for smart cards as covered in [2]. This part (ISO/IEC 7816-4) specifies byte sequences to invoke commands, transmission of command data (parameters), and the status flags a command could possibly return; command processing is left unspecified. This standard is specifically tied to smart card technology but is independent of the applications realized with the help of smart cards.

Standard CWA14890-1 [7] is maintained by the European Committee for Standardization. It builds on [2] as it uses smart card commands when specifying protocols. Although [7] is not intended for a specific application it refers specifically to secure signature creation devices. Thus, security is a topic of the standard, which is primarily an interoperability standard. Section 3.1 will discuss an authentication protocol based on symmetric key cryptography in detail. The standard neither includes a security argument—formal or informal—nor does it give a reference to such a security argument.

The protocol specification for mutual authentication defines 2KTDES as the encryption algorithm. In contrast to [8], no statements on the security or life-span of these protocols are made. Application designers face the problem that

2KTDES is prohibited in [8] but a protocol using 2KTDES is specified in [7]. In [8], there is no guidance on resolving this inconsistency. Although the CWA14890-1 standard is security related, it does not discuss in general the properties smart card commands must have to ensure secure protocol execution.

Lastly, Common Criteria protection profiles are relevant for eCard applications. They indicate which tests eCard components have to pass in order to get certified. However, designers cannot extract requirements that, say, smart card commands have to fulfill from the standards above. Instead application designers must trust smart card producers that cards are suitable for the intended task.

We now have five documents that designers have to consider when building eCard applications. Between these documents, there is no clear hierarchy controlling document relationships. For example, it is unclear which document manages the length of challenges used in security protocols. Furthermore, security properties of smart card commands and their processing is left to the application designer although this topic is important and requires security expertise. In the following section, we will discuss the implications of this missing hierarchy.

3 Case Study: Specifications for Smart Card Based Authentication Protocols

We first detail a protocol specification for mutual authentication given in CWA 14890-1 [7]. The second protocol specification starts from the mutual authentication protocol specification in order to build a unilateral two pass authentication protocol for smart cards.

3.1 Authentication with Key Establishment

The following authenticated key establishment protocol has two goals [7]. First, it provides evidence that smart card and reader know previously shared secret keys and thus are legitimate communication partners. Second, smart card and reader agree on two session keys to protect subsequent communications. In smart card applications, one key is used for encryption/decryption operations while the second key is used to compute message authentication codes (MAC). CWA 14890-1 specifies 2KTDES in cipher block chaining mode with fixed initialization vector 0 as encryption method. The length of challenges is specified as 64 bits. Additionally, the procedure to establish session keys is defined and the length of the key derivation data is set to 256 bits. Consequently, designers have all information needed to integrate the protocol into an application.

The communicating parties are assumed to share two long term symmetric keys used for encryption, decryption and integrity protection. Such symmetric keys are stored on smart card and reader at production time, or could be established in a protocol using asymmetric key cryptography. The communication is depicted in Figure 1. The reader starts the protocol with a `Manage Security Environment` command informing the smart card which keys will be used to

protect subsequent messages. From the command data, the smart card determines that the subsequent protocol establishes two session keys and mutually authenticates reader and smart card. Next, the reader fetches the serial number of the smart card (SN.c) with a `Read Binary` command and stores it for later use. Then, the reader requests a 64 bits (8 bytes) random number from the smart card with a `Get Challenge` command. Upon receiving `Get Challenge`, the smart card generates a 64 bits random number (Rand.c) used as nonce. The smart card stores Rand.c in its internal memory and replies with Rand.c to the reader.

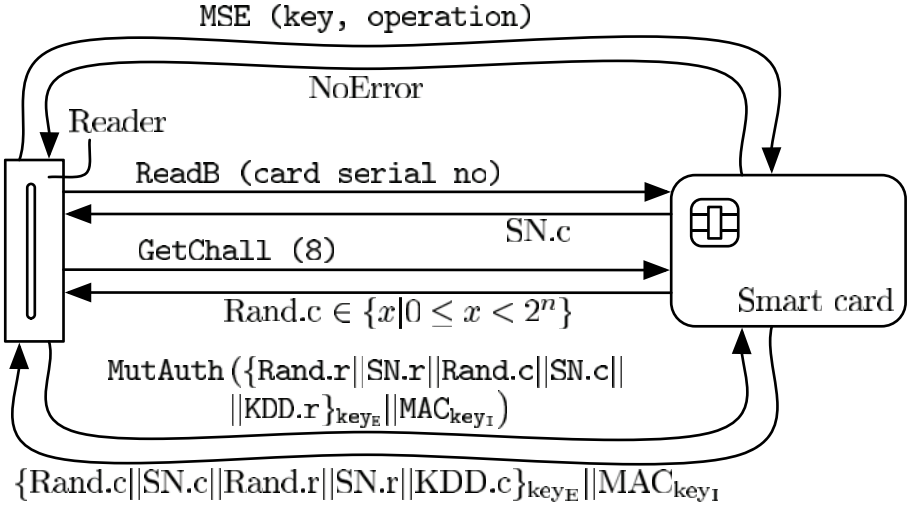


Fig. 1. A Key Establishment Protocol as Described in CWA 14890-1

Once the reader receives Rand.c, it generates its own 64 bits random number (Rand.r). Further 256 random bits (KDD.r) are selected for use as key derivation data. The reader stores KDD.r in its internal memory. Next, the reader generates the command data for a `MutAuth` command: it concatenates Rand.r, SN.r, Rand.c, SN.c, and KDD.r, and encrypts the resulting string under the encryption key selected in the initial `Manage Security Environment` command. Last, the reader generates a MAC of the encrypted data using the selected integrity key. Then, the reader sends the `Mutual Authentication` command with the command data just generated to the smart card.

Upon receiving the `Mutual Authentication` command, the smart card first checks the integrity of the message. If it can confirm command data integrity, the command data are decrypted. Next, the smart card checks whether the second random number (Rand.c) has the same value as the random number stored in the card's internal memory, and whether the second serial number (SN.c) equals its own serial number. If these two tests are successful, the card stores the key derivation data (KDD.r) in its internal memory. Now, the smart card

selects its own 256 bits key derivation data (KDD.c) and stores it in its internal memory, concatenates Rand.c, SN.c, Rand.r, SN.r, and KDD.r, encrypts the concatenated data using the previously selected encryption key, and calculates a MAC over the encrypted data using the integrity key. The smart card then sends the encrypted data and MAC back to the reader. After verifying the MAC value, the reader decrypts the response and checks whether the second random number (Rand.r) equals the one stored in its internal memory and whether the second serial number (SN.r) matches its own.

Once smart card and reader have stored both their own and received key derivation data, they can generate the session keys. CWA 14890-1 specifies 2KTDES as the algorithm to be used for encryption, decryption, and integrity protection. Therefore, four 8 byte keys have to be generated. First, both protocol participants XOR the key derivation data (KDD.r $\oplus$ KDD.c) resulting in a value KDD.rc. Then, two 32 bit counters are appended to KDD.rc, resulting in KDD.rc1 and KDD.rc2. The value of the first counter is 1, the value of the second is 2. Each protocol participant then calculates hash values of KDD.rc1 and KDD.rc2. CWA 14890-1 stipulates SHA-1 as hash algorithm. The first 8 bytes of SHA-1(KDD.rc1) are used as the first encryption key, the second 8 bytes are used for the second encryption key, while the last four bytes of the hash value are not used. The value of SHA-1(KDD.rc2) is used similarly. The first 16 bytes are used for both integrity keys and the last four bytes are not used. These keys are stored in the internal state of both communication partners. From now on, these keys can be used to secure further communications.

3.2 Challenge-Response Authentication Protocol

Often, smart cards do not need session keys to perform application oriented tasks. However, smart cards typically require devices to authenticate themselves, i.e. communicating devices prove knowledge of certain cryptographic keys. A successful authentication is stored in the smart cards internal state as long as the smart card is connected to the reader or the smart card explicitly removes this information from its state. Thus, these authentication protocols can be simple challenge-response protocols. None of the standards detailed in Section 2 give a smart card implementation of a simple challenge-response authentication protocol. Thus, eCard application designers have to create their own implementation. Naturally, designers may turn to the protocol from the previous section—the only protocol from eCard relevant standards designed for smart cards.

Removing the `Read Binary` command pair and replacing the `Mutual Authentication` with `External Authenticate`, we get a challenge-response authentication protocol, depicted in Figure 2.

First, the reader informs the smart card which protocol it wants to run, sending a `Manage Security Environment` command that states an operation the smart card has to perform later in the protocol. The `Manage Security Environment` command also selects a key required to check the encrypted nonce later in the protocol. Operation and key are stored in the smart card's internal state. Next, the reader sends a `Get Challenge` command requesting a random

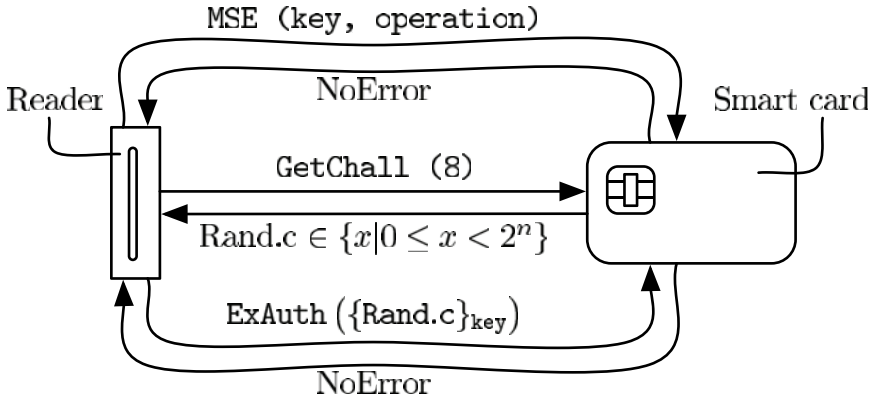


Fig. 2. Smart Card Adaption of the Simple Challenge-Response Protocol from [5]

value that serves as a nonce. The `Get Challenge` command includes command data specifying the length of the challenge in bytes. Upon receiving the command, the smart card generates a random number (Rand.c) of the requested length, stores it in its internal state, and responds with Rand.c. The reader now encrypts Rand.c with the shared secret key it had previously selected with the `Manage Security Environment` command. The encrypted challenge is then used as command data in an `External Authentication` command sent to the smart card. On receiving this command, the smart card encrypts the challenge stored in its internal memory with the secret key specified in the initial `Manage Security Environment` command. After encrypting the nonce stored in its internal memory, the smart card compares the result with the command data of the `External Authenticate` command. If the two values match, the smart card accepts that the reader knows the shared secret key. In the domain of eCard applications, knowing the shared secret key proves that the reader is legitimate and the smart card grants access to sensitive information.

The protocol discussed in this section is related to the unilateral two pass authentication protocol specified in ISO/IEC 9798-2 [5]. The standardized technology independent version of this protocol, however, includes an optional identifier to prevent reflection attacks. In scenarios where reflection attacks cannot occur, the identifier can be omitted. None of the standards introduced in Section 2 discusses in which situations reflection attacks are impossible.

4 Gaps in Smart Card Based Authentication Protocol Specifications

We will now discuss vulnerabilities of the two protocols described that can be attributed to incomplete protocol specifications. We also discuss existing gaps when standards reference other standards.

4.1 Authentication Protocol with Key Establishment

The protocol specification in CWA 14890-1 [7] mentions two mandatory checks explicitly: the check whether the challenge received equals the challenge stored and the check whether the correspondent's serial number is included in the message received. Since these two checks are mentioned explicitly, application designers may be led to believe that performing these two checks suffices to run the protocol securely. In the domain of eCard applications, however, readers have to cope with rogue smart cards. For example, the adversary might respond with the serial number SN.r of the reader she is communicating with. This would enable the following reflection attack.

In order to use the command data from the reader's `Mutual Authenticate` command as response to the reader, the adversary has to predict Rand.r before the reader sends the `Mutual Authenticate` command. If she responds with the correct Rand.r to `Get Challenge`, Rand.c and Rand.r have the same value as well as SN.c and SN.r. In this situation, the reader will accept its own command data as a valid response. Although the adversary does not know KDD.r, she knows the information used to generate all four keys. Key generation now starts from two identical key derivation data (KDD.c and KDD.r are the same). Thus independent from the chosen KDD.r, the XOR operation always results in 0. For unpredictable 8 byte challenges, the adversary would need on average 2^{63} attempts for an attack to succeed. The adversary would learn all four keys needed for encryption and integrity computations. Thus, with a complexity of 2^{63} the adversary is able to guess 224 bits (four session 56 bit keys).

This potential reflection attack vulnerability shows that the security of the mutual authentication protocol as specified in CWA 14890-1 [7] does not completely depend on the key length of 2KTDES. Arguably, nonce length and key length address different threat scenarios. Nonces protect against on-line attacks, i.e. nonce prediction. This type of attack is limited by the speed of the card reader. The key length of the shared encryption and integrity keys protect against off-line attacks. Here, the adversary is only limited by the computational power she has at her disposal.

Increasing key size (using 3KTDES) or changing encryption algorithms (to, say, AES) only increases security against off-line prediction attacks. Protocol security against off-line and on-line attacks only increases when challenge and key size are increased together. None of the standards discussed in Section 2 addresses this relationship between protocol message formats, cryptographic algorithms, and key length.

It is not required that nonce and keys have the same length. However, using 3KTDES keys and 64 bits nonces implies the card reader interface is 2^{104} times slower than the adversary's computational power. In case of such substantial discrepancies, standards ought to document that this issue has been considered.

4.2 On the Use of Xor

The rationale for using XOR would be mistrust of the communication partner's random number generator. However, when the adversary can reflect (unknown)

random data back to its creator, XORing these values can result in a loss of security. The XOR operation does not only weaken protocol security it is also unnecessary. Both key derivation data KDD.r and KDD.c could be fed into SHA-1 and still both protocol participants would not have to trust their partner's random number generator. As long as their own random input is unknown to the attacker, the SHA-1 result is unpredictable for the attacker.

4.3 Challenge-Response Authentication Protocol

The *verifier* in the protocol from ISO/IEC 9798 [5] starts the protocol by sending a random challenge. In contrast, the smart card in the protocol from Section 3.2 reacts to a command sent by the reader. Thus, a smart card has to protect itself from adversaries that alter command sequences.

An obvious problem resides in the `Get Challenge` command. The adversary could set the length of the challenge requested to one byte, limiting the smart card to choose the challenge from 256 possibilities.

Smart cards could protect themselves from this attack by rejecting requests for random numbers that are too short. In fact, ISO/IEC 7816-4 [2] allows to reject `Get Challenge` commands. However, checking `Get Challenge` command data is not part of the design pattern described in CWA 14890-1 [7]. The standard stipulates 64 bits challenges but does not include error or plausibility checking nor does it give reason to why 64 bits challenges are chosen. For reasons explained in Section 4.4, it is unlikely that smart card centric standards would include such checks.

The simple authentication protocol from Section 3.2 is susceptible to a reflection attack. First, the adversary sends a `Manage Security Environment` command to the smart card setting the shared key for encryption in an `Internal Authenticate` command. Then, she requests Rand.c from the smart card using `Get Challenge`. Upon receiving Rand.c, she uses Rand.c as command data in an `Internal Authenticate` command causing the smart card to reply with the encrypted challenge. Once she receives the encrypted challenge, the adversary again sends a `Manage Security Environment` command to the smart card setting the same symmetric key for use with `External Authenticate`. Then, the adversary sends the encrypted challenge with an `External Authenticate` to the smart card. The verifier will accept the encrypted challenge and thus believe that the adversary is a legitimate reader.

It is the reader that deviates from the intended protocol. From the smart card point view, however, the reader cannot be trusted until successful completion of the authentication protocol run. Therefore, smart cards must not accept extra commands from the reader before authentication is assured. The smart card operating system can prevent this attack without changing the protocol. First, the operating system could erase random values stored in the card's internal state whenever it receives a `Manage Security Environment` command. Thus, the second `Manage Security Environment` command would delete Rand.c from the card's internal memory and `External Authenticate` would fail. Alternatively, the use of symmetric keys could be restricted so that they could either be

used for encryption or decryption but not for both. Now, the smart card would not encrypt Rand.c with the key it uses to verify the encrypted Rand.c. In yet another solution, a protocol automaton could detect commands arriving out of sequence in a protocol run.

A slight change to the protocol would stop the reflection attack without making assumptions about smart card commands other than `External Authenticate`. Instead of encrypting only Rand.c, the serial number of the device performing the encryption could be included. When the serial number is linked to Rand.c and the smart card checks whether the received serial number is its own, the reflection attack can be prevented.

The attacks described above are not meant to demonstrate new attack methods or to show that insecure implementations are possible. Rather, we used this simple authentication protocol to indicate how easily application designers may make mistakes when adapting protocols from standards.

4.4 Protocol Performance

In applications where protocols are executed frequently, performance is particularly important. Therefore, it is likely that designers would omit optional message fields like the identifier in the authentication protocol described in [5]. Using the identifier (e.g. reader's serial number) to counter reflection attacks would result in added computation and communication overheads. Reader and smart card have to encrypt more data to generate the token. The reader also has to transmit the additional encrypted identifier to the smart card. (In scenarios where the smart card cannot use the selected key to decrypt messages, the identifier must be included in the clear. Otherwise the smart card would be unable to generate and compare the token with the received command data.)

Performance optimizations like the omission of the identifier field result in additional smart card requirements. Application designers may not be aware of those extra requirements since no standard relating to eCard applications describes them. Restricting the use of the encryption key together with `Internal Authenticate` is a common restriction to avoid reflection attacks. Discussing explicitly smart card requirements in at least one of the standards would save designers from obvious mistakes.

5 The Maze of Standards

Section 2 has introduced standards crucial for protocol security in eCard applications. Each standard has its own remit and abstraction layer. However, none of these standards addresses restrictions or requirements they impose on other standards. As a result, application designers can take all the right turns and still get lost in the maze of security standards. Given our observed protocol design vulnerabilities, application designers require security expertise to successfully negotiate this maze. The main issue is that adhering to standards will not automatically result in secure applications.

For many years the security community has told application writers not to design their own security procedures but to adhere to standards. However, when application designers follow this orthodoxy, they are lost in the maze. They must be aware of possible attacks and of suitable countermeasures in order to incorporate related standards into their application architecture. More likely, application designers who do not have the required security expertise will turn to national standards defining cryptographic parameters [8] and standards describing smart card specific authentication protocols [7] for guidance.

Standards that do not properly reference the standards they build upon are another reason why application designers may get lost in the maze. None of the standards mentioned above manages its own security assumptions or handles the security requirements of other standards. Dependencies between standards exacerbate this problem. When a standard references another standard, the standard referenced cannot be aware of the standard referencing. At the point of the referenced standard, the maze cannot be unravelled. In contrast, standards using another standard are aware and should reference them and refine their specifications. For example, the standard defining the smart card based mutual authentication protocol [7] used in this paper, should refine the standard for smart card commands [2]. Whenever one standard imposes restrictions it must specify how conflicts with referenced standards can be resolved. While the national guideline for eCard applications [8] postulates the use of 3KTDES, the protocol from [7] uses 2KTDES only. Since the guideline is aware of the protocol specification, the guideline should advise application designers on how to upgrade the protocol since changing the key length only may be insufficient to increase security. Technology independent standards like [5] should not be referenced from technology and application specific standards like [8] without giving concrete implementation information. This missing hierarchy results in a lack of clarity about the security requirements for the application. Who is responsible for the security requirements and where are those documented?

Standards should not limit possible implementations unnecessarily. For example, developers of smart card operating systems should be free to implement commands as they see fit. Since [2] is a general standard describing what kind of commands smart cards must be able to process, a specification of the interface is sufficient. However, the mutual authentication protocol standard [7] has a concrete security topic since it covers secure signature creation devices. Therefore, it might specify command properties beyond the interface so that secure protocol execution is ensured. This need not confine smart card developers since there may be design choices for fulfilling the security requirement (see Section 4.3).

There are several ways of blocking reflection attacks on the unilateral two pass protocol. We could use the serial number as an identifier. When the smart card receives the `External Authenticate`, it must check whether the serial number is its own. However, this solution might result in performance loss since more data has to be encrypted and transmitted. This solution works as long as the internal state is deleted once the smart card is detached from the reader. In the setting of smart cards, the smart card's internal state must be deleted once it

is pulled from the reader. Therefore, not the protocol itself maintains a session but the smart card. This is contradictory to protocol designs in other areas where protocol engineering tries to remove state information from the entities and include all state information into protocol messages.

6 Certification Issues

The analysis of security protocols in eCard applications ultimately takes place in the context of an executive decision whether to go live with the application. Decision makers want convincing evidence that all relevant security issues have been considered and that security risks are properly mitigated. In today's IT landscape, the security case for an application is frequently made by certifying *compliance* with relevant standards. At the level of security protocols, there is a preference for protocols where formal security proofs exist.

In our investigations, we have shown why this process may not deliver the desired results with current eCard applications. There is the familiar problem of mapping high level security requirements, e.g. card reader authentication, to the kind of security properties formal verification tools are dealing with (correspondence properties). There is the issue that best practice recommendations on cryptographic parameters do not reach out to protocols and, for example, do not cover the length of random challenges. There is the major issue that some protocols as specified would fail protocol verification although their actual implementations are not vulnerable.

We can move forward in two ways from this situation. On one hand, we could advise eCard application designers to follow 'best practice' in protocol design an adopt protocols that are verifiable at the abstract level, giving up on some degrees of freedom in smart card design. Alternatively, we may work on the certification process making sure that dependencies between security standards are properly documented and conducting separate formal protocol analyses per type of card approved for the application.

7 Conclusion

We have analyzed dependencies between a subset of the standards relevant for eCard applications. We have noted that it would be possible to have standard compliant protocol implementations that are vulnerable to attacks violating certain correspondence properties. The attacks we have described follow known patterns, may in practice be nothing more than certificational weaknesses, and may be blocked by specific features on a card. There are however, no provisions in the current set of security standards that point the designer or evaluator to these additional features.

Acknowledgements. We thank the anonymous reviewers for valuable comments and suggestions.

References

1. Bundesministerium für Wirtschaft und Technologie: Chipkarten-Strategie der Bundesregierung (eCard-Strategie) (May 2005), http://www.bmwi.de/BMWi/Redaktion/PDF/C-D/chipkarten-strategie-der-bundesregierung.pdf
2. ISO/IEC 7816-4: Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange (November 2004)
3. Lowe, G.: A hierarchy of authentication specifications. In: Proceedings of the 10th IEEE Computer Security Foundations Workshop, pp. 31–43 (1997)
4. ISO/IEC 9798-1: Information technology — Security techniques — Entity authentication — Part 1: General (July 1997)
5. ISO/IEC 9798-2: Information technology — Security techniques — Entity authentication — Part 2: Mechanisms using symmetric encipherment algorithms (December 2008)
6. ISO/IEC 9798-3: Information technology — Security techniques — Entity authentication — Part 2: Mechanisms using asymmetric encipherment algorithms (October 1998)
7. CEN Workshop Agreement 14890-1: Application Interface for smart cards used as Secure Signature creation Devices — Part 1: Basic requirements (March 2004)
8. Bundesamt für Sicherheit in der Informationstechnik: BSI TR-03116 Technische Richtlinie für eCard-Projekte der Bundesregierung (April 2009), https://www.bsi.bund.de/cae/servlet/contentblob/477230/publicationFile/30897/BSI-TR-03116_pdf.pdf

User-Role Reachability Analysis of Evolving Administrative Role Based Access Control*

Mikhail I. Gofman, Ruiqi Luo, and Ping Yang

Dept. of Computer Science, State University of New York at Binghamton, NY 13902, USA

Abstract. Role Based Access Control (RBAC) has been widely used for restricting resource access to only authorized users. Administrative Role Based Access Control (ARBAC) specifies permissions for administrators to change RBAC policies. Due to complex interactions between changes made by different administrators, it is often difficult to comprehend the full effect of ARBAC policies by manual inspection alone. Policy analysis helps administrators detect potential flaws in the policy specification.

Prior work on ARBAC policy analysis considers only static ARBAC policies. In practice, ARBAC policies tend to change over time in order to fix design flaws or to cope with the changing requirements of an organization. Changes to ARBAC policies may invalidate security properties that were previously satisfied. In this paper, we present incremental algorithms for user-role reachability analysis of ARBAC policies, which asks whether a given user can be assigned to given roles by given administrators. Our incremental algorithms determine if a change may affect the analysis result, and if so, use the information of the previous analysis to incrementally update the analysis result. To the best of our knowledge, these are the first known incremental algorithms in literature for ARBAC analysis. Detailed evaluations show that our incremental algorithms outperform the non-incremental algorithm in terms of execution time.

1 Introduction

Role Based Access Control (RBAC) [12] has been widely used for restricting resource access to only authorized users. In large organizations, RBAC policies are often managed by multiple administrators with varying levels of authority. Administrative Role Based Access Control'97 (ARBAC97) [15] specifies permissions for administrators to change RBAC policies.

Correct understanding of ARBAC policies is critical for maintaining the security of underlying systems. However, ARBAC policies in large organizations are usually large and complex. Consequently, it is difficult to comprehend the full effect of such policies by manual inspection alone. Automated policy analysis helps administrators understand the policy and detect potential flaws. A number of analysis techniques have been developed for user-role reachability analysis of ARBAC [16,18,8], which asks, given an RBAC policy and an ARBAC policy, a set of administrators A, a target user u, and a set of roles (called the "goal"), is it possible for administrators in A to assign the target user u to roles in the goal? It has been shown that the reachability analysis for ARBAC (with fixed role hierarchy) is PSPACE-complete [16,8].

* This work was supported in part by NSF Grant CNS-0855204.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 455–471, 2010.

Prior work on ARBAC analysis considers static ARBAC policies. In practice, ARBAC policies tend to change over time in order to correct design flaws or to cope with changing requirements of an organization. Changes to ARBAC policies may invalidate security properties that were previously satisfied. Hence, the policies need to be re-analyzed to verify their overall correctness. In this paper, we present algorithms for user-role reachability analysis of evolving ARBAC. Our algorithms use the information of the previous analysis to incrementally update the analysis result. In a predominant majority of cases, the information obtained from the previous analysis can be used to update the result more quickly than a complete re-analysis. In a small minority of cases, a complete re-analysis cannot be avoided. To the best of our knowledge, these are the first known algorithms in literature for analysis of evolving ARBAC policies.

We propose three forward incremental analysis algorithms – IncFwd1, IncFwd2, and LazyInc. These algorithms are based on the forward algorithm developed in [18], which constructs a *reduced transition graph* from an ARBAC policy; the goal is reachable iff it is a subset of some state in the graph. IncFwd1 determines if a change to the policy may affect the analysis result, and if so, performs non-incremental analysis; otherwise, the algorithm simply returns the previous analysis result. IncFwd2 reuses the transition graph constructed in the previous analysis and incrementally updates the graph. If a change to the policy may affect the transition graph, IncFwd2 updates the graph by pruning the states and transitions that are invalidated by the change and adding new states and transitions that are enabled by the change. Our experimental data show that updating the transition graph is faster than reconstructing it from scratch. LazyInc reuses the graph constructed in the previous analysis, but delays the updates to the graph until a change is made that may affect the analysis result. Subsequently, the algorithm updates the graph based on the change and other delayed changes.

We have also developed a backward algorithm for incremental analysis based on the backward algorithm in [18], which consists of two stages. In the first stage, a goal-directed backward search is performed, which constructs a directed graph G_b. In the second stage, a forward search is performed on G_b to determine if the goal is reachable. Our algorithm reuses G_b as well as the result computed in the second stage. If a can_revoke rule (which specifies authority to remove a user from a role) is added or deleted, G_b remains the same and the algorithm simply updates the result of the second stage. If a can_assign rule (which specifies the authority to add a user to a role) is added or deleted, the algorithm determines if the change may affect G_b, and if so, updates G_b as well as the result of the second stage.

We have implemented the incremental analysis algorithms presented in this paper and compared the algorithms against the non-incremental algorithms on a set of randomly generated policies and a set of operations that change policies. The experimental data show that our incremental analysis algorithms outperform the non-incremental algorithms in terms of the execution time.

Organization. The rest of the paper is organized as follows. Section 2 provides a brief overview of RBAC, ARBAC, and the user-role reachability analysis algorithms in [18]. Sections 3 and 4 present the incremental forward and backward analysis algorithms, respectively. The experimental results are given in Section 5. The related work is discussed in Section 6 and our concluding remarks appear in Section 7.

2 Preliminaries

2.1 Role Based Access Control

In Role Based Access Control (RBAC), the association of permissions with users is decomposed into two relations: user-role relation and permission-role relation. The user-role relation $(u, r) \in UA$ specifies that user u is a member of role r. The permission-role relation $(r, p) \in PA$ specifies that role r is granted permission p. RBAC also allows to specify a role hierarchy, which is a partial order on the set of roles. For example, role hierarchy relation $r_1 \succeq r_2$ specifies that role r_1 is senior to role r_2.

Administrative Role-Based Access Control'97 (ARBAC97) [15] controls changes to RBAC policies. Authority to assign a user to a role is specified using the can_assign relation. Each can_assign relation is represented as $can\_assign(r_a, c, r_t)$ where r_a is an administrative role, r_t is the target role, and c is the prerequisite condition (or precondition) of the rule. The precondition is a conjunction of literals, where each literal is either r (positive precondition) or $\neg r$ (negative precondition) for some role r. In this paper, we represent the precondition c as $P \wedge \neg N$ where P contains all positive preconditions in c and N contains all negative preconditions in c. Authority to revoke a user from a role is specified using the can_revoke relation. Each can_revoke relation is represented as $can\_revoke(r_a, r_t)$, which specifies that the administrative role r_a has the permission to remove users from the target role r_t.

ARBAC97 requires that administrative roles and regular roles are disjoint, i.e., administrative roles do not appear in the preconditions and target roles of *can_assign* rules, as well as target roles of *can_revoke* rules. This restriction is called the *separate administration restriction* in [18].

2.2 User-Role Reachability Analysis

The user-role reachability analysis problem asks, given an RBAC policy UA_0, an ARBAC policy ψ, a set of administrators A, a target user u_t, and a set of roles (called the "goal"), is it possible for administrators in A to assign the user u_t to roles in the goal, under the restrictions imposed by ψ? Under the separate administration restriction, the problem can be simplified as follows [16]: (1) Because each user's role memberships are controlled independently of other users' role memberships, UA_0 is simplified to be a set of roles assigned to u_t. (2) Because administrative roles and regular roles are disjoint, it suffices to consider only ARBAC rules whose administrative roles are in A or are junior to roles in A. As a result, administrative roles in A can be merged into a single administrative role and all other administrative roles can be eliminated. With the above simplification, the can_assign rule is simplified as $can\_assign(c, r_t)$, the can_revoke rule is simplified as $can\_revoke(r_t)$, and the user-role reachability analysis problem instance can be represented as a tuple $\langle UA_0, \psi, goal \rangle$.

Our incremental algorithms are developed upon two analysis algorithms in [18]. Below, we summarize these two algorithms.

The forward algorithm. In the forward algorithm, roles are classified into *negative* and *non-negative* roles, and *positive* and *non-positive* roles. A role is *negative* if it appears

negatively in some precondition in the policy; other roles are *non-negative*. A role is *positive* if it appears in the goal or appears positively in some precondition in the policy; other roles are *non-positive*. A role that is both negative and positive is called *mixed*.

Given a reachability analysis problem instance $I = \langle UA_0, \psi, goal \rangle$, the algorithm first performs a slicing transformation, which computes a set of roles that are relevant to the goal, and eliminates irrelevant roles and rules. A role r is relevant to the goal iff (1) there exist $r_1 \in goal$ and $can\_assign(P \wedge \neg N, r_1) \in \psi$ such that $r \in P \cup N$, or (2) there exist a role $r_1 \in goal$, $can\_assign(P \wedge \neg N, r_1) \in \psi$, and $r_2 \in P$ such that r is relevant to r_2. $Rel_+(I)$ and $Rel\_(I)$ are used to denote the set of positive and negative roles relevant to the goal, respectively.

Next, the algorithm constructs a *reduced transition graph* $G(I)$ from I and performs analysis based on $G(I)$. Each state in $G(I)$ is a set of roles assigned to the target user and each transition describes an allowed change to the state defined by the ARBAC policy ψ. A transition either has the form $ua(r)$ which adds role r to the state, or has the form $ur(r)$ which removes role r from the state. The following reductions are applied: (1) Irrelevant roles and revocable non-positive roles are removed from UA_0; the resulting set of roles is represented as $InitRm_I(UA_0)$. (2) Transitions that revoke non-negative roles or add non-positive roles are prohibited because they do not enable any other transitions; (3) Transitions that add non-negative roles or revoke non-positive roles are *invisible*; other transitions are called *visible* transitions. The invisible transitions will not disable any other transitions.

Let $\mathrm{closure}(UA, I)$ be the largest state that is reachable from state UA by performing invisible transitions enabled from UA using rules in ψ. The algorithm constructs $G(I)$ as follows. First, the algorithm computes $\mathrm{closure}(InitRm_I(UA_0), I)$, which is the initial state of $G(I)$. The algorithm then computes states reachable from $\mathrm{closure}(InitRm_I(UA_0), I)$: given a state s, there is a transition $s \xrightarrow{ua(r)} \mathrm{closure}(s \cup \{r\})$ if there exists $can\_assign(c, r) \in \psi$ such that r is a mixed role, s does not contain r, and s satisfies c. There is a transition $s \xrightarrow{ur(r)} \mathrm{closure}(s \setminus \{r\})$ if there exists $can\_revoke(r) \in \psi$ such that r is a mixed role, and s contains r. The algorithm returns true iff a state containing all roles in the goal is reached.

The backward algorithm. The backward algorithm in [18] comprises two stages. The first stage performs a backward search from the goal and constructs a directed graph G_b. Each node in G_b is a set of roles. There is an edge $V_1 \xrightarrow{\langle P \wedge \neg N, T \rangle} V_2$ in G_b if there exists a rule $can\_assign(P \wedge \neg N, T) \in \psi$ such that starting from V_1, revoking all roles that appear in N and adding T, results in node V_2. The second stage performs a forward search from the initial nodes in G_b to determine if the goal is reachable. A node V is an initial node in G_b if $V \subseteq UA_0$. The goal is reachable if there exists a *feasible plan* in G_b. A plan corresponds to a path of edges in G_b from the initial node to the node containing the goal, which is constructed as follows: starting from the initial state, for each edge $V_1 \xrightarrow{\langle P \wedge \neg N, T \rangle} V_2$ in G_b, a plan contains a transition $s_1 \xrightarrow{\alpha} s_2$ where s_1 is a state corresponding to V_1, and α is a sequence of actions that revokes roles in $s_1 \cap N$ and adds T. A plan is feasible if it does not revoke irrevocable roles.

To correctly check if G_b contains a feasible plan, each node in G_b is associated with a set of *additional irrevocable roles* ($airs$). $airs(V)$ represents a set of irrevocable roles that appear in the state corresponding to node V, but not in V itself. Formally, given an edge $V_1 \overset{\langle P \wedge \neg N, T \rangle}{\rightarrow} V_2$, $airs(V_2)$ is defined as:

$$\{S \cup ((UA_1 \setminus UA_2) \cap Irrev) \mid S \in airs(UA_1), ((UA_1 \cap Irrev) \cup S) \cap N = \emptyset\}$$

$Irrev$ is the set of irrevocable roles in the policy. $((UA_1 \cap Irrev) \cup S) \cap N = \emptyset$ is the condition that needs to be satisfied in order to add T to the state. The goal is reachable if and only if $airs(goal) \neq \emptyset$.

3 Incremental Forward Algorithms

This section presents three forward algorithms – IncFwd1, IncFwd2, and LazyInc – for incremental reachability analysis of ARBAC. We consider the following changes: (1) add a can_assign rule, (2) delete a can_assign rule, (3) add a can_revoke rule, and (4) delete a can_revoke rule. Because ARBAC with role hierarchy can be transformed to ARBAC without role hierarchy, this paper considers ARBAC without role hierarchy.

IncFwd1 determines if a change may affect the analysis result, and if so, performs non-incremental analysis; otherwise, IncFwd1 returns the previous analysis result. IncFwd2 reuses the transition graph constructed in the previous analysis and incrementally updates the graph. LazyInc also reuses the graph constructed in the previous analysis, but it does not update the graph until an operation that may affect the analysis result is performed. IncFwd1 does not require additional disk space. IncFwd2 and LazyInc require to store the transition graph computed, but are faster than IncFwd1. All three algorithms have the same worst-case complexity as the non-incremental algorithm.

Let $I = \langle UA_0, \psi, goal \rangle$ be a user-role reachability analysis problem instance and $G(I)$ be the transition graph constructed from I using the non-incremental forward algorithm in [18]. Below, we describe the three incremental analysis algorithms in detail.

3.1 Incremental Algorithm: IncFwd1

IncFwd1 is developed based on the following two observations:

- If the analysis result of I is true, then the following changes will not affect the analysis result: (1) add a can_assign rule; (2) add a can_revoke rule; (3) delete a can_assign rule whose target role is non-positive or is irrelevant to the goal; and (4) delete a can_revoke rule whose target role is neither a mixed role nor a negative role in the initial state.
- If the analysis result of I is false, then the following changes will not affect the result: (1) delete a can_assign rule; (2) delete a can_revoke rule; (3) add a can_assign rule whose target role is non-positive or is irrelevant to the goal; and (4) add a can_revoke rule whose target role is neither a mixed role nor a negative role in the initial state.

IncFwd1 uses the slicing transformation result of the previous analysis to determine if a change may affect the analysis result. If so, IncFwd1 performs re-analysis using the non-incremental algorithm; otherwise, IncFwd1 incrementally updates the slicing result and returns the previous analysis result.

```
1  init' = closure(InitRm_{I_1}(UA_0), I_1)
2  if goal ⊆ init' then add init' to G_inc(I_1); return true endif
3  if T ∈ Rel_+(I_1) ∩ Rel_-(I_1)
4    W = reached = {init}
5    while W ≠ ∅ do
6      Remove s from W
7      for s -α-> s_1 ∈ G(I) do
8        add s -α-> s_1 to G_inc(I_1)
9        if goal ⊆ s_1 then markUnproc(W ∪ {s}); return true endif
10       if s_1 ∉ reached then W = W ∪ {s_1}; reached = reached ∪ {s_1} endif
11     endfor
12     if T ∈ s then
13       s' = closure(s \ {T}, I_1)
14       if s' ∉ G(I) then markUnproc({s'}) endif
15       add s -ur(T)-> s' to G_inc(I_1)
16       if goal ⊆ s' then markUnproc(W); return true endif
17     endif
18   endwhile
19 elseif T ∈ (UA_0 ∩ Rel_-(I_1)) and T ∉ Rel_+(I_1) then
20   W_1 = reached = {⟨init, init'⟩}
21   while W_1 ≠ ∅ do
22     Remove ⟨s, s'⟩ from W_1
23     for (s -α-> s_1) ∈ G(I) do
24       s'_1 = closure((s_1 \ {T}) ∪ (s' \ s), I_1)
25       add s' -α-> s'_1 to G_inc(I_1)
26       if goal ⊆ s'_1 then markUnproc({s'_i | ⟨s_i, s'_i⟩ ∈ W_1} ∪ {s'}); return true endif
27       if ⟨s_1, s'_1⟩ ∉ reached_1 then W_1 = W_1 ∪ {⟨s_1, s'_1⟩}; reached_1 = reached_1 ∪ {⟨s_1, s'_1⟩} endif
28     endfor
29     if addNewTrans(s, s') == true then markUnproc({s'_i | ⟨s_i, s'_i⟩ ∈ W_1} ∪ {s'}); return true endif
30   endwhile
31 else return the analysis result of I endif
32 if G_inc(I_1) == ∅ then add init' to G_inc(I_1); return false endif
33 process all states marked UnProcessed with non − incremental alg.
```

Fig. 1. Pseudocode for adding $can\_revoke(T)$

3.2 Incremental Algorithm: IncFwd2

IncFwd2 reuses the slicing transformation result and the transition graph computed in the previous analysis, and incrementally updates the graph. States whose outgoing transitions are not computed or not computed completely are marked as "UnProcessed". If a goal that is reachable in I becomes unreachable after a change is made to the policy, IncFwd2 performs non-incremental analysis from states that were previously marked as "UnProcessed". Let $G_{inc}(I)$ denote the transition graph computed by IncFwd2 for the problem instance I. Below, we describe IncFwd2 in detail.

Add a can_revoke rule Suppose that $can\_revoke(T)$ is added to the policy ψ. Let $I_1 = \langle UA_0, \psi \cup \{can\_revoke(T)\}, goal\rangle$. Figure 1 gives the pseudocode for constructing graph $G_{inc}(I_1)$ from $G(I)$.

Adding a *can_revoke* rule does not change the result of the slicing transformation. Thus, $Rel_+(I_1) = Rel_+(I)$ and $Rel_-(I_1) = Rel_-(I)$. If T is a mixed role relevant to the goal, the algorithm starts from the initial state $init$ of $G(I)$ and, for every state containing T, adds a transition $ur(T)$ and marks new target state as

"UnProcessed" (lines 5 – 18). Otherwise, if T is in UA_0 and is both negative and non-positive, the algorithm replaces $init$ with $closure(init \setminus \{T\}, I_1)$ and propagates roles in $closure(init \setminus \{T\}, I_1) \setminus init$ to states reachable from $init$ (lines 19–28). Because different states in $G_{inc}(I_1)$ may be computed from the same state in $G(I)$, the workset W_1 contains pairs of the form $\langle s, s' \rangle$ where $s \in G(I)$, $s' \in G_{inc}(I_1)$, and s' is computed from s. The algorithm then calls function *addNewTrans* to add new transitions enabled by the rule; this function returns true if a state containing the goal is reached (line 29). The above process is then repeated on states reachable from $init$. In other cases, the transition graph as well as the analysis result remain the same (line 31).

If the goal is reachable, the algorithm calls function *markUnproc* to mark states in $G_{inc}(I_1)$, whose outgoing transitions are not computed (i.e., the remaining states in W or W_1) or not computed completely (i.e., the state from which the goal is reached by a transition), as "UnProcessed" (lines 9, 16, 26, 29). Otherwise, the algorithm processes all states marked as "UnProcessed" using the non-incremental algorithm (line 33).

Delete a can_revoke rule. Suppose that $can\_revoke(T)$ is removed from ψ. Let $I_2 = \langle UA_0, \psi \setminus \{can\_revoke(T)\}, goal \rangle$. Deleting this rule does not change the analysis result if T is not a relevant mixed role and is not a negative role in UA_0. Otherwise, the initial state may change and transitions that revoke T should be deleted. The incremental algorithm is described below.

Deleting a *can_revoke* rule does not change the result of the slicing transformation. If T is a mixed role relevant to the goal, the algorithm starts from the initial state of $G(I)$ and deletes transitions that revoke T. If T is in UA_0 and is both negative and non-positive, then after deleting $can\_revoke(T)$, T should be added back to the initial state. In this case, the algorithm computes a set R_T of roles that may be invalidated by T. A role $r \in R_T$ if (1) T is a negative precondition of a can_assign rule whose target role is r or (2) there exists a role $r' \in R_T$ such that r' is a positive precondition of a can_assign rule whose target role is r. If the number of relevant roles in R_T, which are both positive and non-negative, is small, then for every transition $s \xrightarrow{\alpha} s_1$ in $G(I)$ and state $s' \in G_{inc}(I_2)$ computed from s, the algorithm computes transition $s' \xrightarrow{\alpha} s'_1$ by removing such roles that do not appear in s' from s_1, and computing the closure of the resulting state. The algorithm also checks if transitions that add mixed roles in R_T are invalidated, and if so, removes the transitions. Otherwise, the algorithm performs non-incremental analysis as removing a large number of roles from every state may be more expensive than a complete re-analysis. If the state containing the goal is deleted, the algorithm performs non-incremental analysis from states marked as "UnProcessed".

Add a can_assign rule. Suppose that $can\_assign(P \wedge \neg N, T)$ is added to the policy ψ. Let $I_3 = \langle UA_0, \psi \cup \{can\_assign(P \wedge \neg N, T)\}, goal \rangle$. Figure 2 gives the pseudocode for constructing graph $G_{inc}(I_3)$ from $G(I)$.

If T is non-positive or is irrelevant to the goal, then adding this rule does not change the transition graph (line 1). Otherwise, the classification of roles may change: (1) An irrelevant role in I may become relevant in I_3; (2) A relevant role that is both positive and non-negative in I may become a mixed role in I_3; and (3) A relevant role that is both negative and non-positive in I may become a mixed role in I_3. In this case, the algorithm performs incremental slicing to compute $Rel_+(I_3)$ and $Rel_-(I_3)$ (function

1 if $T \notin Rel_+(I)$ then *return the analysis result of* I
2 else
3 $\langle Rel_+(I_3), Rel_-(I_3)\rangle = inc\_slicing()$
4 $init' = closure(InitRm_{I_3}(UA_0), I_3)$
5 if $goal \subseteq init'$ then *add* $init'$ *to* $G_{inc}(I_3)$; return $true$ endif
6 $RevRoles = ((init' \setminus init) \cap (Rel_+(I_3) \cap Rel_-(I_3)) \setminus Irrev)$
7 $PosNonnegToMix = ((Rel_+(I) \setminus Rel_-(I)) \cap (Rel_+(I_3) \cap Rel_-(I_3)))$
8 $AddRoles = init \cap PosNonnegToMix$
9 if $AddRoles \cup RevRoles \neq \emptyset$ then
10 $\langle answer, lastState\rangle = addTransSeq(init, init', RevRoles, AddRoles)$
11 if $answer == true$ then return $true$ else $W = reached = \{\langle init, lastState\rangle\}$ endif
12 else $W = reached = \{\langle init, init'\rangle\}$ endif
13 while $W \neq \emptyset$ do
14 *Remove* $\langle s, s'\rangle$ *from* W
15 for $s \xrightarrow{\alpha} s_1 \in G(I)$ do
16 $AddRoles = s_1 \cap PosNonnegToMix$
17 if $AddRoles \neq \emptyset$ then
18 $\langle answer, s_1'\rangle = addTransSeq(s, s', \emptyset, AddRoles)$
19 if $answer == true$ then $markUnproc(\{s_j' | \langle s_j, s_j'\rangle \in W\} \cup \{s'\})$; return $true$ endif
20 else
21 $s_1' = closure(s_1 \cup (s' \setminus s), I_3)$
22 *add* $s' \xrightarrow{\alpha} s_1'$ *to* $G_{inc}(I_3)$
23 if $goal \subseteq s_1'$ then $markUnproc(\{s_j' | \langle s_j, s_j'\rangle \in W\} \cup \{s'\})$; return $true$ endif
24 endif
25 if $\langle s_1, s_1'\rangle \notin reached$ then $reached = reached \cup \{\langle s_1, s_1'\rangle\}$; $W = W \cup \{\langle s_1, s_1'\rangle\}$ endif
26 endfor
27 if $addNewTrans(s') == true$ then $markUnproc(\{s_j' | \langle s_j, s_j'\rangle \in W\} \cup \{s'\})$; return $true$ endif
28 endwhile
29 if $G_{inc}(I_3) == \emptyset$ then *add* $init'$ *to* $G_{inc}(I_3)$; return $false$ endif
30 *process all states marked UnProcessed with non − incremental alg.*
31 endif

Fig. 2. Pseudocode for adding $can\_assign(P \wedge \neg N, T)$

$inc\_slicing$ in line 3): $Rel_+(I_3) = Rel_+(I) \cup \{r | r$ is a positive role relevant to $T\}$ and $Rel_-(I_3) = Rel_-(I) \cup \{r | r$ is a negative role relevant to $T\}$. The algorithm also computes a set $RelRule$ of rules, which are sufficient to consider during the incremental analysis. Let ρ be a can_assign rule, $target(\rho)$ be the target role of ρ, and $poscond(\rho)$ be the set of positive preconditions of ρ. $RelRule$ is defined as follows:

(1) $can\_assign(P \wedge \neg N, T) \in RelRule$.
(2) a can_assign rule $\rho \in RelRule$ if
  (a) $target(\rho) \in Rel_+(I_3)$ and there exists $\rho' \in RelRule$ such that $target(\rho') \in poscond(\rho)$ or
  (b) $target(\rho)$ is a positive role relevant to T.
(3) $can\_revoke(r) \in RelRule$ if r is a mixed role in I_3 or is a negative role in UA_0.

$RelRule$ consists of (1) the new rule, (2a) relevant can_assign rules enabled by the new rule, (2b) can_assign rules that enable the new rule, and (3) can_revoke rules which revoke mixed roles or negative roles in UA_0.

Next, the algorithm computes the new initial state $init' = \text{closure}(InitRm_{I_3}(UA_0), I_3)$ (line 4), which may be different from the initial state $init = \text{closure}(InitRm_I(UA_0), I)$ of $G(I)$. Theorem 1 gives the relationship between $init'$ and $init$, which enables us to reuse $G(I)$ to construct $G_{inc}(I_3)$.

Theorem 1. Let $init = \text{closure}(InitRm_I(UA_0), I)$ and $init' = \text{closure}(InitRm_{I_3}(UA_0), I_3)$. One of the following holds: (1) $init = init'$; (2) $init' = \text{closure}(init, I_3)$; or (3) $init$ is reachable from $init'$ through a sequence of transitions: $init' \xrightarrow{ur(r_1)} s_1 \ldots \xrightarrow{ur(r_n)} s_n \xrightarrow{ua(r_{n+1})} s_{n+1} \ldots s_{m-1} \xrightarrow{ua(r_m)} \text{closure}(init \cup s_{m-1}, I_3)$ where $\{r_1, \ldots, r_n\}$ are revocable mixed roles in $init' \setminus init$ and $\{r_{n+1}, \ldots, r_m\}$ are roles in $init \setminus init'$ that are turned from both positive and non-negative to mixed. □

Case (1) states that the initial state does not change after the rule is added. In Case (2), new roles are added to the initial state, but no roles are turned from both positive and non-negative to mixed. In these two cases, the algorithm adds roles in $init' \setminus init$ to $init$ and updates the graph (lines 20–24). In case (3), some roles in $init$ are turned from both positive and non-negative to mixed ($AddRoles$ in line 8), from irrelevant to relevant, or from revocable non-positive to mixed ($RevRoles$ in line 6). In this case, the algorithm calls function *addTransSeq* (line 10) to add a sequence of transitions from $init'$ to $\text{closure}(init \cup s_{m-1}, I_3)$, which revokes roles in $RevRoles$, adds roles in $AddRoles$, and marks new states not containing the goal as "UnProcessed". This function returns $\langle answer, lastState \rangle$ where $answer$ is true if the sequence contains the goal state and is false otherwise, and $lastState$ is the last state of the sequence.

Finally, the algorithm calls function *addNewTrans* to add new transitions from $init'$ (line 27) using rules in $RelRule$ and marks new states as "UnProcessed". The above process is then repeated on states reachable from $init$.

Delete a can_assign rule. Suppose that $can\_assign(P \wedge \neg N, T)$ is deleted from policy ψ. Let $I_4 = \langle UA_0, \psi \setminus \{can\_assign(P \wedge \neg N, T)\}, goal \rangle$. If T is not a positive role relevant to the goal, then the transition graph remains the same. Otherwise, role T and roles that are reachable through T may become unreachable. Let A_T be a set of roles that may be reachable through T. A role r is in A_T if: (1) T is a positive precondition of a can_assign rule whose target role is r or (2) there exists a role $r' \in A_T$ such that r' is a positive precondition of a can_assign rule whose target role is r.

Deleting $can\_assign(P \wedge \neg N, T)$ may change the classification of T from mixed to both positive and non-negative. This occurs when targets of all can_assign rules, which contain T in their negative preconditions, become non-positive after the rule is deleted. Similarly, roles other than T may change from mixed to both positive and non-negative (*MixtoNonneg*), from mixed to both negative and non-positive (*MixtoNonpos*), and from relevant to irrelevant (*RevtoIrr*). Below, we describe the algorithm.

First, the algorithm performs slicing and computes the new initial state $init' = closure(InitRm_{I_4}(UA_0), I_4)$. $init'$ may be different from the initial state $init = closure(InitRm_I(UA_0), I)$ of $G(I)$. Theorem 2 gives the relationship between $init$ and $init'$, which enables us to reuse $G(I)$ to construct $G_{inc}(I_4)$.

Theorem 2. Let $init = closure(InitRm_I(UA_0), I)$, $init' = closure(InitRm_{I_4}(UA_0), I_4)$, and $Invalid = (init \setminus init') \cap (A_T \cup \{T\})$. One of the following holds: (1) $init' = init$; (2) $init' = init \setminus (RevtoIrr \cup Invalid \cup \{S \in MixtoNonpos | S \text{ is revocable}\})$; or (3) $G(I)$ contains the following sequence of transitions: $init \xrightarrow{ua(r_1)} s_1 \ldots s_{n-1} \xrightarrow{ua(r_n)} (init' \cup (s_{n-1} \cap (RevtoIrr \cup Invalid \cup MixtoNonpos)))$ where $\{r_1, \ldots, r_n\} = (init' \setminus init) \cap MixtoNonneg$. □

Case (1) states that the initial state does not change. In Case (2), $init$ does not contain roles turned from mixed to both positive and non-negative, but may contain roles turned

from relevant to irrelevant, revocable roles turned from mixed to non-positive, or roles that cannot be rederived after the can_assign rule is deleted. In this case, the algorithm updates the graph from $init$, removes such roles from $init$ and states reachable from $init$, and removes transitions that add or revoke such roles. In case (3), $init$ contains roles turned from mixed to both positive and non-negative. In this case, the algorithm identifies the state $(init' \cup (s_{n-1} \cap (RevtoIrr \cup Invalid \cup MixtoNonpos)))$ in $G(I)$. The algorithm then updates the graph from this state by removing roles in $(s_{n-1} \cap (RevtoIrr \cup Invalid \cup MixtoNonpos))$ from this state and all reachable states.

The graph is updated as follows. For every transition $s_1 \xrightarrow{\alpha} s_2$ in $G(I)$, if α adds/revokes a role that is turned from mixed to non-positive or if α can no longer be derived, the algorithm removes the transition. Otherwise, if $s_2 \setminus s_1$ contains T and T cannot be re-derived, the algorithm removes T and all roles that cannot be derived without T from s_2. If α adds a role that is turned from mixed to both positive and non-negative, the algorithm removes the transition and updates the graph using a way similar to (3) of Theorem 2.

3.3 Lazy Incremental Forward Algorithm

This section presents a lazy incremental analysis algorithm that delays updates to the transition graph until an operation, which may affect the analysis result, is performed. Due to space constraints, this section presents only the algorithm for the case where the analysis result of the original policy is true. The case where the analysis result is false is handled similarly. Let $I = \langle UA_0, \psi, goal \rangle$ be a reachability analysis problem instance. The algorithm is described below.

Add a can_assign or a can_revoke rule. Adding a can_assign or a can_revoke rule does not affect the analysis result though it may affect the transition graph. In this case, we do not update the graph. Instead, we store the rule in a set $DelayedRule$. This set will be used to update the transition graph when an operation that may affect the analysis result is performed.

Delete a can_assign or a can_revoke rule. Assume that $can\_assign(P \wedge \neg N, T)$ is deleted from ψ. If T is not a positive role relevant to the goal, the algorithm returns true. Otherwise, the algorithm performs the following steps.

Let $\psi' = (\psi \setminus \{can\_assign(P \wedge \neg N, T)\}) \cup DelayedRule$ and $I' = \langle UA_0, \psi', goal \rangle$. First, the algorithm computes $Rel_+(I')$ and $Rel_-(I')$. The algorithm then updates the transition graph using the deleted rule and delayed operations that may affect the analysis result after the rule is deleted. Such operations include addition of can_assign rules in $DelayedRule$ whose target roles are in $Rel_+(I')$, and addition of can_revoke rules in $DelayedRule$ that revoke relevant mixed roles or negative roles in UA_0. Finally, the algorithm updates the graph using one of the following two approaches.

In the first approach, we update every state and transition of the graph by performing all operations of IncFwd2 described in sections 3.2 with the following difference: when applying Theorem 1, not all transitions adding roles $r_{n+1}, \ldots, r_m$ may be enabled in I' because some of them may depend on the deleted can_assign rule.

In the second approach, the algorithm first considers the operation that deletes $can\_assign(P \wedge \neg N, T)$ and applies IncFwd2 for deleting can_assign to update the

graph. If the analysis result changes from true to false, the algorithm updates the graph using rules in $DelayedRule$. The graph is updated in a way similar to algorithms in Figures 1 and 2, but considering multiple added rules.

The second approach is expected to perform better than the first one if the analysis result does not change after the rule is deleted, and worse otherwise. This is because in the former case, the graph is processed once, but in the latter case, the graph is processed twice. Both approaches are expected to perform better than IncFwd2 where each change is processed individually, because the graph will be processed fewer times when applying these two approaches. Our implementation adopted the first approach.

Deleting a can_revoke rule is handled similarly.

4 Incremental Backward Algorithm

This section presents a backward algorithm for incremental user-role reachability analysis. Similar to IncFwd2, our backward algorithm uses the graph G_b and the airs computed in the previous analysis to incrementally update the result. Ideas used in IncFwd1 and LazyInc are also applicable to the backward algorithm.

To support efficient incremental analysis, we extend the non-incremental algorithm as follows: (1) Prior to analysis, we compute a set of roles relevant to the goal, which enables us to quickly determine if a change to the policy may affect the analysis result. (2) We store the graph as well as the airs computed in a file. The initial nodes are also stored, in the order in which they are processed in the second stage. (3) For every node V, we associate every set of $airs(V)$ with the edge along which the set is computed. This enables us to quickly identify the set of airs computed from a given edge. (4) If an edge is processed in the second stage of the algorithm, the edge is marked 1; otherwise, the edge is marked 0. Let $airs'(V)$ denote the airs of node V computed by the incremental algorithm. Below, we describe the algorithm.

Add a can_revoke rule. Assume that $can\_revoke(T)$ is added to the policy. Graph G_b is unaffected and we simply update the airs of nodes from the first initial node V_0. Let $rm(T, airs(V))$ denote $\{S - \{T\} | S \in airs(V)\}$. $airs'(V)$ is computed as follows:

- $airs'(V_0) = rm(T, airs(V_0))$
- For every edge $V_1 \overset{\langle P \wedge \neg N, r \rangle}{\rightarrow} V_2$, $airs'(V_2)$ is computed as the union of the following sets:
  (1) $rm(T, airs(V_2))$
  (2) $\{S \cup (Irrev \cap (V_2 \setminus V_1)) | S \in airs'(V_1) \setminus rm(T, airs(V_1)), ((Irrev \cap V_1) \cup S) \cap N = \emptyset\}$
  (3) $((T \in N)?\{(S \setminus \{T\}) \cup (Irrev \cap (V_2 \setminus V_1)) | S \in airs(V_1), T \in S, ((Irrev \cap V_1) \cup (S \setminus \{T\})) \cap N = \emptyset\} : \emptyset)$

(1) contains $airs(V_2)$ with T removed from every set. (2) contains sets of additional irrevocable roles computed from new sets in airs of V_1 along the edge. (3) is computed from sets in $airs(V_1)$ that did not satisfy the negative precondition of the edge because they contained T; since T becomes revocable, they are added to $airs'(V_2)$.

If the goal is not reachable from V_0, we pick up the second initial node and repeat the above process. If the goal is reachable, the algorithm updates the airs of nodes until it encounters an edge marked 0. This is because, after a can_revoke rule is added, the goal

that was previously unreachable may become reachable and the goal that was previously reachable may be reached earlier.

Delete a can_revoke rule. Suppose that $can\_revoke(T)$ is deleted from the policy. Graph G_b remains the same and we simply update the airs of nodes. First, starting from the first initial node V_0, the algorithm searches G_b along edges marked 1, for nodes whose airs may change. The airs of a node V may change if: (1) T is in the initial state and V does not contain T; or (2) there is an edge $V' \xrightarrow{\alpha} V$ such that $T \in V'$ and $T \notin V$. If such a node does not exist, the algorithm returns the previous analysis result. Otherwise, the algorithm updates the airs of the node as well as the airs of all nodes reachable from this node by edges marked 1 as follows: for every edge $e = V_1 \xrightarrow{\alpha} V_2$, if $airs'(V_1) = airs(V_1)$, then $airs'(V_2) = airs(V_2)$; otherwise, $airs'(V_2)$ is computed by removing sets in $airs(V_2)$ that are computed along e and recomputing airs along e using the non-incremental algorithm. If an edge marked 0 is encountered, the algorithm computes the set of airs along this edge. If the goal is not reachable from V_0, the algorithm picks up another initial node and repeats the above process.

Add a can_assign rule. Suppose that $can\_assign(P \wedge \neg N, T)$ is added to the policy. If T is not a positive role relevant to the goal, the algorithm returns the previous analysis result; otherwise, the algorithm incrementally updates graph G_b and the airs of nodes.

In the first stage, starting from nodes that contain T, the algorithm computes all reachable edges enabled by the new rule. For each new edge $V \xrightarrow{\alpha} V'$, if V is a node in G_b and $airs(V) \neq \emptyset$, V is added to a set *affectedNodes*. Also, all new initial nodes are added to a set *newInit*. The new edges are marked 0, indicating that they have not been processed in the second stage.

If the previous analysis result is true, the algorithm returns true. Otherwise, the algorithm updates the airs of nodes as follows. For every node $V \in affectedNodes$, it updates the airs of nodes reachable from V along new edges until a node containing T is encountered. The algorithm then updates the airs of this node as well as the airs of all nodes reachable from this node: for every edge $V_1 \xrightarrow{\alpha} V_2$, the algorithm computes $airs'(V_2)$ by adding sets that are computed from $airs'(V_1) \setminus airs(V_1)$ along the edge, to $airs(V_2)$. If $airs(goal) \neq \emptyset$, the algorithm returns true. Otherwise, the algorithm computes the airs of nodes reachable from the new initial nodes in $newInit$.

Delete a can_assign rule. Suppose that $can\_assign(P \wedge \neg N, T)$ is deleted from the policy. If T is not a positive role relevant to the goal, the algorithm returns the previous analysis result; otherwise, the algorithm performs the following steps.

First, the algorithm back-chains from the goal and marks the following nodes as valid nodes: (1) the goal node, and (2) for every edge $V_1 \xrightarrow{\alpha} V_2$, if $\alpha \neq \langle P \wedge \neg N, T \rangle$ and V_2 is valid, then V_1 is valid. Valid nodes are nodes that remain in the graph after the rule is deleted. Next, for every edge $V_1 \xrightarrow{\langle P \wedge \neg N, T \rangle} V_2$, the algorithm deletes the sets in $airs(V_2)$ that are computed through the edge and adds V_2 to a set L_T. The algorithm then deletes all nodes not marked valid, edges containing at least one such node, and edges of the form $V_1 \xrightarrow{\langle P \wedge \neg N, T \rangle} V_2$. Finally, the algorithm updates the airs of nodes reachable from nodes in L_T: for every edge $V_1 \xrightarrow{\alpha} V_2$, $airs'(V_2)$ is computed by removing all sets from $airs(V_2)$ that are computed from $airs(V_1) \setminus airs'(V_1)$. If the goal was previously

reachable but $airs'(goal) = \emptyset$, the algorithm computes airs of nodes that have not been processed using the non-incremental algorithm.

Note that, an alternative (and incorrect) approach to detecting invalidated nodes is to back-chain from the goal, delete edges computed through the deleted rule, delete nodes without outgoing edges, and then delete edges that contain deleted nodes. Such an approach will fail if the graph contains cycles: nodes in a cycle may not be reachable from the goal node after the rule is deleted, but still contain outgoing edges.

5 Experimental Results

This section presents experimental results of our incremental analysis algorithms. All reported data were obtained on a 2.5GHz Pentium machine with 4GB RAM.

5.1 Experimental Results: Incremental Forward Analysis Algorithms

We apply the non-incremental and incremental forward algorithms to an ARBAC policy ψ_1 generated using the random policy generation algorithm in [18]. The parameter values (e.g., the percentage of mixed roles) in ψ_1 are similar to those in the university ARBAC policy developed in [18]. We choose two goals $goal_1$ and $goal_2$ such that $goal_1$ is reachable from the initial state $\emptyset$ and $goal_2$ is unreachable, and the size of transition graphs constructed during analysis is reasonably large. We then randomly generate a set of operations that add rules to ψ_1 or delete rules from ψ_1, and use them to compare the performance of our incremental algorithms against the non-incremental algorithm.

Table 1 compares the execution time of the non-incremental algorithm (NonInc), IncFwd1, and IncFwd2 for goals $goal_1$ and $goal_2$, when a change is made to policy ψ_1. Each data point reported is an average over 32 randomly generated rules, except for the case "add can_revoke": because only 10 roles cannot be revoked in ψ_1, we generate only rules that revoke these 10 roles. Columns "States" and "Trans" give the average number of states and transitions computed using NonInc, respectively.

Results for adding/deleting a can_revoke rule. Table 1 shows that, when a can_revoke rule is added, IncFwd2 is 26.39 and 7.52 times faster than NonInc for $goal_1$ and $goal_2$, respectively. When only the 2 rules that revoke mixed roles are considered, IncFwd2 is 5.88 times faster than NonInc for $goal_1$ and 2.17 times faster than NonInc for $goal_2$. When considering both goals, IncFwd2 is 10.34 and 1.85 times faster than IncFwd1 and NonInc, respectively.

When a *can_revoke* rule is deleted, IncFwd2 is 24.51 and 25.76 times faster than NonInc for $goal_1$ and $goal_2$, respectively. When considering only the 8 rules that revoke

Table 1. Performance results of NonInc, IncFwd1, and IncFwd2 on ψ_1 for goals $goal_1$ and $goal_2$

Operation	$goal_1$					$goal_2$				
	States	Trans	Time(Sec.)			States	Trans	Time(Sec.)		
			NonInc	IncFwd1	IncFwd2			NonInc	IncFwd1	IncFwd2
add can_assign	30568	233298	103.69	0	13.87	33396	440052	236.19	233.38	160.22
delete can_assign	19983	153931	58.07	56.23	13.58	18000	220248	84.38	0	18.2
add can_revoke	20866	159290	60.43	0	2.29	19968	247834	97.8	28.23	13.01
delete can_revoke	20297	154853	58.59	44.09	2.39	18432	224928	86.81	0	3.37

Table 2. Performance results of NonInc, IncFwd1, IncFwd2, and LazyInc for $goal_1$ and $goal_2$ on ten sequences of operations

Goal	States	Trans	Time(Sec.)			
			NonInc	IncFwd1	IncFwd2	LazyFwd
$goal_1$	30219	240802	110.76	45.75	12.58	6.09
$goal_2$	19176	288142	94.86	34.57	12.41	10.59

mixed roles, IncFwd2 is 6.07 times faster than NonInc for $goal_1$ and 6.41 times faster than NonInc for $goal_2$. When both goals are considered, IncFwd2 is 25.24 and 7.65 times faster than NonInc and IncFwd1, respectively.

Results for adding/deleting a can_assign rule. All can_assign rules added/deleted are relevant to the goal. Observe from Table 1 that, when a can_assign rule is added, IncFwd2 is 7.48 and 1.47 times faster than NonInc for $goal_1$ and $goal_2$, respectively. This is because, the size of the transition graph increases less significantly for $goal_1$ than $goal_2$ (160550 vs 440052 transitions). In particular, for one of the 32 rules generated for $goal_2$, the size of the graph constructed after the rule is added is 10 times the size of the graph constructed before the rule is added. As a result, IncFwd2 computes a large number of new states and transitions using the non-incremental algorithm, and hence is only slightly faster than NonInc for this rule (1868sec vs 1971sec). When considering both goals, IncFwd2 is 2 and 1.35 times faster than NonInc and IncFwd1, respectively.

When a can_assign rule is deleted, IncFwd2 is 4.3 and 4.6 times faster than NonInc for goals $goal_1$ and $goal_2$, respectively. When both goals are considered, IncFwd2 is 4.48 and 1.77 times faster than NonInc and IncFwd1, respectively.

Results for adding/deleting a sequence of rules. Table 2 compares the performance of the non-incremental algorithm and three incremental algorithms on 10 sequences of operations. In every sequence, only the last operation affects the analysis result. The column "Time" gives the average execution time of the algorithms for each operation. The results show that, when analyzing policy ψ_1 with $goal_1$, for each operation, LazyInc is 18.18, 7.51, and 2.07 times faster than NonInc, IncFwd1, and IncFwd2, respectively. When analyzing ψ_1 with $goal_2$, for each operation, LazyInc is 8.96, 3.26, and 1.17 times faster than NonInc, IncFwd1, and IncFwd2, respectively.

Disk space consumption. IncFwd1 does not require additional disk space. The amount of disk space used in IncFwd2 and LazyInc to store the transition graph is 29.6 MB and 43.8 MB, respectively. This is because: (1) the size of the graph is large; and (2) we store states for every transition, which results in storing a state multiple times, and the size of the state is usually large. We expect that, by storing each state only once and storing the references to states in each transition, we will be able to significantly reduce the disk space consumption without significantly affecting their performance.

5.2 Experimental Results: Incremental Backward Algorithm

Table 3 compares the execution time of non-incremental and incremental backward algorithms. The column headings "NonIncBack" and "IncBack" refer to the non-incremental and incremental algorithm, respectively. We choose a randomly generated

Table 3. Performance comparison of NonIncBack and IncBack on ψ_2 for $goal_3$ and $goal_4$

Operation	$goal_3$						$goal_4$					
	Nodes	Edges	Time (NonIncBack)		Time (IncBack)		Nodes	Edges	Time (NonIncBack)		Time (IncBack)	
			Stage 1	Stage 2	Stage 1	Stage 2			Stage 1	Stage 2	Stage 1	Stage 2
add can_revoke	37053	269878	24.73	23.82	1.06	2.54	12475	66128	3.37	67.28	0.92	17.32
add can_assign	37112	285768	25.6	68.47	2.92	0	12481	66355	3.36	44.21	0.97	5.31
delete can_revoke	37112	284297	69.89	95.26	0.59	0.32	12481	68199	3.35	91.82	0.15	0.40
delete can_assign	37085	283541	25.39	41.83	4.31	0.22	12476	68071	3.56	36.9	1.28	0.16

policy ψ_2, two goals $goal_3$ and $goal_4$, and two initial states i_3 and i_4, so that (1) $goal_3$ is reachable from i_3 and $goal_4$ is not reachable from i_4; (2) the size of the graph is reasonably large (269889 and 66097 edges for $goal_3$ and $goal_4$, respectively); and (3) both stages of the algorithm are performed during analysis. The additional amount of disk space used to store the graph is 15.4 MB and 3.7 MB for $goal_3$ and $goal_4$ respectively.

When a can_revoke rule is added or deleted, the graph remains the same. The execution time of the first stage of IncBack refers to the time for loading the graph constructed in the previous analysis. Table 3 shows that, when a can_revoke rule is added, IncBack is 13.49 and 3.87 times faster than NonIncBack for $goal_3$ and $goal_4$, respectively. When a can_revoke rule is deleted, IncBack is 181.48 and 173.04 times faster than NonIncBack for $goal_3$ and $goal_4$, respectively.

When a can_assign rule is added to the policy, IncBack is 32.22 times and 7.57 times faster than NonIncBack for $goal_3$ and $goal_4$, respectively. When the previous analysis result is true, IncBack does not perform the second stage and hence the execution time of the second stage is 0. When a can_assign rule is deleted, IncBack is 14.84 and 28.1 times faster than NonIncBack for $goal_3$ and $goal_4$, respectively.

6 Related Work

A number of researchers investigated the problem of analyzing (a subset of) static ARBAC policies. In contrast, we consider changes to ARBAC policies. Below, we summarize their work. Li et al. [10] presented algorithms and complexity results for analysis of two restricted versions of ARBAC97 – AATU and AAR. This work did not consider negative preconditions in ARBAC policies. Schaad and Moffett [17] used the Alloy analyzer [7] to check separation of duty properties for ARBAC97. However, they did not consider preconditions in ARBAC policies. Sasturkar et al. [16] and Jha et al. [8] presented algorithms and complexity results for analysis of ARBAC policies subject to a variety of restrictions. Stoller et. al. [18] proposed the first fixed-parameter-tractable algorithms for analyzing ARBAC policies, which lays the groundwork for incremental analysis algorithms presented in this paper.

Crampton [2] showed undecidability of reachability analysis for RBAC, whose changes are controlled by commands consisting of pre-conditions and administrative operations; some commands are not expressible in the form allowed in ARBAC97 and some commands use administrative operations that change the ARBAC policy.

Prior works [6,14,9] have also considered changes to access control policies. However, the changes are not controlled by ARBAC, nor did these work consider changes to

administrative policies. Fisler *et al.* [3] developed a change-impact analysis algorithm for RBAC policies based on binary decision diagram (BDD) by computing the semantic difference of two policies and checking properties of the difference. We consider changes to ARBAC policies, instead of RBAC policies. Furthermore, we also propose a lazy analysis algorithm.

Incremental computation has been studied in several areas, including deductive databases, logic programming, and program analysis. However, to the best of our knowledge, we are the first to develop the incremental algorithms for ARBAC policy analysis. Gupta [4] incrementally updated materialized views in databases by counting the number of derivations for each relation. Our approach is more efficient for analyzing ARBAC policies: we compute each transition only once; counting derivations would require determining all ways in which a transition can be computed. Gupta *et al.* [5] proposed a two-phase delete-rederive algorithm, which first deletes relations that depend on the deleted relation, and then rederives the deleted relations that have alternative derivations. Similar approaches were adapted in [13]. We avoid the rederivation phase by removing only those roles from the state for which all derivations have been invalidated. Lu *et al.* [11] proposed a Straight Delete algorithm (StDel) which eliminates the rederivation phase of delete-rederive algorithm. Direct application of StDel to ARBAC policy analysis would require storing all derivations for every state and every transition and, just as with counting, would be less efficient. Conway *et. al.* [1] developed algorithms to incrementally update control flow graphs of C programs. Since ARBAC has no control flow, their algorithms are not directly applicable to our problem. All aforementioned work, in contrast to our algorithms, computed the exact data structure. Further, none of them have proposed a lazy algorithm as we do.

7 Conclusion

This paper presents algorithms for incremental user-role reachability analysis of ARBAC policies. The incremental algorithms identify changes to the policy that may affect the analysis result and use the information computed in the previous analysis to update the result. We have evaluated our incremental algorithms using a set of randomly generated ARBAC policies and a set of operations that change the policies. Our experimental data show that our incremental algorithms outperform the non-incremental algorithm in terms of execution time. We will further optimize the incremental analysis algorithms. A promising optimization is not to perform operations, which do not affect the analysis result, on the graph. Such operations include operations that remove irrelevant roles from the graph and operations that change visible transitions to invisible transitions.

Acknowledgement. The authors thank Jian He, Jian Zhang and Arunpriya somasundaram for their contributions to the implementation.

References

1. Conway, C., Namjoshi, K., Dams, D., Edwards, S.: Incremental algorithms for interprocedural analysis of safety properties. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 449–461. Springer, Heidelberg (2005)

2. Crampton, J.: Authorizations and antichains, ph.d. thesis, university of london (2002)
3. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: International Conference on Software Engineering (ICSE), pp. 196–205 (2005)
4. Gupta, A., Katiyar, D., Mumick, I.S.: Counting solutions to the view maintenance problem. In: Workshop on Deductive Databases, pp. 185–194 (1992)
5. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In: International Conference on Management of Data, pp. 157–166 (1993)
6. Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: Protection in operating systems. Communications of the ACM 19(8), 461–471 (1976)
7. Jackson, D., Schechter, I., Shlyakhter, I.: Alcoa: the alloy constraint analyzer, pp. 730–733 (June 2000)
8. Jha, S., Li, N., Tripunitara, M., Wang, Q., Winsborough, W.: Towards formal verification of role-based access control policies. IEEE Transactions on Dependable and Secure Computing 5(2) (2008)
9. Jha, S., Reps, T.: Model-checking SPKI-SDSI. Journal of Computer Security 12, 317–353 (2004)
10. Li, N., Tripunitara, M.V.: Security analysis in role-based access control. ACM Transactions on Information and System Security 9(4), 391–420 (2006)
11. Lu, J., Moerkotte, G., Schu, J., Subrahmanian, V.S.: Efficient maintenance of materialized mediated views (1995)
12. Sandhu, D.F.F.R., Kuhn, D.R.: The NIST model for role based access control: Towards a unified standard. In: ACM SACMAT, pp. 47–63 (2000)
13. Saha, D., Ramakrishnan, C.R.: Incremental evaluation of tabled logic programs. In: International Conference on Logic Programming, pp. 392–406 (2003)
14. Sandhu, R.: The typed access matrix model. In: Proc. IEEE Symposium on Security and Privacy, pp. 122–136 (1992)
15. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and Systems Security 2(1), 105–135 (1999)
16. Sasturkar, A., Yang, P., Stoller, S.D., Ramakrishnan, C.: Policy analysis for administrative role based access control. In: 19th IEEE Computer Security Foundations Workshop (2006)
17. Schaad, A., Moffett, J.D.: A lightweight approach to specification and analysis of role-based access control extensions. In: Proc.of SACMAT, pp. 13–22 (2002)
18. Stoller, S., Yang, P., Ramakrishnan, C.R., Gofman, M.: Efficient policy analysis for administrative role based access control. In: ACM CCS, pp. 445–455 (2007)

An Authorization Framework Resilient to Policy Evaluation Failures

Jason Crampton[1] and Michael Huth[2]

[1] Information Security Group, Royal Holloway, University of London
[2] Department of Computing, Imperial College London

Abstract. In distributed computer systems, it is possible that the evaluation of an authorization policy may suffer unexpected failures, perhaps because a sub-policy cannot be evaluated or a sub-policy cannot be retrieved from some remote repository. Ideally, policy evaluation should be resilient to such failures and, at the very least, fail "gracefully" if no decision can be computed. We define syntax and semantics for an XACML-like policy language. The semantics are incremental and reflect different assumptions about the manner in which failures can occur. Unlike XACML, our language uses simple *binary* operators to combine sub-policy decisions. This enables us to characterize those few binary operators likely to be used in practice, and hence to identify a number of strategies for optimizing policy evaluation and policy representation.

1 Introduction

Many access control models and systems are *policy-based*, in the sense that a request for access to protected resources is evaluated with respect to a policy that defines which requests are authorized. Many languages have been proposed for the specification of authorization policies, perhaps the best known being XACML [4,8,12]. However, it is generally acknowledged that XACML suffers from having poorly defined and counterintuitive semantics, see e.g. [9,10]. More formal approaches have provided well-defined semantics and typically use "policy operators" to construct complex policies from simpler sub-policies [3,5,13].

Each component of an XACML policy has a so-called target, and a policy is applicable to a request only if said request "matches" that policy's target. XACML was designed to operate in heterogeneous, distributed environments, and XACML "policies" (technically, `<PolicySet>` elements) may reference sub-policies (`<Policy>` or `<PolicySet>` elements) that may be held in remote repositories. In addition to returning the usual allow and deny decisions, the result of evaluating an XACML policy may be "not applicable" or "indeterminate", the latter since evaluations in open, distributed systems may fail.

There are three practical drawbacks to existing, more formal algebraic approaches to policy languages: first, it becomes difficult to answer the question "Is this request authorized?", which is central to any access-control mechanism; second, it is difficult to see how practical policies can be written in this way; and finally, no means of handling policy evaluation failures has been provided.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 472–487, 2010.

Our goal in this paper is to develop a practical authorization language that is resilient to authorization evaluation failures, supports different assumptions for when failures may occur, has rigorous semantics, and leads to optimization of policy evaluation and policy representation. As a side effect, our framework allows for a simple static analysis that, at times, can fully recover from evaluation failures. The contributions of our paper include

- the definition of a simple policy language, which introduces the notion of resolution function for possible decisions;
- a concise characterization of the most commonly used decision-combining algorithms as *binary* decision-combining operators;
- the identification of two important classes of decision-combining operators, and a discussion of their significance for policy specification and evaluation;
- three different and successively more robust semantics for policy evaluation and a characterization of the failures with which they can cope;
- a description of how our semantics can be implemented and a discussion of optimizations that can significantly simplify the evaluation of policies.

To reiterate, our framework presented here combines the rigor of the work on policy algebras (which tends to use binary operators to compose policies and bottom-up semantics but does not consider evaluation failures or implementation) with the practicality of the work on XACML and related languages (which tends to use decision-combining algorithms and top-down policy evaluation but lacks the rigor of the work on policy algebras). In other words, our contributions synthesize and extend existing approaches to the specification and evaluation of authorization languages whilst also dealing with evaluation failures.

In the next section, we define our policy language and discuss how binary operators can be used to implement decision-combining functions. In Sect. 3, we define our policy semantics. In Sect. 4, we explain how our formal semantics can be realized in practice, describe our techniques for optimizing policy evaluation, and discuss the implications and potential applications of these techniques. We conclude with a summary of the paper and some discussion of future work in Sect. 5.

2 A Simple Policy Language

The language we use is rather similar to (core) XACML: policies are built from other policies, and policies may or may not be applicable to requests. However, our language is much simpler syntactically, although no less expressive.

We assume that the decisions that may arise from policy evaluation are given by the set $D = \{\mathsf{a}, \mathsf{d}, \bot\}$, denoting "allow","deny" and "not applicable", respectively. We assume that two decisions may be combined using any one of the possible binary operators $\oplus$ of the form $\oplus : D \times D \to D$. We will write $\oplus$ using infix notation: that is, we prefer $x \oplus y$ to $\oplus(x, y)$.

Policy Syntax. *Atomic policies* have the form (π, a, ϕ) or (π, d, ϕ), where π is used to determine the applicability of the policy and ϕ is a *possible-decisions resolution-function* of type $2^D \to 2^D$. If p_1 and p_2 are policies, then $(\pi, p_1, p_2, \oplus, \phi)$ is a policy, where $\oplus : D \times D \to D$ is a *sub-decision combining-operator*. Henceforth, we will usually refer to ϕ as a resolution function and $\oplus$ as a decision operator. We describe resolution functions and decision operators in more detail later in this section.

Policy Applicability. Informally, when evaluating a request q with respect to some policy p, we first determine whether p is applicable to q. The role of π in our policy language is similar to that of `<Target>` and `<Condition>` elements in XACML rules and policies [12], or of access predicates in [6]. We refer to π as the *applicability predicate*.[1]

Hence, to build an access control mechanism, we need a language for defining the applicability predicate and a method for evaluating whether a request satisfies the predicate. In XACML, for example, the syntax for defining `<Target>` and `<Condition>` elements forms part of the core language and the evaluation method forms part of the implementation of the policy decision point (PDP). While these issues are certainly important, the concern of this paper is how to evaluate policies under the assumption that such tools are available.

Decision Operators. We assume that any policy either has no "child" policies (as in policies of the form (π, x, ϕ), where $x \in \{\mathsf{a}, \mathsf{d}\}$), or exactly two child policies. Equivalently, we assume that all decision operators are binary operators. There are two main reasons for this choice. First, the three most common methods used to combine a set of decisions – allow-overrides, deny-overrides and first-applicable [12] – can all be realized using binary operators. Consider, for example, the family of allow-overrides functions $\mathsf{AO}_n : D^n \to D$, $n \geqslant 2$, where

$$\mathsf{AO}_n(x_1, \ldots, x_n) = \begin{cases} \mathsf{a} & \text{if } x_i = \mathsf{a} \text{ for some } i, \\ \bot & \text{if } x_i = \bot \text{ for all } i, \\ \mathsf{d} & \text{otherwise.} \end{cases}$$

Then it is easy to see that for any $n \geqslant 2$, $\mathsf{AO}_n(x_1, \ldots, x_n) = (\ldots(x_1 \vee x_2) \vee \ldots x_n)$, where $\vee$ is defined below in Fig. 1. Similar results hold for deny-overrides and first-applicable. Second, it is very simple to characterize binary operators and this provides many opportunities for optimizing policy evaluation (as we shall see in Sect. 4). We classify operators using the following definitions.

Definition 1. *Let $\oplus : D \times D \to D$ be a decision operator.*

- *If $x \oplus x = x$ for all $x \in D$, then we say $\oplus$ is* idempotent.
- *If $x \oplus \bot = x = \bot \oplus x$ for all $x \in D$, then we say $\oplus$ is a* $\cup$-operator.
- *If $x \oplus \bot = \bot = \bot \oplus x$ for all $x \in D$, then we say $\oplus$ is an* $\cap$-operator.
- *We say $\oplus$ is* well-behaved *if it is either a* $\cup$- *or an* $\cap$-*operator.*

[1] If one were to use XACML syntax in order to define our applicability predicate, then our atomic policies would be analogous to XACML rules, and policies of the form $(\pi, p_1, p_2, \oplus, \phi)$ would be analogous to XACML policies and policy sets.

Idempotent operators are a natural choice, as idempotency is expected when composing access-control decisions. In total, there are $3^6 = 729$ possible idempotent, binary operators. However, far fewer operators are of practical interest. In Sect. 4, we consider how restricting attention to idempotent, well-behaved operators can considerably simplify policy evaluation.

An idempotent, well-behaved decision operator is uniquely defined by the choices of $x \oplus \bot$, $\mathsf{a} \oplus \mathsf{d}$ and $\mathsf{d} \oplus \mathsf{a}$. If we assume that $\oplus$ is commutative, then there are only six possible choices for $\oplus$ (and only four if we assume that $\mathsf{a} \oplus \mathsf{d} \in \{\mathsf{a}, \mathsf{d}\}$). The decision tables for two of the four commutative, idempotent, well-behaved binary operators such that $\mathsf{a} \oplus \mathsf{d} \in \{\mathsf{a}, \mathsf{d}\}$ are shown Fig. 1 as $\vee$ and $\wedge$. As we noted above, the operator $\vee$ has the same effect as the allow-overrides policy-combining algorithm in XACML, while $\wedge$ has the same effect as the deny-overrides algorithm.[2] If $\oplus$ is not commutative, then there are 18 possible choices for $\oplus$ (and eight choices if $\mathsf{a} \oplus \mathsf{d} \in \{\mathsf{a}, \mathsf{d}\}$ and $\mathsf{d} \oplus \mathsf{a} \in \{\mathsf{a}, \mathsf{d}\}$).

The other binary operators shown in Fig. 1 are: $\vee'$ and $\wedge'$, the $\cap$-operator analogues of $\vee$ and $\wedge$; and the non-commutative, "first-applicable", $\cup$-operator $\rhd$, which returns the first conclusive decision (a or d).

$\wedge$	a	d	$\bot$
a	a	d	a
d	d	d	d
$\bot$	a	d	$\bot$

$\vee$	a	d	$\bot$
a	a	a	a
d	a	d	d
$\bot$	a	d	$\bot$

$\wedge'$	a	d	$\bot$
a	a	d	$\bot$
d	d	d	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\vee'$	a	d	$\bot$
a	a	a	$\bot$
d	a	d	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\rhd$	a	d	$\bot$
a	a	a	a
d	d	d	d
$\bot$	a	d	$\bot$

Fig. 1. Decision tables for some binary operators

Resolution Functions. In Sect. 3 we define three different semantics for policies, which specify how a policy should be evaluated. Two of these evaluation methods handle exceptional events by considering different possible outcomes, which leads to the possibility of policy evaluation returning a set of possible decisions, rather than a single decision, as is more usual in access-control mechanisms. The express purpose of the resolution function ϕ is to modify the set of possible outcomes.

In many cases, ϕ will be the identity function ι, where $\iota(X) = X$ for all $X \subseteq D$. We will simply omit ϕ if $\phi = \iota$ (as will be the case in most subsequent examples). However, we would expect that the top-level policy would define ϕ so that for all $X \subseteq D$, $\phi(X) = \{x\}$ for some $x \in \{\mathsf{a}, \mathsf{d}, \bot\}$. In other words, evaluation of the top-level policy always results in a single response.

As for decision operators, very few resolution functions will be of practical relevance. For a policy $(\pi, p_1, p_2, \oplus, \phi)$, we might expect ϕ to be "semantically related" to $\oplus$: if $\oplus$ is $\vee$ (allow-overrides), for example, we might define $\phi(X) = \{\mathsf{a}\}$ if $\mathsf{a} \in X$ and $\phi(X) = X$ otherwise. However, it must be stressed that X represents a set of possible outcomes and (even when $\oplus$ equals $\vee$) it is probably prudent to be conservative and define

[2] Interpreting a as 1 and d as 0, $x \wedge y$ is analogous to logical AND (when $x, y \in \{\mathsf{a}, \mathsf{d}\}$) and $\vee$ is analogous to logical OR.

$$\phi(X) = \begin{cases} \{\mathsf{d}\} & \text{if } \mathsf{d} \in X, \\ \{\bot\} & \text{if } \bot \in X, \\ \{\mathsf{a}\} & \text{otherwise.} \end{cases}$$

Policy Trees. A policy tree is a convenient way of visualizing a policy and can be constructed recursively from a policy. A policy of the form $(\pi, p_1, p_2, \oplus, \phi)$ has a tree with root node $(\pi, \oplus, \phi)$ and two child sub-trees p_1 and p_2. A policy of the form (π, x, ϕ), where $x \in \{\mathsf{a}, \mathsf{d}\}$, is a leaf node (π, x, ϕ). Consider, for example, the policy

$$(\pi_5, (\pi_3, (\pi_1, \mathsf{a}), (\pi_2, \mathsf{d}), \wedge), (\pi_4, \mathsf{a}), \vee),$$

whose policy tree is shown in Fig. 5(a). Henceforth, we will tend to use this tree representation of policies.

3 Policy Semantics

Policies are used to evaluate whether an access request is authorized. When a policy is evaluated, one first checks whether the policy is applicable to the request, which will be determined by the request and π. Under normal circumstances, the evaluation of the applicability of a policy returns either true or false.

However, if we wish to account for exceptional circumstances – perhaps it is not possible to retrieve certain information due to communication, software or hardware failures, or perhaps the request is malformed – then it may not be possible to evaluate some component of a policy. As we noted above, it is natural to then consider the possible outcomes that *could* have arisen from evaluating the policy. We use a resolution function ϕ to combine these possible outcomes.

The evaluation of policy $p = (\pi, p_1, p_2, \oplus, \phi)$ at request q is determined by

- the applicability of the policy to q (π);
- the evaluation of the sub-policies of p (p_1 and p_2) at q;
- the method by which evaluation results of the sub-policies are combined ($\oplus$);
- the combination of different possible evaluations of p (using ϕ).

We wish to account for indeterminacy that might arise in the evaluation of policy applicability and in the retrieval of policies. To this end, we consider the following possibilities.

1. Normal evaluation, where all policy components can be retrieved and the applicability of all sub-policies can be determined.
2. Indeterminate applicability of sub-policies, where all sub-policies can be retrieved, but the applicability of a sub-policy may be impossible to determine.
3. Indeterminate applicability or non-retrievability of some sub-policies.

The second and third items account for differing types of exceptional behavior that might occur during policy evaluation. These differences are reflected in the evaluation of the parent policy.

In the presence of indeterminacy, we adopt a conservative evaluation strategy and consider all possible outcomes. If the applicability of a sub-policy cannot be determined, then we consider two possibilities when evaluating the parent policy: that the sub-policy was applicable and that the sub-policy was not applicable. If a sub-policy cannot be retrieved, then we consider three possibilities: that the sub-policy was applicable and returned a, that the sub-policy was applicable and returned d, and that the sub-policy was not applicable. We use ϕ to modify the set of possible outcomes: ϕ could, for example, reduce a set of two or more possible outcomes to a single outcome.

Below, we treat the three assumptions on evaluation failures separately, but we prove that in each case the semantics are extended in such a way that the semantics for simpler assumptions are preserved (see Proposition 1, for example).

Our technical development assumes the existence of an evaluation function e that determines whether policy p is applicable to request q. We define three methods of evaluation for policies, corresponding to the failure assumptions identified above. We refer to them as Type 1, 2 and 3 semantics, respectively.

For a policy p, we write $[[p]]_i(q)$ to mean the result of evaluating p at point q using Type i semantics. We write $[[p]]_i = [[p']]_i$ if and only if for all requests q, we have $e(p,q) = e(p',q)$ and $[[p]]_i(q) = [[p']]_i(q)$.

Type 1 Semantics. In this case, we assume that for all policies p and all requests q, either $e(p,q) = \mathsf{true}$ or $e(p,q) = \mathsf{false}$. Henceforth, we write t and f for true and false, respectively.

Our Type 1 semantics is depicted in Fig. 2. An alternative form of the same semantics, explained in the next section, is given in Fig. 3(a).

$$[[(\pi, p_1, p_2, \oplus), \phi]]_1(q) = \begin{cases} [[p_1]]_1(q) \oplus [[p_2]]_1(q) & \text{if } e(p,q) = \mathsf{t}, \\ \bot & \text{otherwise;} \end{cases}$$

$$[[(\pi, x, \phi)]]_1(q) = \begin{cases} x & \text{if } e(p,q) = \mathsf{t} \text{ and } x \in \{\mathsf{a}, \mathsf{d}\}, \\ \bot & \text{otherwise.} \end{cases}$$

Fig. 2. Type 1 semantics

Note that for all policies p and all requests q, we have that $[[p]]_1(q) \in \{\mathsf{a}, \mathsf{d}, \bot\}$. This can be proved by a simple induction on the depth of the policy tree.

Type 2 Semantics. For ease of exposition (and to aid implementation), we introduce a "dummy" policy that is applicable to every request: if p is a policy, then $\hat{p}$ is a policy that is identical to p except that π is replaced with the reserved word t. Hence:

- if $p = (\pi, x, \phi)$, where $x \in \{\mathsf{a}, \mathsf{d}\}$, then $\hat{p} \stackrel{\text{def}}{=} (\mathsf{t}, x, \phi)$;
- if $p = (\pi, p_1, p_2, \oplus, \phi)$, then $\hat{p} \stackrel{\text{def}}{=} (\mathsf{t}, p_1, p_2, \oplus, \phi)$.

By definition, $e(\hat{p}, q) = \mathsf{t}$ for all policies p and all requests q.

Type 2 policy semantics are presented in Fig. 3(b). Note that we can also define Type 1 semantics for p using $\hat{p}$ as shown in Fig. 3(a). The uniform presentation of the semantics in Fig. 3 illustrates how Type 2 semantics are related to Type 1, and how Type 3 are related to Type 2.

In defining Type 2 semantics, we do not assume that $e(p,q)$ takes a unique value in $\{\mathsf{t},\mathsf{f}\}$. Hence, we must provide a method of evaluating p if $e(p,q) \neq \mathsf{t}$ and $e(p,q) \neq \mathsf{f}$. In this case, we consider two possible evaluations of the policy tree: one when the policy is applicable and one when the policy is not (when the response is $\perp$). In Fig. 3(b), we see that $[[p]]_2(q)$ introduces a third option to explicitly handle this possibility.

Now, of course, the evaluation of a policy may return a set of possible responses, rather than a single response. Hence $[[p]]_2$ returns a set of responses, to which the resolution function ϕ is applied.

Note that for Type 2 and Type 3 semantics, we have assumed that for all ϕ and all $x \in D$, $\phi(\{x\}) = \{x\}$, the intuition being that if the evaluation of a

$$[[p]]_1(q) = \begin{cases} [[\hat{p}]]_1(q) & \text{if } e(p,q) = \mathsf{t}, \\ \perp & \text{otherwise;} \end{cases}$$

$$[[\hat{p}]]_1(q) = \begin{cases} [[p_1]]_1(q) \oplus [[p_2]]_1(q) & \text{if } p = (\pi, p_1, p_2, \oplus, \phi), \\ x & \text{if } p = (\pi, x, \phi),\ x \in \{\mathsf{a},\mathsf{d}\}. \end{cases}$$

(a) Type 1

$$[[p]]_2(q) = \begin{cases} \{[[\hat{p}]]_2(q)\} & \text{if } e(p,q) = \mathsf{t}, \\ \{\perp\} & \text{if } e(p,q) = \mathsf{f}, \\ \phi(\{\perp\} \cup \{[[\hat{p}]]_2(q)\}) & \text{otherwise;} \end{cases}$$

$$[[\hat{p}]]_2(q) = \begin{cases} \phi(\{x \oplus y : x \in [[p_1]]_2(q), y \in [[p_2]]_2(q)\}) & \text{if } p = (\pi, p_1, p_2, \oplus, \phi), \\ \{x\} & \text{if } p = (\pi, x, \phi),\ x \in \{\mathsf{a},\mathsf{d}\}. \end{cases}$$

(b) Type 2

$$[[p]]_3(q) = \begin{cases} \{\mathsf{a},\mathsf{d},\perp\} & \text{if } p \text{ cannot be retrieved}, \\ \{[[\hat{p}]]_3(q)\} & \text{if } e(p,q) = \mathsf{t}, \\ \{\perp\} & \text{if } e(p,q) = \mathsf{f}, \\ \phi(\{\perp\} \cup \{[[\hat{p}]]_3(q)\}) & \text{otherwise;} \end{cases}$$

$$[[\hat{p}]]_3(q) = \begin{cases} \phi(\{x \oplus y : x \in [[p_1]]_3(q), y \in [[p_2]]_3(q)\}) & \text{if } p = (\pi, p_1, p_2, \oplus, \phi), \\ \{x\} & \text{if } p = (\pi, x, \phi),\ x \in \{\mathsf{a},\mathsf{d}\}. \end{cases}$$

(c) Type 3

Fig. 3. Three types of policy semantics corresponding to our three failure assumptions

policy returns a single outcome, then ϕ should return that outcome unmodified. Then Type 2 semantics are an extension of Type 1 semantics, and preserve Type 1 semantics in the following sense.

Proposition 1. *Let p be any policy comprising sub-policies $p_1, \ldots, p_k$ and let $e(p,q) \in \{\mathsf{t}, \mathsf{f}\}$ and $e(p_i, q) \in \{\mathsf{t}, \mathsf{f}\}$ for all i. Then $[[p]]_2(q) = \{[[p]]_1(q)\}$.*

Proof. By induction on the depth of the policy tree. Consider the base case when the tree has depth 1, whence $p = (\pi, x, \phi)$ for some $x \in \{\mathsf{a}, \mathsf{d}\}$. Then $[[p]]_1(q) = x$ if $e(p,q) = \mathsf{t}$ and $[[p]]_1(q) = \bot$ otherwise. Now, by assumption, $e(p,q)$ is known. Therefore, $[[p]]_2(q) = [[\hat{p}]]_2(q) = \{x\}$ if $e(p,q) = \mathsf{t}$, and $[[p]]_2(q) = \{\bot\}$ if $e(p,q) = \mathsf{f}$. Hence the result holds if the tree has depth 1.

Now suppose that the result holds for all trees of depth less than or equal to n and suppose the tree for $p = (\pi, p_1, p_2, \oplus, \phi)$ has depth $n+1$. Then, by assumption, $e(p,q), e(p_1,q), e(p_2,q) \in \{\mathsf{t}, \mathsf{f}\}$. Moreover,

$$[[p]]_1(q) = \begin{cases} [[p_1]]_1(q) \oplus [[p_2]]_1(q) & \text{if } e(p,q) = \mathsf{t}, \\ \bot & \text{otherwise.} \end{cases}$$

Now let us consider $[[p]]_2(q)$. If $e(p,q) = \mathsf{t}$, then $[[p]]_2(q)$ equals $[[\hat{p}]]_2(q) = \phi(\{x \oplus y : x \in [[p_1]]_2(q), y \in [[p_2]]_2(q)\})$. By the inductive hypothesis $[[p_1]]_2(q) = \{[[p_1]]_1(q)\}$ and $[[p_2]]_2(q) = \{[[p_2]]_1(q)\}$. Hence,

$$\begin{aligned} [[p]]_2(q) &= \phi(\{x \oplus y : x \in [[p_1]]_2(q), y \in [[p_2]]_2(q)\}) \\ &= \phi(\{[[p_1]]_2(q) \oplus [[p_2]]_2(q)\}) \\ &= \{[[p_1]]_2(q) \oplus [[p_2]]_2(q)\} \\ &= \{[[p]]_1(q)\} \end{aligned}$$

Alternatively, if $e(p,q) = \mathsf{f}$ then $[[p]]_2(q) = \{\bot\} = \{[[p]]_1(q)\}$. Hence, the result follows by induction. □

In other words, if the applicability of all component policies can be determined, then the evaluation of p with respect to Type 2 semantics returns a unique response which is that obtained by using Type 1 semantics.

Type 3 Semantics. In this case, we do not assume that we can always retrieve a sub-policy, so it may be the case that we have no sub-policy to evaluate. We can still attempt to evaluate the root policy by considering all possible responses that could be returned by that sub-policy. This is reflected in the Type 3 semantics illustrated in Fig. 3(c), where the evaluation of a policy p simply returns the set $\{\mathsf{a}, \mathsf{d}, \bot\}$ if p cannot be retrieved.

It is very easy to see that if all policies can be retrieved, the evaluation of a policy will be the same whether Type 2 or Type 3 semantics are used. More formally, we have the following result.

Proposition 2. *Let p be any policy comprising sub-policies $p_1, \ldots, p_k$ and suppose that it has been possible to retrieve all $p, p_1, \ldots, p_k$. Then $[[p]]_3(q) = [[p]]_2(q)$.*

Proof. Since we assume that all policies are retrievable, the result follows directly from the definitions of Type 2 and Type 3 semantics. □

4 Policy Evaluation

In this section, we consider several evaluation strategies that realize the semantics defined in the previous section. We first present a simple algorithm that can be used to implement Type 1 and Type 2 semantics.

Naïve Algorithm. We can implement Type 2 semantics directly using an algorithm of the form shown in Fig. 4. We assume that a tree representation of the entire policy can always be constructed. (In other words, Type 2 semantics are sufficient to derive a decision.) The function *evaluateTree*(,) takes a pointer to the root of the policy tree and a request and returns the set of possible authorization decisions for that request with respect to the policy tree. The function *evaluateApplicability*(,) determines whether a policy is applicable to a request (in other words, it is a realization of the function e used in the previous section), taking a policy and a request as input and returning t, f or neither.

We assume each node in a policy tree has form $(\pi, lptr, rptr, effect, \phi)$, where *effect* may be a decision a or d or it may be a decision operator $\oplus$. Hence, we model a policy of the form (π, a, ϕ), for example, as $(\pi, \mathsf{null}, \mathsf{null}, \mathsf{a}, \phi)$, and $(\pi, p_1, p_2, \oplus, \phi)$ as $(\pi, lptr, rptr, \oplus, \phi)$. In an attempt to keep the pseudo-code easy to read, we refer directly to the components of a node, so we write π in preference to $p.\pi$ or $p \to \pi$, for example.

```
[Inputs: pointer to policy tree p; request q]
[Outputs: set of decisions]
  evaluateTree(p, q)
    if (π == t) then
        if (lptr == null) and (rptr == null) then
            return {effect}
        else
            X = evaluateTree(lptr, q)
            Y = evaluateTree(rptr, q)
            result = ∅
            for all x ∈ X
                for all y ∈ Y
                    result = result ∪ {x ⊕ y}
            return φ(result)
    else
        if (evaluateApplicability(π, q) == t) then
            π = t
            evaluateTree(p, q)
        else-if (evaluateApplicability(π, q) == f) then
            return {⊥}
        else
            π = t
            return φ({⊥} ∪ evaluateTree(p, q))
```

Fig. 4. A possible implementation of Type 2 semantics

Let us now consider the evaluation of the policy illustrated in Fig. 5(a), where the operators $\wedge$ and $\vee$ are as defined in Fig. 1. We will write p_i to refer to the

policy (sub-)tree with root $(\pi_i, \oplus_i)$. (The indices assigned to the node identifiers correspond to a post-order traversal [1] of the tree.)

An *evaluation tree* (for request q) is obtained by labeling each node of the policy tree with its applicability (with respect to q) and its response.

Let us now consider the effect of evaluating a request for which we cannot decide whether certain policies are applicable or not. First, if $e(p_2, q) \neq \mathsf{t}$ and $e(p_2, q) \neq \mathsf{f}$, with all other policies being applicable, then p_2 returns the set of (possible) responses $\{\mathsf{d}, \bot\}$. Then $\mathsf{a} \wedge \bot = \mathsf{a}$ and $\mathsf{a} \wedge \mathsf{d} = \mathsf{d}$, which means that p_3 returns $\{\mathsf{d}, \bot\}$. Hence, p_5 returns $\{\mathsf{a}\}$. This example is illustrated in Fig. 5(b).

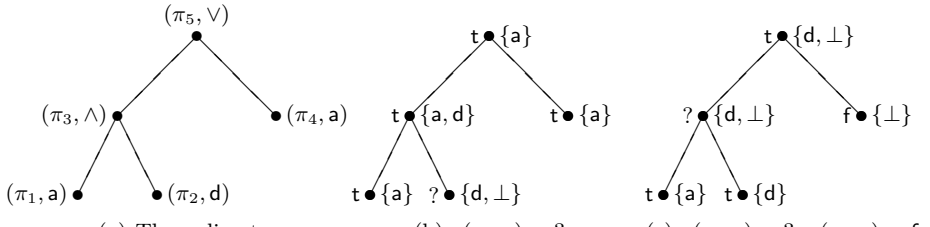
Finally, suppose that $e(p_3, q) \neq \mathsf{t}$ and $e(p_3, q) \neq \mathsf{f}$, the applicability of other policies being shown in Fig. 5(c). Then we evaluate p_1 and p_2 and combine the results using $\wedge$ to obtain d. To this we add the response $\bot$ to account for the possibility that p_3 may not have been applicable. Hence, the set of possible responses for p_3 is $\{\mathsf{d}, \bot\}$. If $e(p_4, q) = \mathsf{f}$ (as shown in Fig. 5(c)), then p_5 returns $\{\mathsf{d}, \bot\}$. If $e(p_4, q) = \mathsf{t}$ (not illustrated), then p_5 returns $\{\mathsf{a}\}$.

Well-Behaved Operators. A policy $p = (\pi, p_1, p_2, \oplus, \phi)$ may not return a conclusive decision (a or d) even if p is applicable, because neither p_1 nor p_2 may be applicable. There are two standard interpretations of what might be termed the "effective applicability" of a policy $p = (\pi, p_1, p_2, \oplus, \phi)$.

1. One is to regard p as being effectively applicable to every request for which p is applicable and at least one of p_1 or p_2 is applicable, as in XACML.
2. The other is to regard p as being applicable to a request only if p, p_1 and p_2 are all applicable.

In the first case, $[[p]](q)$ is defined if $e(p, q) = \mathsf{t}$ and either $e(p_1, q) = \mathsf{t}$ or $e(p_2, q) = \mathsf{t}$. In the second case, $[[p]](q)$ is defined if $e(p, q) = e(p_1, q) = e(p_2, q) = \mathsf{t}$. This interpretation appears in several papers on "policy algebras" (see [5], for example). In both cases, if the policy p is applicable to request q, then the decision returned by the policy is $[[p_1]] \oplus [[p_2]]$.

In fact, each of these interpretations can be realized provided $\oplus$ is chosen appropriately. If we want the first interpretation, then we ensure that for all $x \in \{\bot, \mathsf{a}, \mathsf{d}\}$, $x \oplus \bot = \bot \oplus x = x$. That is, $\oplus$ is a $\cup$-operator. A $\cup$-operator effectively ignores all $\bot$ values, ensuring that p will return a value whenever at least one of p_1 or p_2 is applicable. All the standard policy-combining algorithms



(a) The policy tree (b) $e(p_2, q) = ?$ (c) $e(p_3, q) = ?$; $e(p_4, q) = \mathsf{f}$

Fig. 5. Policy and evaluation trees for policy $(\pi_5, (\pi_3, (\pi_1, \mathsf{a}), (\pi_2, \mathsf{d}), \wedge), (\pi_4, \mathsf{a}), \vee)$

in XACML [12] have this behavior. If $\oplus$ is a $\cup$-operator, then we say any p of the form $(\cdot, \cdot, \cdot, \oplus, \cdot)$ is a $\cup$-*policy*.

If we want the second interpretation, then we ensure that for all x, $x \oplus \bot = \bot \oplus x = \bot$. That is, $\oplus$ is a $\cap$-operator, which has the effect of returning $\bot$ whenever at least one of p_1 or p_2 is not applicable. If $\oplus$ has this property, then we say any p of the form $(\cdot, \cdot, \cdot, \oplus, \cdot)$ is an $\cap$-*policy*.

Optimizing Policy Evaluation. We now show how a policy evaluation tree can be pruned, *without* changing its meaning, when the decision operators are known to be well-behaved (and all sub-policies can be retrieved). Suppose that p uses the identity resolution function, and so $p = (\pi, p_1, p_2, \oplus)$. Table 1 illustrates the evaluation of p given the applicability of p, p_1 and p_2, and the nature of $\oplus$. Abusing notation slightly, we write $[[p_1]] \oplus [[p_2]]$ for $\{x_1 \oplus x_2 : x_1 \in [[p_1]], x_2 \in [[p_2]]\}$. We write "$-$" to denote that the applicability of a sub-policy is irrelevant to the evaluation of p.

Table 1. Optimized evaluation of $p = (\pi, p_1, p_2, \oplus)$ when $\oplus$ is well-behaved

Applicability			$[[p]]$	
p	p_1	p_2	**$\cup$-operator**	**$\cap$-operator**
t	t	t	$[[p_1]] \oplus [[p_2]]$	$[[p_1]] \oplus [[p_2]]$
t	t	f	$[[p_1]]$	$\{\bot\}$
t	f	t	$[[p_2]]$	$\{\bot\}$
t	f	f	$\{\bot\}$	$\{\bot\}$
t	t	?	$([[p_1]] \oplus [[p_2]]) \cup [[p_1]]$	$\{\bot\} \cup ([[p_1]] \oplus [[p_2]])$
t	?	t	$([[p_1]] \oplus [[p_1]]) \cup [[p_2]]$	$\{\bot\} \cup ([[p_1]] \oplus [[p_2]])$
t	?	?	$([[p_1]] \oplus [[p_2]]) \cup [[p_1]] \cup [[p_2]] \cup \{\bot\}$	$\{\bot\} \cup ([[p_1]] \oplus [[p_2]])$
t	f	?	$\{\bot\} \cup [[p_2]]$	$\{\bot\}$
t	?	f	$\{\bot\} \cup [[p_1]]$	$\{\bot\}$
?	–	–	$\{\bot\} \cup ([[p_1]] \oplus [[p_2]])$	$\{\bot\} \cup ([[p_1]] \oplus [[p_2]])$
f	–	–	$\{\bot\}$	$\{\bot\}$

Given a request, we now assume the applicability of every sub-policy is first evaluated. We can then apply re-writing rules to the policy-evaluation tree on the basis of the applicability of each sub-policy and the semantics shown in Table 1. To illustrate this point, we define and prove the correctness of one such re-write rule in Proposition 3. Similar results exist for the other re-writing rules, but are omitted due to space constraints.

Proposition 3. *Let $p = (\pi, p_1, p_2, \oplus)$, where $\oplus$ is well-behaved, and let q be a request such that $e(p, q) = e(p_1, q) = \mathtt{t}$ and $e(p_2, q) = \mathtt{f}$. Then*

$$[[p]]_2(q) = \begin{cases} \{\bot\} & \text{if } \oplus \text{ is an } \cap\text{-operator,} \\ \{[[p_1]]_2(q)\} & \text{if } \oplus \text{ is a } \cup\text{-operator.} \end{cases}$$

Proof. By definition, $[[p]]_2(q) = \{x \oplus \bot : x \in [[p_1]]_2(q)\}$. If $\oplus$ is an $\cap$-operator, then $x \oplus \bot = \bot$ for all $x \in \{\mathsf{a}, \mathsf{d}, \bot\}$; hence $[[p]]_2(q) = \{\bot\}$. If $\oplus$ is a $\cup$-operator, then $x \oplus \bot = x$ for all $x \in \{\mathsf{a}, \mathsf{d}, \bot\}$; hence $[[p]]_2(q) = [[p_1]]_2(q)$. □

Using these re-writing rules we can simplify the evaluation of a policy considerably. Also, these re-writing rules can be implemented easily using a recursive post-order tree traversal algorithm. To illustrate, consider the policy represented by the tree in Fig. 6(a), where each node has been assigned an identifier of the form p_i to facilitate explanation. (The indices assigned to the node identifiers correspond to a post-order traversal of the tree.) The applicability of each sub-policy for some request q is indicated to the left of each node.

If all decision operators are $\cap$-operators, then we can simplify this evaluation tree for q to a single node comprising a non-applicable policy. This is because p_2 is not applicable (since one of its children is not applicable), which in turn means that p_4 and p_5 are not applicable.

The simplified policy-evaluation tree, when all operators are $\cup$-operators, is shown in Fig. 6(b). The sub-tree rooted at policy p_9 reduces to an evaluation of p_7 and the evaluation of p_7 reduces to an evaluation of p_5. Suppose that the (relevant) leaf policies are $p_0 = (\pi_0, \mathsf{a})$, $p_5 = (\pi_5, \mathsf{a})$ and $p_9 = (\pi_9, \mathsf{d})$. Let $\oplus_i$ denote the decision operator for the policy at node p_i. Now $x \oplus_i \bot = \bot \oplus x_i = x$ for all operators $\oplus_i$ in the policy tree (since, by assumption, $\oplus_i$ is a $\cup$-operator) and assuming that $\oplus_i$ is idempotent, we have $[[p_0]]_2(q) = \{\mathsf{a}\}$, $[[p_5]]_2(q) = \{\bot, \mathsf{a}\}$, $[[p_9]]_2(q) = \{\bot, \mathsf{d}\}$ and $[[p_{11}]]_2(q) = \{\bot, \mathsf{a}, \mathsf{d}, \mathsf{a} \oplus_{11} \mathsf{d}\}$ from which we obtain

$$\begin{aligned}[[p_{12}]]_2(q) &= \{\mathsf{a}, \mathsf{a} \oplus_{12} \mathsf{a}, \mathsf{a} \oplus_{12} \mathsf{d}, \mathsf{a} \oplus_{12} (\mathsf{a} \oplus_{11} \mathsf{d})\} \\ &= \{\mathsf{a}, \mathsf{a} \oplus_{12} \mathsf{d}, \mathsf{a} \oplus_{12} (\mathsf{a} \oplus_{11} \mathsf{d})\}\,.\end{aligned}$$

By specifying $\oplus_{11}$ and $\oplus_{12}$ we can compute $[[p_{12}]]_2(q)$. For example:

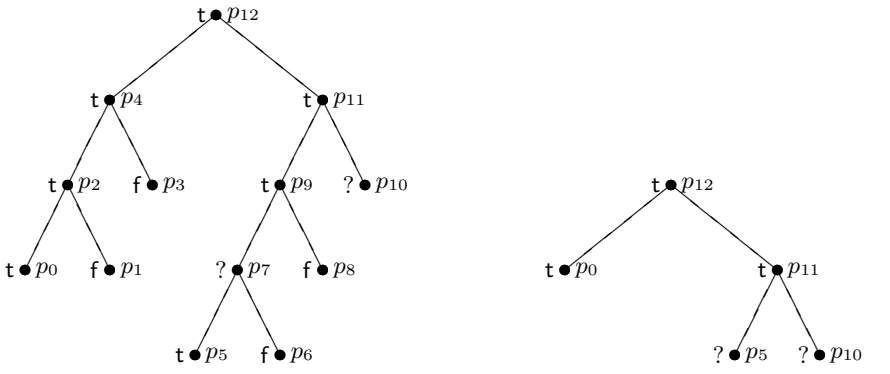
$$[[p_{12}]]_2(q) = \begin{cases} \{\mathsf{a}\} & \text{if } \oplus_{12} = \vee, \\ \{\mathsf{a}, \mathsf{d}\} & \text{if } \oplus_{12} = \wedge. \end{cases}$$

Thus one may derive a conclusive decision for p (namely $\{\mathsf{a}\}$ in the case that $\oplus_{12} = \wedge$) even if the applicability of some sub-policies cannot be determined.

In practice, all decision operators are likely to be well-behaved. Indeed, all the standard policy-combining algorithms in XACML (the equivalent of our decision operators) are $\cup$-operators. Note that we neither require that all operators in the policy tree are $\cup$-operators nor that they are all $\cap$-operators in order to use our re-write rules, simply that they are all well-behaved.

Under the assumption that all decision operators are well-behaved, it is always possible to perform tree re-writing, thereby simplifying policy evaluation. A flag in each policy could indicate if it is a $\cup$- or an $\cap$-policy, thereby indicating how $\oplus$ should treat the $\bot$ value. This provides sufficient information to re-write the evaluation tree and means that we only need to define $x \oplus y$ for $x, y \in \{\mathsf{a}, \mathsf{d}\}$.

Indeed, recalling the discussion in Sect. 2, we can completely specify any idempotent, well-behaved operator with three pieces of information: a flag indicating



(a) Policy tree and applicability

(b) Evaluation tree for ∪-operators

Fig. 6. Policy evaluation by tree re-writing

whether it is a ∪- or ∩-operator, the value of $\mathsf{a} \oplus \mathsf{d}$ and the value of $\mathsf{d} \oplus \mathsf{a}$. This information can be included in the policy definition (rather than providing a pointer to a decision table) and used directly by the *evaluateTree*(,) function.

Partial Evaluation Trees. Type 3 semantics are only relevant when we are unable to build a complete policy tree. Such a situation could arise when policies are not self-contained, in the sense that they may reference sub-policies stored in remote repositories.

Under these operating assumptions, we build an evaluation tree at request evaluation time. This evaluation tree may not be isomorphic to the policy tree, since some policy (that would give rise to sub-trees) may not be retrievable at evaluation time. In constructing the evaluation tree, we label the nodes with an applicability value (if possible) or with the decision set $\{\mathsf{a}, \mathsf{d}, \bot\}$ otherwise.

To illustrate, let us evaluate the policy depicted in Fig. 6 for request q under assumption that we cannot retrieve policy p_9. Then we construct the (partial) evaluation tree shown in Fig. 7(a). If all operators are ∪-operators, we can re-write this evaluation tree to obtain the tree shown in Fig. 7(b).

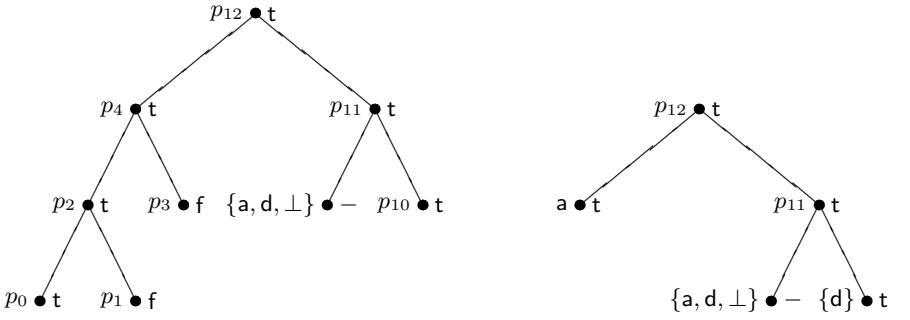
Then we have

$$[[p_{11}]]_3(q) = \{\mathsf{a} \oplus_{11} d, \mathsf{d} \oplus_{11} \mathsf{d}, \bot \oplus_{11} \mathsf{d}\} = \{\mathsf{a} \oplus_{11} \mathsf{d}, \mathsf{d}\};$$
$$[[p_{12}]]_3(q) = \{\mathsf{a} \oplus_{12} (\mathsf{a} \oplus_{11} \mathsf{d}), \mathsf{a} \oplus_{12} \mathsf{d}\}.$$

Assuming that $\oplus_{11}$ and $\oplus_{12}$ belong to $\{\vee, \wedge\}$, it can be shown that $[[p_{12}]]_3(q)$ is a conclusive decision, except when $\oplus_{12} = \wedge$ and $\oplus_{11} = \vee$.

Applications. The fact that we can obtain conclusive results from a partial evaluation of a policy opens up interesting possibilities. We sketch two of them here briefly.

A first application is an access-control architecture in which there are two PDPs: one is co-located with the policy-evaluation point (PEP) and is used to



(a) Sub-policy p_9 could not be retrieved (b) Re-written evaluation tree

Fig. 7. Partial request-time evaluation tree and its rewrite

make rapid decisions where possible, while the other may be remote.[3] The "local" PDP we envisage is provided with a partial representation of an authorization policy that returns a (comparatively) quick response to the PEP. If the response is not conclusive, then the PEP forwards the request to the other PDP which evaluates the full policy tree and returns a decision.

Consider, for example, the policy tree in Fig. 6(a) and suppose that we expect that p_{10} will be applicable to a large percentage of requests. Then we might choose to provide the local PDP with the tree depicted in Fig. 8 (deliberately preventing the local PEP from evaluating the whole policy tree by omitting the relatively complex policies p_4 and p_9).

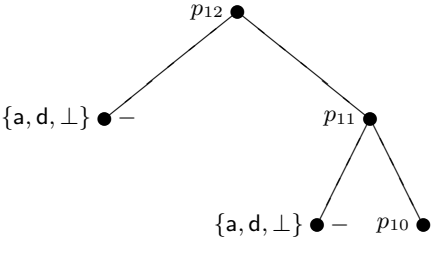


Fig. 8. The policy evaluated by the local PDP

Now suppose that p_{10}, p_{11} and p_{12} are applicable to q. Then

$$[[p_{12}]]_3(q) = \begin{cases} \{\mathsf{a}\} & \text{if } [[p_{10}]]_3(q) = \{\mathsf{a}\} \text{ and } \oplus_{11} = \oplus_{12} = \vee, \\ \{\mathsf{d}\} & \text{if } [[p_{10}]]_3(q) = \{\mathsf{d}\} \text{ and } \oplus_{11} = \oplus_{12} = \wedge, \\ \{\mathsf{a}, \mathsf{d}\} & \text{otherwise.} \end{cases}$$

[3] This architecture is structurally similar to those used for authorization recycling that cache previous authorization decisions at the PEP to improve response times [7].

Clearly, it would be worth providing the local PDP with the reduced policy in Fig. 8 if $\oplus_{11} = \oplus_{12}$ (and it is known that p_{10} is applicable to many requests).

A second application of our authorization framework is to define a PDP that can process multiple requests in a single pass through the evaluation tree. There are many practical instances where it is necessary to decide several different access requests in order to determine whether an attempted subject-object interaction is authorized. Two obvious examples are:

- In Unix, a subject is authorized to access an object only if it is authorized to access every directory (multiple objects) in said object's path name.
- In the stack-walk algorithm used in Java, where it is necessary to check that every subject on the call stack (multiple subjects) is authorized.

In this case, the PDP processes all requests at the same time, treating each of these as possible evaluations of the tree. We introduce the top-level resolution function $\phi_\forall$, where $\phi_\forall(X) = \{\mathsf{a}\}$ if $X = \{\mathsf{a}\}$ and $\phi_\forall(X) = \{\mathsf{d}\}$ otherwise. In contrast, a role-based PDP can evaluate multiple requests, one for each role for which the requester is authorized, and use the $\phi_\exists$ resolution function to compute a final decision, where $\phi_\exists(X) = \{\mathsf{a}\}$ if $\mathsf{a} \in X$ and $\phi_\exists(X) = \{\mathsf{d}\}$ otherwise.

5 Concluding Remarks

We have presented a framework for tree-like authorization policies that are resilient to evaluation failures. This resiliency is achieved by defining three different semantics for those policies, representing three different sets of assumptions about the operational environment in which these failures may occur.

We have provided a succinct characterization of decision operators, which yields numerous opportunities for optimizing policy evaluation and policy representation. Our semantics improve on existing work in enabling policy evaluation to be completed even if it is not possible to recover one or more sub-policies. Our approach is conceptually similar to static analysis [11]. In particular, if our semantics return a conclusive decision, then our over-approximation of decisions is precise (Propositions 1 and 2). Our work also enables the design of efficient PDPs and novel access-control architectures, to be explored in future work.

There are many other ways in which our work could and should be extended. From a technical perspective, it is important to establish whether our language can accommodate a fourth decision value to represent conflicting decisions from sub-policies [6]. Equally important is to establish what set of binary operators would be sufficient to articulate any desired policy. There are clear parallels here with establishing a minimal set of logical connectives that is functionally complete [2]. From a practical perspective, it would be interesting to develop an XML schema for our policy language, perhaps re-using those parts of XACML that are used to specify `<Target>` and `<Condition>` elements, and to develop a PDP that implements our policy semantics.

Acknowledgements. The authors would like to thank the anonymous referees for their comments.

References

1. Aho, A., Hopcroft, J., Ullman, J.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1975)
2. Aireli, O., Avron, A.: The value of the four values. Artificial Intelligence 102, 97–141 (1998)
3. Backes, M., Dürmuth, M., Steinwandt, R.: An algebra for composing enterprise privacy policies. In: Proceedings of the 9th European Symposium on Research in Computer Security, pp. 33–52 (2004)
4. Bertino, E., Castano, S., Ferrari, E.: Author-$\mathcal{X}$: A comprehensive system for securing XML documents. IEEE Internet Computing 5(3), 21–31 (2001)
5. Bonatti, P., Vimercati, S.D.C.D., Samarati, P.: An algebra for composing access control policies. ACM Transactions on Information and System Security 5(1), 1–35 (2002)
6. Bruns, G., Huth, M.: Access-control policies via Belnap logic: Effective and efficient composition and analysis. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, pp. 163–176 (2008)
7. Crampton, J., Leung, W., Beznosov, K.: The secondary and approximate authorization model and its application to Bell-LaPadula policies. In: Proceedings of 11th ACM Symposium on Access Control Models and Technologies (2006)
8. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. ACM Transactions on Information and System Security 5(2), 169–202 (2002)
9. Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., Lin, D.: Access control policy combining: Theory meets practice. In: Proceedings of 14th ACM Symposium on Access Control Models and Technologies, pp. 135–144 (2009)
10. Ni, Q., Bertino, E., Lobo, J.: D-algebra for composing access control policy decisions. In: Proceedings of 2009 ACM Symposium on Information, Computer and Communications Security, pp. 298–309 (2009)
11. Nielson, F., Nielson, H., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
12. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0. In: Moses, T. (ed.) OASIS Committee Specification (2005)
13. Wijesekera, D., Jajodia, S.: A propositional policy algebra for access control. ACM Transactions on Information and System Security 6(2), 286–235 (2003)

Optimistic Fair Exchange with Multiple Arbiters

Alptekin Küpçü and Anna Lysyanskaya

Brown University, Providence, RI, USA
{kupcu,anna}@cs.brown.edu

Abstract. Fair exchange is one of the most fundamental problems in secure distributed computation. Alice has something that Bob wants, and Bob has something that Alice wants. A fair exchange protocol would guarantee that, even if one of them maliciously deviates from the protocol, either both of them get the desired content, or neither of them do. It is known that no two-party protocol can guarantee fairness in general; therefore the presence of a trusted *arbiter* is necessary. In optimistic fair exchange, the arbiter only gets involved in case of faults, but needs to be trusted. To reduce the trust put in the arbiter, it is natural to consider employing multiple arbiters.

Expensive techniques like byzantine agreement or secure multi-party computation with $\Omega(n^2)$ communication can be applied to distribute arbiters in a non-autonomous way. Yet we are interested in efficient protocols that can be achieved by keeping the arbiters autonomous (non-communicating), especially for p2p settings in which the arbiters do not even know each other. Avoine and Vaudenay [6] employ multiple autonomous arbiters in their optimistic fair exchange protocol which uses global timeout mechanisms; all arbiters have access to -loosely- synchronized clocks. They left two open questions regarding the use of distributed autonomous arbiters: (1) Can an optimistic fair exchange protocol without timeouts provide fairness (since it is hard to achieve synchronization in a p2p setting) when employing multiple autonomous arbiters? (2) Can any other optimistic fair exchange protocol with timeouts achieve better bounds on the number of honest arbiters required? In this paper, we answer both questions negatively. To answer these questions, we define a general class of optimistic fair exchange protocols with multiple arbiters, called "distributed arbiter fair exchange" (DAFE) protocols. Informally, in a DAFE protocol, if a participant fails to send a correctly formed message, the other party must contact some subset of the arbiters and get correctly formed responses from them. The arbiters do not communicate with each other, but only to Alice and Bob. We prove that no DAFE protocol can meaningfully exist.

Keywords: Optimistic fair exchange, distributed arbiters, trusted third party.

1 Introduction

Optimistic fair exchange is a very useful primitive in distributed system design with many applications including contract signing, electronic commerce, or even peer-to-peer file sharing [2,3,4,5,7,8,15,18,19,20]. In a fair exchange protocol, Alice and Bob want to exchange some items, and they want to do so fairly. Fairness intuitively refers to Alice getting Bob's item and Bob getting Alice's item at the end of the protocol, or neither of them getting anything, even if one of them maliciously deviates from the

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 488–507, 2010.

protocol. For technical definitions of optimistic fair exchange protocols, we refer the reader to [18].

It has been shown that no general fair exchange protocol can provide complete fairness without a trusted entity [21], called the *arbiter*. In an optimistic fair exchange protocol, the arbiter is not involved unless there is a dispute between the participants. But having a single trusted entity is one of the biggest problems that make the use of such protocols hard in practice. Therefore, the use of multiple arbiters is generally motivated by reducing the trust put on the arbiter [6,18].[1] A very natural question is how to achieve fairness in the absence of a single trusted arbiter; for example, what if we have n arbiters only a fraction of whom we want to put our trust in? It is clear that this can be achieved using byzantine agreement or secure multi-party computation techniques [17,9,10,14] with $\Omega(n^2)$ communication, but can we do better than that? In particular, can we do anything in a setting where the arbiters need not communicate with each other to resolve disputes? This issue is highly relevant especially for peer-to-peer settings in which the arbiters do not even know each other, and may not have enough resources for complicated schemes. Furthermore, if the scheme gets more costly, it will be hard to incentivize multiple arbiters to arbitrate, since they will get overloaded.

Avoine and Vaudenay (AV) [6] address this problem in their paper by using verifiable secret sharing techniques to employ multiple arbiters in their fair exchange protocol for a p2p system. In their setting, two peers are performing a fair exchange, and a number of other peers constitute the arbiters. They provide bounds on the number of arbiters that should be honest for their protocol to be fair (see Section 6). A crucial point is that the protocol uses global timeout mechanisms, which assumes all arbiters have access to -loosely- synchronized clocks, and the arbiters are autonomous (they do not communicate with each other). They leave two important issues as open questions: (1) Can an optimistic fair exchange protocol without timeouts provide fairness (since it is hard to achieve synchronization in a p2p setting) when employing multiple autonomous arbiters? (2) Can any other optimistic fair exchange protocol with timeouts achieve better bounds on the number of arbiters that need to be honest?

Unfortunately, in this paper, we answer both of these questions negatively. Inspired by state-of-the-art optimistic fair exchange protocols with a single arbiter, we define a general class of optimistic fair exchange protocols with multiple arbiters, called "distributed arbiter fair exchange" (DAFE) protocols. Informally, in a DAFE protocol, if one of the participants fails to send a correctly formed message, the other participant must contact some subset of the arbiters and get correctly formed responses from them in order to make the exchange fair.[2] Two main properties of a DAFE protocol are its abort/resolve semantics and the autonomy of multiple arbiters used, as discussed in Section 2. In a DAFE protocol, the arbiters are autonomous; they do not talk to each other, but talk only to Alice and Bob. A third property is the state machine semantics of the participants. We show that this class of protocols capture currently known state-of-the-art optimistic fair exchange protocols extended to use multiple distributed arbiters

[1] It is possible to have multiple arbiters deployed for reducing the load, but if only one of them is employed per exchange, we do not consider that protocol as having distributed arbiters.

[2] Of course, if no message is sent yet, there is no need to contact arbiters, which is not an interesting case to analyze anyway.

in a very intuitive manner, as shown in Section 2.1. Under this framework, in Section 4 we analyze scenarios that can occur during the execution of instances of optimistic fair exchange protocols, and prove some predicates every such protocol must satisfy to be able to provide semantic fairness, which is a property that needs to be satisfied by all optimistic fair exchange protocols.

In Section 5, we prove that no DAFE protocol can provide fairness meaningfully[3], answering the first open question negatively. In Appendix B, we prove impossibility of DAFE protocols using threshold-based mechanisms (any k arbiters are enough for resolution) even when the autonomous arbiters assumption is relaxed. For protocols using general set-based mechanisms (any k arbiters will not be enough for resolution, specific sets of arbiters need to be contacted), we cannot prove impossibility in this relaxed setting, but we conjecture that such protocols are not possible. However, our impossibility results can be overcome in the timeout model (where all arbiters have access to loosely synchronized clocks) and also in case the arbiters can communicate. We use our framework to analyze the existing AV protocol [6] in this timeout model in Section 6, showing how easy it is to apply our framework. We prove that the bounds on the required number of honest arbiters proven earlier for that protocol are optimal, and hence answer the second open question also negatively.

These results mean that many optimistic fair exchange protocols that want to efficiently distribute their arbiters may need to employ synchronized clocks. And even in this case, they cannot hope to require fewer honest arbiters than the Avoine and Vaudenay protocol [6]. If they do not want to employ synchronized clocks, then they may need to employ costly solutions like secure multi-party computation or Byzantine agreement.

2 Definition of a DAFE Protocol

In this section, we define a general optimistic fair exchange model that fits currently known state-of-the-art optimistic fair exchange schemes that uses an arbiter, and has semantics for aborting and resolving that we define below.

All the participants (Alice, Bob and the arbiters) are interactive Turing Machines (ITMs)[4]. Those ITMs have the following 4 semantic states: *Working* , *Aborted* , *Resolved* , *Dispute* (see Figure 1). These semantic states can correspond to multiple states in the actual ITM definitions of the participants, but these abstractions will be used to prove our results.

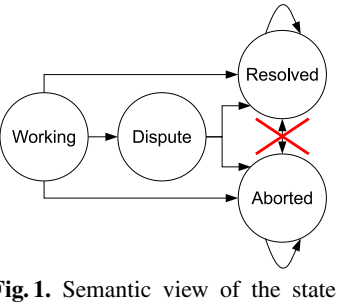


Fig. 1. Semantic view of the state machines of the participants

The ITM of each participant starts in the *Working* state. Semantically, *Working* state denotes any state that the actual ITM of a participant is in when the protocol is still taking place. When a participant does not receive the expected correctly formed message from the other participant, he can possibly abort or decide to contact the arbiters

[3] We prove that multiple arbiters are no better (or actually worse) than a single arbiter in terms of trust in the DAFE framework.

[4] The ITMs have access to –possibly synchronized– clocks for timeout mechanisms.

for resolving or aborting with them, in which case the ITM of that participant enters its *Dispute* state. If everything goes well in the protocol execution (all messages received from the other party are correctly formed), then the ITM of a participant transitions to the *Resolved* state directly from the *Working* state. Otherwise, if the arbiters needed to be contacted, the ITM first visits the *Dispute* state, and then transitions to either *Resolved* or *Aborted* state. Arbiters' *Dispute* state is dummy, and hence not needed in our analysis. Furthermore, when in Appendix B we relax one of our assumptions, even Alice and Bob will not have this *Dispute* state.

When the protocol ends, Alice and Bob are allowed to end only in *Aborted* or *Resolved* states. If Alice or Bob ends at its *Resolved* state, then, by definition, (s)he must have obtained the exchange item from the other party. When the protocol ends, if the ITM of a participant is not in its *Resolved* state, it is considered to be in its *Aborted* state.

Using these semantic definitions, even an adversarial ITM can be considered to have those 4 states (since it either obtains the other party's item and hence ends at its *Resolved* state, or not therefore ending at its *Aborted* state). The adversarial ITM does not necessarily have a *Dispute* state, but this will not affect any results presented in this paper. One can think that the moment the honest party's ITM enters its *Dispute* state, the adversarial ITM also enters its *Dispute* state.

We will talk about only complete DAFE protocols (for a definition of optimistic fair exchange protocols, see Appendix A): when both participants are honest, they end at their *Resolved* states. Since our goal here is to analyze fairness of such protocols, the only interesting case is when we have one honest party denoted H and one malicious party denoted M. We will not consider cases where both parties are malicious since there is no honest party to protect.

Definition 1 (End of the Protocol). *We say that the protocol has ended if (1) the honest party ended up being in her either* Resolved *or* Aborted *state, and (2) the adversary produced its final output at its either* Resolved *or* Aborted *state after running at most a polynomial number of steps (polynomial in some security parameter).*

Now that we defined our participants carefully, we can state our assumptions on them and define DAFE protocols.

DISTRIBUTED ARBITER FAIR EXCHANGE (DAFE) PROTOCOLS: DAFE protocols are optimistic fair exchange protocols that can be characterized with the following:

- Exclusive states assumption
- Connection between arbiters' state and Alice's and Bob's
- Autonomous arbiters assumption

EXCLUSIVE STATES ASSUMPTION: This assumption states that the *Resolved* and *Aborted* states are mutually exclusive. For an arbiter, those states informally mean whether or not the arbiter helped one of the parties to resolve or abort. We assume that there is no combination of state transitions that can take an honest *arbiter* from the *Aborted* state to the *Resolved* state, or vice versa. In most existing protocols, this corresponds to the fact that the arbiter will not abort with a participant first and then decide to resolve with him or the other participant, or vice versa. An honest arbiter can keep

executing abort (or resolve) protocols with other participants in the exchange while he is in the *Aborted* (or *Resolved* , respectively) state, but can not switch between states for different participants.

Definition 2 (Aborting and Resolving with an Arbiter). *If a participant interacts with an arbiter and aborts with him, the arbiter goes to his* Aborted *state, from where he will never switch to his* Resolved *state. Similarly, if a participant resolves with an arbiter, the arbiter goes to his* Resolved *state, from where he will never switch to his* Aborted *state.*[5]

Definition 3 (Aborted and Resolved Protocol Instance). *A protocol instance is called aborted if both Alice and Bob ended at their* Aborted *states, and called resolved if both Alice and Bob ended at their* Resolved *states.*

CONNECTION BETWEEN ARBITERS' STATE AND ALICE'S AND BOB'S: A resolution makes sense if at least one of the parties has not resolved yet. In such a case, Alice or Bob can end in their *Resolved* states (unless they already are in their *Resolved* states) only if a set of arbiters end in their *Resolved* states. This set of arbiters can be different for Alice or Bob. Actually, there can be more than one set of arbiters that is enough for this resolution. All these will be clear in later sections when we define those sets of arbiters that will be sufficient for resolution.

AUTONOMOUS ARBITERS ASSUMPTION: We assume that the honest arbiters' decisions are made autonomously, without taking into account the decisions of the other arbiters. Arbiters can arrive at the same decision seeing the same input, but they will not consider each other's decision while making their own decisions. In particular, this means no communication takes place between honest arbiters (malicious arbiters can do anything they want).

Our goal in this is to distribute the trust efficiently. Without autonomy, byzantine fault tolerance or secure multiparty computation techniques [17,9,10,14] can be applied, yielding costly solutions ($\Omega(n^2)$ communication when n arbiters are employed). Furthermore, autonomy of the arbiters render the deployment of such a real system practical, since no coordination of the arbiters is necessary.

Yet, a dependence between the arbiters' decisions can be generated by Alice or Bob, by contacting the arbiters with some specific order. Therefore, to model the autonomy, we require the protocol design to direct the honest participants to contact all the arbiters without any order. More formally, when the ITM of an honest participant decides to contact the arbiters for dispute resolution, the participant creates the message to send to all of the arbiters before receiving any response from any arbiter. One can model this with the *Dispute* state in which the message to send to the arbiters are prepared all at once. We will call this simultaneous (or unordered) resolve/abort. Note that this only constrains honest Alice or Bob. A malicious party can introduce dependence between messages to arbiters and responses from other arbiters. Later in Appendix B we will relax this autonomy assumption and discuss its consequences. We realize that this assumption is not necessary for most of our results, but helps making the presentation clearer.

[5] Due to the exclusive states assumption, these happen only if an arbiter is not already in his *Resolved* or *Aborted* state, respectively.

All optimistic fair exchange protocols need to satisfy the following semantic fairness property.

SEMANTIC FAIRNESS: The semantic fairness property states that at the end of the protocol, Alice and Bob both end at the same state (they both end at their *Aborted* states, or they both end at their *Resolved* states). In other words, we need the protocol instance to be either resolved or aborted as in Definition 3, for every possible instance of the protocol.[6]

Optimistic fair exchange protocols should also satisfy the *timely resolution* property, meaning that the honest party need not wait indefinitely for any message from any other party. He can have a local timeout mechanism with which he can decide to proceed without waiting. In particular, he can end his side of the protocol any time he wants, ending at his *Resolved* or *Aborted* state, according to the rules we defined above. Note that in general providing timely resolution guarantees necessitates mutually exclusive *Resolved* and *Aborted* states, and a way for the arbiters to transition to their *Aborted* states through interaction with other parties or through the use of timeouts.

Regular DAFE protocols do not have global timeout mechanisms, and the sets of arbiters that Alice or Bob can resolve with are well-defined by the protocol, and does not change once the honest party is in its *Dispute* state. We will show an extended version called DAFE with timeouts (DAFET) where the protocols are allowed to use timeouts. At the timeout specified by the protocol, honest arbiters transition into their *Aborted* states. This is done using the (loosely synchronized) clocks of the ITMs. We call this event "an arbiter timeouts". We allow the possible sets of arbiters to resolve with to change at this timeout. This timeout model bypasses the impossibility results for DAFE protocols. These will be clear later.

We will first provide examples of existing optimistic fair exchange protocols with intuitive extensions to employ multiple autonomous arbiters and show how they fit our DAFE classification. Then, after defining some notation, we will analyze different possible protocol instances under different scenarios, and possible protocol types. We then show that it is impossible for some common types of DAFE protocols to provide semantic fairness, thus warning researches not to pursue that direction. We also analyze some positive results using global timeout mechanisms, and prove the optimality of the bounds of the AV protocol, showing the usability of our framework for easy analysis. We then discuss the role of autonomous arbiters and timeouts in our results and elaborate on different ideas.

2.1 Sample DAFE Protocols

Many currently known optimistic fair exchange protocols can be considered as special cases of DAFE protocols in which there is only one arbiter. In this section, we also discuss a way to extend them to employ multiple autonomous arbiters. Unfortunately, this means, those extended protocols cannot provide fairness, as we will prove later

[6] There will not be any cases where the honest party ends at its *Resolved* state whereas the malicious party ends at its *Aborted* state and this affects our results. Therefore, this semantic fairness definition is enough for our purposes. Furthermore, it is subjective whether or not to consider a case where two parties end at different states as fair.

in this paper that no DAFE protocol can provide fairness. Precisely, our impossibility result states that all arbiters need to be trusted in a DAFE protocol, hence they are not realistic. For the special single-arbiter case, this points out to the trust assumption on the arbiter.

To the best of our knowledge, all currently known optimistic fair exchange protocols adhere with our framework. As a representative of optimistic fair exchange protocols, we will analyze a protocol due to Asokan, Shoup and Waidner (ASW) [4]. They have two versions of their protocol: one version that uses timeout-based aborts (can be converted to a DAFET protocol, see Section 6), and one that does not employ timeouts (we will discuss now). It is considered one of the state-of-the-art signature exchange protocols, and is the first completely fair optimistic exchange protocol. A state-of-the-art optimistic fair exchange protocol for exchanging files are given in [18], and all our discussion here applies to that protocol too. The ASW protocol without timeouts is described below for reference:

1. Alice sends Bob a non-verifiable escrow of her signature, with a label defining how Bob's signature should look like. Bob checks if the definition is the correct definition.
2. Bob sends Alice a *verifiable* escrow of his signature, with the label defining how Alice's signature should look like and also attaching the escrow he obtained in step 1. Alice verifies the verifiable escrow. She furthermore checks if the label is formed correctly. If anything goes wrong at this step or a *message timeout* occurs, she aborts the protocols and runs AliceAbort with the Arbiter.
3. Alice sends Bob her signature. Bob verifies this signature, and stops and runs BobResolve if it does not verify or a *message timeout* occurs.
4. Bob sends Alice his signature. If the signature does not verify, Alice runs AliceResolve.

AliceAbort tells the Arbiter to consider that trade as aborted and not to honor any further resolution request on that particular trade. BobResolve gets Alice's signature by providing Bob's signature, and similarly, AliceResolve gets Bob's signature by providing Alice's signature.

In terms of the state semantics of the participants, it is clear that the ending states of the participants can be parsed into *Aborted* and *Resolved* states which are mutually exclusive. Furthermore, honest participants are not allowed to transition between *Aborted* and *Resolved* states. In particular, once Alice aborts with the arbiter taking him to his *Aborted* state, he will refuse resolving with Bob. Since there is only one arbiter, it is autonomous. As for the connection between arbiter's state and Alice's and Bob's, it is clear that in case of a dispute, their state depends on the arbiter's.

Now, if we want to extend those protocols to use multiple autonomous arbiters, one easy way is to employ verifiable secret sharing techniques [6,22,16]. The state-of-the-art optimistic fair exchange protocols employ verifiable escrows [11,13,4,18] under the (one and only) arbiter's public key. The intuition behind using verifiable escrows is that the recipient can verify, without learning the actual content, that the encrypted content is the content that is promised and the arbiter can decrypt it. Verifiable secret sharing techniques can be employed to split the promised secret per arbiter. Each of these secrets

will be encrypted under a different arbiter's public key. The recipient can still verify those encrypted shares can be decrypted and combined to obtain the promised secret, thereby effectively achieving the same goal as a verifiable escrow, but for multiple arbiters. For a detailed explanation of how to use verifiable secret sharing in distributing the arbiters, we refer the reader to [6].

When we extend the ASW protocol to use multiple autonomous arbiters, instead of this verifiable escrow, the participants will use verifiable secret sharing techniques as explained above and in [6]. Regardless of whether threshold- or set-based secret sharing mechanisms are used, the resolution procedure now requires contacting multiple arbiters. For example, if the threshold for the secret sharing method used is k, the resolution will involve contacting at least k arbiters.

In terms of the state semantics of the participants, it is clear that the ending states of the participants can be parsed into *Aborted* and *Resolved* states which are mutually exclusive. Because we assume the arbiters are contacted simultaneously, the autonomy of the arbiters hold. As for the connection between arbiters' state and Alice's and Bob's, since resolution needs k shares, and secure secret sharing and encryption methods are used, a participant can obtain the other participant's exchange item if and only if (s)he resolves with at least k arbiters (in case of a dispute). This relationship makes perfect sense when multiple autonomous arbiters are used, since the main goal in distributing the arbiter is distributing the trust. Therefore, the goal is to find some number of honest arbiters each one of which will individually contribute to dispute resolution between participants by resolving or aborting with them. When arbitrary sets are used instead of thresholds, it is easy to see all these arguments will still apply.

The same techniques can be applied to another state-of-the-art optimistic fair exchange protocol [18] designed to exchange multiple files between participants. This protocol employs a verifiable escrow for escrowing the payment (endorsement of an unendorsed e-coin [12]) sent by the participants. All the arguments for the ASW protocol also apply here. Again, verifiable secret sharing techniques as discussed above will be used instead of the verifiable escrow. The resolution mechanism will be similar to the ones we described for the extended ASW protocol. As for the state semantics, a participant goes to her/his *Resolved* state if (s)he gets other participant's file or e-coin, and goes to his/her *Aborted* state otherwise.

In Section 4 we will analyze possible scenarios in an optimistic fair exchange protocol. The first two scenarios will be applicable to this extended protocol types, as we show in Section 5, where we analyze protocols that have the same structure as ASW protocol.

3 Notation

Remember that in a fair exchange scenario, Alice and Bob want to exchange some items fairly. In case of a dispute, they need to contact the arbiters. They are allowed to take the following two actions with the arbiters: *abort* or *resolve*. As noted in Definition 2, aborting with an honest arbiter takes him to his *Aborted* state, whereas resolving with him would take him to his *Resolved* state.[7] Remember, those states are mutually

[7] This happens only if an arbiter is not already in its *Resolved* or *Aborted* state, respectively.

exclusive, and there is no transition between them, direct or indirect. We assume that the arbiters are autonomous: They do not take into account other arbiters' decision while acting. More formally, the honest participant contacts all arbiters simultaneously (her messages to arbiters do not depend on any response from any of the arbiters).

Let N denote the set of all arbiters, where there are a total of n of them ($|N| = n$). An honest arbiter acts as specified by the protocol. Let F be the set of arbiters who are friends with a malicious participant. Those arbiters are adversarial.[8]

Define two sets $\mathcal{H}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{R}}$, which are sets of sets. Any set $H_R \in \mathcal{H}_{\mathcal{R}}$ is a set of arbiters that is sufficient for the *honest* party to *resolve* (as defined in Section 2 during the discussion about the connection between arbiters' state and Alice's and Bob's). Similarly, any set $M_R \in \mathcal{M}_{\mathcal{R}}$ is a set of arbiters that is sufficient for the *malicious* party to *resolve*. Therefore, by definition, in case of a dispute, the honest party will end at her *Resolved* state **if and only if** she resolves with **all** the arbiters in **any one** of the sets in $\mathcal{H}_{\mathcal{R}}$ (unless she already is in her *Resolved* state). Similarly, the malicious party will end at his *Resolved* state **if and only if** he resolves with **all** the arbiters in **any one** of the sets in $\mathcal{M}_{\mathcal{R}}$ (unless he already is in his *Resolved* state). *For DAFE protocols, these sets are well-defined by the protocol description, and do not change once the honest party enters its* Dispute *state.*

A special case of these sets can be represented as thresholds. Let T_H be the number of arbiters the honest party needs to contact for resolution. Similarly, T_M denotes the number of arbiters the malicious party needs to contact for resolution. Thus, the set $\mathcal{H}_{\mathcal{R}}$ is composed of all subsets of N with T_H or more arbiters. Similarly, the set $\mathcal{M}_{\mathcal{R}}$ is composed of all subsets of N with T_M or more arbiters.

Define R_H as the set of arbiters the honest party H has already resolved with, and R_M as the set of arbiters the malicious party M has already resolved with. Also define R_A as the set of all arbiters that are available for H for resolution. Initially, when the dispute resolution begins, we assume that $R_H = \varnothing$, $R_M = F$, and $R_A = N - F$ (and all arbiters are available for resolution to the malicious party). We furthermore have the following actions and their effects on these sets:

Action 1 (H resolves with an arbiter X) *The effect is that R_H becomes $R_H \cup \{X\}$.*

Action 2 (M resolves with an arbiter X) *The effect is that R_M becomes $R_M \cup \{X\}$.*

Action 3 (H aborts with an arbiter $X \in R_A$) *The effect is that R_A becomes $R_A - \{X\}$.*

Action 4 (M aborts with an arbiter $X \in R_A$) *The effect is that R_A becomes $R_A - \{X\}$.*

Note that we do not care what these sets actually are, or whether or not one can find such sets of sets. For our impossibility result, it is enough that conceptually these sets of sets exist.

As in previous work on optimistic fair exchange [4,18], we assume that the adversary can re-order messages, delay the honest party's messages to the arbiters, insert his own messages, *etc.* But he cannot delay honest party's messages indefinitely: the honest party eventually reaches the arbiters that he wants to contact initially, and this occurs before the timeout if the protocol uses timeout mechanisms.

[8] For example, they may appear as aborted to the honest party, but they may still resolve with the malicious party.

3.1 DAFET Protocols (DAFE Protocols with Timeouts)

In DAFET protocols, we allow for timeouts by giving the arbiters access to loosely synchronized clocks. Instead of actions 3 and 4 above (honest or malicious party aborting), the following action is allowed:

Action 5 (An arbiter $X \in R_A - R_H - R_M$ timeouts) *The effect is that* R_A *becomes* $R_A - \{X\}$.

Another difference between DAFE and DAFET protocols is the sets $\mathcal{H}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{R}}$ being static and dynamic, respectively. DAFE protocols define such sets as static: the overall set of arbiters that needs to be contacted for resolution does not change with time once the honest party enters its *Dispute* state (hence the notation $\mathcal{H}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{R}}$). In contrast, we allow DAFET protocols to employ dynamic sets (hence the notation $\mathcal{H}_{\mathcal{R}}(t)$ and $\mathcal{M}_{\mathcal{R}}(t)$). These sets may depend on the timeout and possibly the parties' actions in that particular instance of the protocol. Consider the following two cases as illustrative examples: Some type of protocols allow, let's say, Alice to resolve only after a timeout. Some other type of protocols allow Alice to resolve only with an arbiter that Bob has already resolved with (or vice versa). In analyzing such types of protocols, we will consider $\mathcal{H}_{\mathcal{R}}(t)$ and $\mathcal{M}_{\mathcal{R}}(t)$ as dynamic, letting them change with those actions. We discuss the relation between the use of timeouts and dynamic sets in fair exchange protocols more in Appendix C.

We will consider any action that results in a change in those sets as new time steps, but there is no need to treat other events as separate time steps since they do not constitute a significant part of the analysis. Therefore, one can think as if any party can contact any number of arbiters at a given time step t. $t = 0$ denotes the time when the dispute resolution begins (the time the honest party enters its *Dispute* state, not the time the protocol execution begins).

Lastly, the set of friends of a malicious party can also change with time, if the adversary is allowed to adaptively corrupt arbiters. In that case, we will use the notation $F(t)$.

4 Framework for Analysis of DAFE Protocols

In this section, we will provide our framework for analyzing DAFE (and DAFET) protocols. Our framework is composed of different scenarios that can take place during the execution of an instance of a DAFE protocol. Once we have lemmas related to those scenarios stating the necessary (not necessarily sufficient) conditions that need to be satisfied so that the given scenario satisfies the semantic fairness property, then we can analyze different protocol types in the next section.

Since our results are impossibility or lower bound type of results, it is enough to analyze necessary (but maybe not sufficient) conditions. In all our scenarios (except the last one), we assume that neither party is in the *Resolved* state yet. We consider dynamic resolution sets for our scenario analysis, since static sets are a special case of dynamic sets.

4.1 Scenario 1: M Can Abort

In this scenario, we consider a protocol instance where the malicious party has the ability to abort and resolve. The honest party can abort and resolve too, but the results still apply even if he is restricted to only resolve action. In this scenario, actions 1, 2, and 4 in Section 3 are possible. Our results in this section will remain valid regardless of action 3 being possible.

Lemma 1. *Every DAFE protocol instance needs to make sure that there exists a time t when $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$* s.t. $H_R \subseteq M_R - F(t)$.

Proof. Assume otherwise: At any time in the protocol instance $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$ *s.t.* $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)$ $H_R \not\subseteq M_R - F(t)$. The malicious party can break fairness as follows: He aborts with the set of arbiters $R_A - M_R$, and resolves with the set of arbiters M_R. Since no H_R is now a subset of the available arbiters $R_A = M_R - F(t)$, the honest party cannot resolve, while the malicious party already resolved. Thus this protocol instance is unfair (does not satisfy semantic fairness).

Corollary 1. *At any given time t during the protocol instance before the protocol is resolved for H, we need $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $M_R \not\subseteq F(t)$ since otherwise we need $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$* s.t. $H_R = \varnothing$.

Corollary 2. *We need a time t to exist satisfying $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$* s.t. $H_R \cap F(t) = \varnothing$ *since otherwise the lemma cannot be satisfied (H can never resolve).*

Corollary 3. *Using threshold-based mechanisms, we need that there exists a time t that satisfies $T_H \leq T_M - |F(t)|$.*

Corollary 4. *Using threshold-based mechanisms, at any given time t during the protocol instance before the protocol is resolved for H, we need $T_M > |F(t)|$ since otherwise we need $T_H \leq 0$.*

Corollary 5. *Using threshold-based mechanisms, we need a time t to exist satisfying $T_H \leq n - |F(t)|$ since otherwise H can never resolve.*

4.2 Scenario 2: Only H Can Abort

In this scenario, we assume that the malicious party has the ability to resolve only, whereas the honest party can abort and resolve. In this scenario, actions 1 to 3 in Section 3 are possible (action 4 is not possible).

Lemma 2. *Every DAFE protocol instance needs to make sure that there exists a time t when $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$* s.t. $H_R \subseteq M_R - F(t)$.

Proof. Assume otherwise: At any given time $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$ *s.t.* $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)$ $H_R \not\subseteq M_R - F(t)$. The malicious party can break fairness as follows: When H wants to abort the protocol, M lets abort messages to all arbiters in $R_A - M_R$ to reach their destination, but intercept the messages to $M_R - F(t)$ ($F(t)$ really does not matter since his friends will help him anyways). He then resolves with M_R. Even if H notices this, he cannot go and resolve since there is no set $H_R \in \mathcal{H}_{\mathcal{R}}(t)$ that will allow him to. Therefore, this protocol instance also does not satisfy semantic fairness.

Note that Lemma 2 is the same as Lemma 1, and therefore all the corollaries apply to this scenario too.

4.3 Scenario 3: H Can Resolve Only After Timeout

In this scenario, aborts can be caused by timeouts only. The malicious party can resolve before and after the timeout, but the honest party can resolve only after the timeout. Therefore, actions 2 and 5 are possible, but not 3 and 4. Action 1 is possible only after the timeout.

Lemma 3. *Every DAFET protocol instance needs to make sure there exists a time t when* $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)\ \exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$ s.t. $H_R \subseteq M_R - F(t)$.

Proof. Assume otherwise: At any given time $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$ *s.t.* $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)\ H_R \nsubseteq M_R - F(t)$. The malicious party can break fairness as follows: M resolves with M_R before the timeout. When the timeout occurs, all arbiters in $R_A - R_H - R_M$ to go to their *Aborted* states (R_H being the empty set), which means now $R_A = M_R - F(t)$. But H cannot resolve with the remaining available arbiters and hence this protocol instance is not semantically fair.

Note that Lemma 3 is the same as Lemma 1, and therefore all the corollaries apply to this scenario too.

4.4 Scenario 4: M Already Resolved

All of the scenarios above assumed that both H and M start in their *Working* states when they are performing the resolutions. Yet, it might be perfectly possible that the resolution starts at a point in the protocol where one of the parties has already resolved (and hence is in its *Resolved* state). If H has already resolved, then there is no point to further analyze, since we do not care if the protocol is fair to the malicious party. But if M has already resolved, then we need the following lemma to hold:

Lemma 4. *Every DAFE protocol instance needs to make sure that there exists a time t when* $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$ s.t. $H_R \cap F(t) = \varnothing$.

Proof. Assume at all times $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)\ H_R \cap F(t) \neq \varnothing$. The malicious party has already resolved but since all possible ways to resolve for H has to go through one of the malicious party's friends, he has no hope of resolving.

This lemma corresponds to corollary 2 and hence corollary 5 also applies here.

5 Impossibility Results on DAFE Protocols

The previous section analyzed possible scenarios in DAFE and DAFET protocol instances. In this section, we will analyze DAFE protocol types, using the results from different scenarios that might come up in instances of such protocols. We will conclude that no DAFE protocol can provide fairness under any realistic assumption. DAFET protocols using dynamic sets are possible indeed, and we analyze an existing DAFET protocol in Section 6.

For every protocol type, we will consider the following two cases: The case where the honest player plays the role of Alice, and the case where he plays the role of Bob. We denote the set of sets for Alice to resolve as $\mathcal{A}_{\mathcal{R}}(t)$; similarly $\mathcal{B}_{\mathcal{R}}(t)$ is for Bob to resolve. The difference in types of protocols related to these sets being static or dynamic will play a big role. For ease of analysis (and since it is enough for the impossibility results in this section) we will assume the friend list $F(t)$ of the malicious party is static (does not change with time).[9] Since this is a weaker adversary, our impossibility results will also apply when we consider stronger (adaptive) adversaries. We will use F_A to denote friends of a malicious Alice, and F_B to denote friends of a malicious Bob.

In the DAFE protocol types below, we will consider the sets $\mathcal{A}_{\mathcal{R}}(t)$ and $\mathcal{B}_{\mathcal{R}}(t)$ as static (therefore using the notation $\mathcal{A}_{\mathcal{R}}, \mathcal{B}_{\mathcal{R}}$), which eases the use of the lemmas. With static sets, we do not need to consider different times in the protocol instance. A lemma saying there must exist a time t can be simplified by just looking at the initial sets.

5.1 Protocol 1: Alice and Bob Can Abort and Resolve

In this type of protocols, Alice is given the ability to abort and resolve, and Bob is also given the ability to abort and resolve.

Case 1: Honest Alice *vs.* Malicious Bob: This case falls under Scenario 1, which means (for the static case) any DAFE protocol needs to have $\forall B_R \in \mathcal{B}_{\mathcal{R}}\ \exists A_R \in \mathcal{A}_{\mathcal{R}}$ *s.t.* $A_R \subseteq B_R - F_B$.
Case 2: Malicious Alice *vs.* Honest Bob: This case also falls under Scenario 1, which means (again for the static case) any DAFE protocol needs to have $\forall A_R \in \mathcal{A}_{\mathcal{R}}\ \exists B_R \in \mathcal{B}_{\mathcal{R}}$ *s.t.* $B_R \subseteq A_R - F_A$.

These two cases lead to the conclusion that every protocol instance needs two sets $A_R \in \mathcal{A}_{\mathcal{R}}$ and $B_R \in \mathcal{B}_{\mathcal{R}}$ *s.t.* $A_R = B_R \subseteq \{\text{trusted arbiters}\}$. These arbiters must be trusted, and so there is no point in distributing the arbiters. It is even worse: If any of these arbiters are corrupted, the DAFE protocol fails to be fair. Therefore, no such realistic DAFE protocol can exist.

When considering threshold-based schemes, this corresponds to the requirement that $T_B \leq T_B - F_A - F_B$, which means no party should have any friends for such a protocol to be fair. If even one arbiter is corrupted, the protocol becomes unfair. Therefore, no such realistic DAFE protocol can exist. Since set-based mechanisms cover threshold-based ones, we will not discuss threshold-based schemes separately again unless necessary. All impossibility results proven for set-based mechanisms directly apply in the context of threshold-based ones.

5.2 Protocol 2: Only One Party Can Abort

In this type of protocols, Alice is given the ability to abort and resolve, whereas Bob is given only the ability to resolve. Analysis of protocols that are symmetric to this type of protocols (where Bob can abort and resolve, and Alice can only resolve) obviously yields to the same conclusions.

[9] This corresponds to the familiar "static corruption model" in many other works.

Case 1: Honest Alice *vs.* Malicious Bob: This case falls under Scenario 2, which requires that DAFE protocols need to make sure $\forall B_R \in \mathcal{B}_{\mathcal{R}}\ \exists A_R \in \mathcal{A}_{\mathcal{R}}\ s.t.\ A_R \subseteq B_R - F_B$.
Case 2: Malicious Alice *vs.* Honest Bob: This case falls under Scenario 1, which means any DAFE protocol needs to have $\forall A_R \in \mathcal{A}_{\mathcal{R}}\ \exists B_R \in \mathcal{B}_{\mathcal{R}}\ s.t.\ B_R \subseteq A_R - F_A$.

We can conclude as in the previous section (Section 5.1) that every protocol instance needs two sets $A_R \in \mathcal{A}_{\mathcal{R}}$ and $B_R \in \mathcal{B}_{\mathcal{R}}\ s.t.\ A_R = B_R \subseteq \{\text{trusted arbiters}\}$. Again, this means there is no point in distributing the arbiters in terms of trust. Remember that threshold-based versions have the same impossibility.

Unfortunately, the versions of the state-of-the-art optimistic fair exchange protocols we analyzed in Section 2.1 *without* any timeouts fall under this protocol category. Note that, this means, using static resolution sets and autonomous arbiters, those protocols cannot be extended to use multiple arbiters and remain fair.

6 Applying DAFET Framework to Prove Optimality of an Existing Protocol

In this section, we analyze an existing DAFET protocol that uses dynamic resolution sets: The set of arbiters needed by a party for resolution changes during the course of the execution of the protocol instance. By adjusting resolution sets reactively, this protocol can provide semantic fairness.

AV Protocol [6]. This protocol is due to Avoine and Vaudenay (AV) [6]. In this protocol, timeouts are used for aborting (it is a DAFET protocol). It is a three-step protocol in which Alice starts by sending verifiable secret shares encrypted under each arbiter's public key. Then, Bob responds with his secret, and Alice responds with her secret. To resolve, Bob contacts k arbiters to get the decrypted shares and reconstruct the secret of Alice (where k is the threshold for the secret sharing scheme). Before giving the decrypted share, each honest arbiter asks for the secret of Bob.[10] Hence, the set $\mathcal{B}_{\mathcal{R}}(t)$ contains all subsets of N with k or more arbiters and $\mathcal{A}_{\mathcal{R}}(t)$ is initially empty[11].

The state semantics obviously coincide with our 3-state definition. The participants either succeed in obtaining the other party's exchange item and hence end at their *Resolved* state, or they fail to do so and end at their *Aborted* state. The honest arbiters will either help both participants, or abort at the timeout and help neither.

Even though in the AV protocol the honest arbiters directly contact Alice when Bob resolves with them, we can see it as the arbiters storing Bob's secret, and Alice contacting them to obtain Bob's secret later on. Since Alice can only resolve after Bob, and Bob has to resolve before the timeout, it is safe to think of this protocol as letting Alice to resolve only after the timeout. Unlike the protocols in Section 5 which were proven impossible to be fair, this protocol uses dynamic resolution sets that help it achieve fairness (we talk about the relationship between timeouts and dynamic resolution sets in Appendix C). So, sets $\mathcal{H}_{\mathcal{R}}(t)$ and $\mathcal{M}_{\mathcal{R}}(t)$ change according to the following additional rule regarding the actions (remember the actions in Section 3):

[10] The user should refer to [6] for any more details.

[11] It does not contain the empty set, it is empty. This means no set of arbiters is sufficient for Alice to resolve.

Action 6 (Bob resolves with an arbiter $X \in R_A$) *The effect is that a set $\{X\}$ is added to the set of sets $\mathcal{A}_{\mathcal{R}}(t)$.*

This rule is there since in the AV protocol, when Bob contacts an honest arbiter, that arbiter contacts Alice and sends Bob's whole secret. It guarantees that the moment a malicious Bob resolves with any honest arbiter, Alice is guaranteed to be able to resolve. Let us analyze the two cases and see how this protocol satisfies the lemmas regarding scenarios.

Case 1: Honest Alice *vs.* Malicious Bob: This case falls under Scenario 3, which means any DAFET protocol needs to make sure there exists a time when $\forall B_R \in \mathcal{B}_{\mathcal{R}}(t)$ $\exists A_R \in \mathcal{A}_{\mathcal{R}}(t)$ *s.t.* $A_R \subseteq B_R - F_B$.
Case 2: Malicious Alice *vs.* Honest Bob: Depending on at which point of the protocol the resolution begins, malicious Alice might have already resolved, thus this case falls under Scenario 4, which requires that there exists a time when $\exists B_R \in \mathcal{B}_{\mathcal{R}}(t)$ *s.t.* $B_R \cap F_A = \varnothing$.

Lemma 5. *AV protocol cannot provide semantic fairness unless for all times t $\forall B_R \in \mathcal{B}_{\mathcal{R}}(t)$ $B_R \not\subseteq F_B$ AND for some time t $\exists B_R \in \mathcal{B}_{\mathcal{R}}(t)$* s.t. $B_R \cap F_A = \varnothing$.

Proof. Follows from the analysis of the cases above using corollary 1 for case 1.

The AV protocol achieves semantic fairness using dynamic sets as follows: The set $\mathcal{A}_{\mathcal{R}}(t)$ is initially empty. When Bob contacts an arbiter X, action 6 above takes place, and hence the set $\{X\}$ is added to the set of sets $\mathcal{A}_{\mathcal{R}}(t)$ (the threshold for Alice effectively becomes 1). Therefore, once Bob contacts an honest arbiter (not one of his friends), then Alice is guaranteed to be able to resolve. This saves an honest Alice against a malicious Bob (case 1). In case 2, as long as Bob can find a set of honest arbiters that he can resolve with, he is saved against malicious Alice.

Actually, the AV protocol [6] uses threshold-based mechanisms instead of set-based ones, therefore we have the following corollary:

Corollary 6. *AV protocol cannot provide semantic fairness unless $|F_B| < T_B$ AND $T_B \leq n - |F_A|$.*

It is important to notice that the AV paper [6] proves essentially the same result: They prove that the same bound is also sufficient for their protocol. Thus, we have proven that the bounds proven in that paper are tight and hence the protocol is optimal in that sense. Furthermore, this result is applicable to all protocols of the same type; no DAFET protocol of the same type can achieve better bounds. In particular, the same technique of employing multiple autonomous arbiters can be used on [4] and [18] (as described in Section 2.1) to convert their timeout-based versions to DAFET protocols, and the same lemma will hold. This shows how our framework can easily be applied to prove optimality of a protocol and extended to other protocols of the same type.

As the corollary immediately reveals, when using n arbiters, to obtain maximum tolerance, one should set the threshold for Bob $T_B = n/2$ so that the protocol tolerates up to $n/2 - 1$ friends of each participant. Of course, this greatly reduces the efficiency of the resolution of the optimistic fair exchange protocol.

7 Conclusion

In this paper, we presented a framework to analyze DAFE protocols, which are natural extensions of optimistic fair exchange protocols to make them use multiple autonomous arbiters (those who do not communicate with each other). Autonomy is useful for realistic (efficient) protocols, especially in p2p settings. Using the presented framework, we answered two open questions since [6]. We have proved that DAFE protocols (optimistic fair exchange protocols that employ multiple autonomous arbiters and does not have timeout mechanisms) cannot provide fairness in a realistic setting. Even when we extended our framework by relaxing the autonomy assumption about the arbiters, we found out that even broader classes of optimistic fair exchange protocols fall under our impossibility results. We then switched to the DAFET model to include timeouts and dynamically changing sets of arbiters to resolve with. We analyzed one existing DAFET protocol [6] using our framework and proved that the previous bounds on the required number of honest arbiters are optimal. No DAFET protocol of the same type can achieve better bounds, since our framework can easily be used to come up with generalized results.

Unfortunately, this means many optimistic fair exchange protocols that want to efficiently distribute their arbiters may need to employ synchronized clocks. And even in this case, they cannot hope to require fewer honest arbiters than the Avoine and Vaudenay protocol [6]. If they do not want to employ synchronized clocks, then they may need to employ costly solutions like secure multi-party computation or Byzantine agreement.

One may want to settle down for weaker security guarantees against weaker adversaries to achieve cheaper solutions than Byzantine agreement. Using Byzantine fault tolerance techniques in [1], the arbiters can keep updating some value that is related to the resolution semantics of the fair exchange. Unfortunately, when aborts are considered, it is not clear if the same techniques can be applied here. We leave research in this direction as an open problem.

Finally, our techniques may be applicable to other functionalities that can be implemented using secure multi-party computation. By designing an appropriate framework, we may prove more general results about achieving the same functionality using autonomous multiple parties. We leave such a generalization as an interesting open problem.

References

1. Abd-El-Malek, M., Ganger, G., Goodson, G., Reiter, M., Wylie, J.: Fault-scalable byzantine fault-tolerant services. In: SOSP (2005)
2. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: ACM CCS (1997)
3. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
4. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. IEEE Selected Areas in Communications 18, 591–610 (2000)
5. Ateniese, G.: Efficient verifiable encryption (and fair exchange) of digital signatures. In: ACM CCS (1999)
6. Avoine, G., Vaudenay, S.: Optimistic fair exchange based on publicly verifiable secret sharing. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 74–85. Springer, Heidelberg (2004)

7. Bao, F., Deng, R., Mao, W.: Efficient and practical fair exchange protocols with off-line TTP. In: IEEE Security and Privacy (1998)
8. Belenkiy, M., Chase, M., Erway, C., Jannotti, J., Küpçü, A., Lysyanskaya, A., Rachlin, E.: Making p2p accountable without losing privacy. In: ACM WPES (2007), http://www.cs.brown.edu/research/brownie/p2p-ecash-wpes07.pdf
9. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC, pp. 52–61 (1993)
10. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, pp. 1–10 (1988)
11. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 331. Springer, Heidelberg (2000)
12. Camenisch, J., Lysyanskaya, A., Meyerovich, M.: Endorsed e-cash. In: IEEE Security and Privacy (2007)
13. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
14. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: STOC, pp. 42–51 (1993)
15. Dodis, Y., Lee, P., Yum, D.: Optimistic fair exchange in a multi-user setting. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, p. 118. Springer, Heidelberg (2007)
16. Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 32–46. Springer, Heidelberg (1998)
17. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC, pp. 218–229 (1987)
18. Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 252–267. Springer, Heidelberg (2010), http://eprint.iacr.org/2008/431
19. Micali, S.: Simultaneous electronic transactions with visible trusted parties. US Patent 5,553,145 (1996)
20. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: PODC (2003)
21. Pagnia, H., Gärtner, F.: On the impossibility of fair exchange without a trusted third party. Darmstadt University of Technology, TUD-BS-1999-02 (1999)
22. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)

A Definition of Optimistic Fair Exchange Protocols

We provide an informal definition of optimistic fair exchange protocols taken from [18] just for completeness.

A fair exchange protocol is composed of three interactive algorithms: Alice running algorithm A, Bob running algorithm B, and the Arbiter running the trusted algorithm T. Alice has content f_A, and Bob has content f_B.

Completeness for an optimistic fair exchange states that the interactive run of A and B by *honest parties* results in A getting f_B and B getting f_A (the Arbiter's algorithm T is not involved, assuming an ideal network).

Fairness states that at the end of the protocol, either Alice and Bob both get the content of each other, or neither Alice nor Bob gets anything useful. For formal definitions, we refer the reader to [18].

B Relaxing Autonomous Arbiters Assumption

In this section, we extend our framework by relaxing the autonomous arbiters assumption to allow for ordered aborts by the honest party and therefore include a broader range of protocols in our framework. We still assume that the honest arbiters do not try to communicate, but now the honest parties can contact the arbiters following some particular order. We immediately notice that the only places where we need that assumption are Scenario 2 and Protocol 2. Results about all other scenarios and protocols stay unchanged when we do the relaxation by removing the explicit *Dispute* state in the ITM definitions of the honest participants (Alice and Bob), thus allowing them to contact the arbiters with some specific order. Yet, we still are not considering byzantine fault tolerance or secure multiparty computation techniques.

B.1 Scenario 2 Revisited

In Section 4.2, we analyzed the scenario in which the malicious party has the ability to resolve only, whereas the honest party can abort and resolve. We analyzed that scenario using the autonomous arbiters assumption. Below, we will remove the requirement that arbiters are contacted simultaneously, and revisit our analysis.

Scenario 2 with Threshold-based Mechanisms. Here, we are limiting our protocol instances to the case where only threshold-based mechanisms are used.[12] This means, the sets $\mathcal{H}_{\mathcal{R}}(t)$ and $\mathcal{M}_{\mathcal{R}}(t)$ are of the specific form we have described before. Remember, the set $\mathcal{H}_{\mathcal{R}}(t)$ is composed of all subsets of N with T_H or more arbiters. Similarly, the set $\mathcal{M}_{\mathcal{R}}(t)$ is composed of all subsets of N with T_M or more arbiters. T_H and T_M are the corresponding thresholds.

Lemma 6. *Every DAFE protocol instance needs to make sure there exists a time t when $T_H \leq T_M - |F(t)|$.*

Proof. Assume otherwise: At all times $T_H > T_M - |F(t)|$. Malicious party can break fairness as follows: When H wants to abort the protocol (as directed by the protocol, most probably triggered by an incorrect input from the malicious party), M waits until H aborts with $n - T_H + 1$ arbiters. H can no longer resolve after this point since there are less than T_H arbiters left in the set of available arbiters R_A. At this point, M intercepts any more abort messages from H and resolves with $T_M - |F(t)|$ honest arbiters (as well as $|F(t)|$ friends). Therefore, this protocol instance is unfair (does not satisfy semantic fairness).

Notice that Lemma 6 is the same as Corollary 3. Therefore, Corollaries 4 and 5 also apply here.

[12] Appendix B.1 removes the threshold limitation and allows for any set-based resolution mechanism.

Scenario 2 General Case. Now, we remove all the restrictions we made on our scenario in the previous sub-scenarios. This means, we allow for any set-based resolution mechanism, and we even allow the protocol to specify an order of arbiters for aborting, possibly depending on the execution of the protocol instance. One can think of it as the honest party aborting with one arbiter at every time step, and reconsidering his decision to abort each time. Therefore, the arbiters are no longer completely autonomous.

Lemma 7. *Every DAFE protocol instance needs to make sure that at all times* t $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $M_R \not\subseteq F(t)$ *(before* H *has resolved) AND there exists a time* t *when* $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$ s.t. $H_R \cap F(t) = \varnothing$.

Proof. Assume there exists a time when $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $M_R \not\subseteq F(t)$ (before H has resolved). Malicious party can break fairness as follows: When H wants to abort the protocol, M lets him abort with all the arbiters. Then, he goes and resolves with M_R, all members of which are his friends.

Now assume at all times $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)$ $H_R \cap F(t) \neq \varnothing$. Malicious party can break fairness by just resolving with any $M_R \in \mathcal{M}_{\mathcal{R}}(t)$. Since all possible ways to resolve for H has to go through one of the malicious party's friends, he has no hope of resolving.

In this general scenario, as in the previous cases, we would like to be able to prove that any DAFE protocol instance needs to make sure there exists a time t when $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t)$ $\exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$ *s.t.* $H_R \subseteq M_R - F(t)$. Even though this seems a very plausible and realistic conclusion, several problems arise with its proof.

The general idea is to use an adversary very similar to the one in Section 4.2. So, the adversary will let H to abort with any arbiter in $R_A - M_R$. Then, if H wants to abort with an arbiter in $M_R - F(t)$, M will intercept and resolve with M_R. The problem is that this works depending on the order of aborts. There might be a possible protocol construction and order specification that makes sure H can still resolve once he detects this behavior. We do not know of and could not come up with such a construction, due mostly to the fact that $F(t)$ is unknown to the honest party, and hence designing a protocol instance using an order that works without knowing $F(t)$ seems impossible. Even though the order may work for some protocol instances, having an order that works with high probability (that works on all but negligible fraction of protocol instances) does not seem possible. Furthermore, the moment we allow for more powerful adversaries, since the order of arbiters for the honest participant to abort is public, the adversary might "bribe" some "key" arbiters to become his friends and make sure the ordering fails to provide fairness (in the dynamic/adaptive corruption model). We admit that we have no proof for this general case with less powerful adversaries, but we conjecture that the same predicate for scenario 4.2 as before will hold.

B.2 Protocol 2 Revisited (More Impossibility Results)

In this type of protocols, Alice is given the ability to abort and resolve, whereas Bob is given only the ability to resolve. Analysis of protocols that are symmetric to this type of protocols (where Bob can abort and resolve, and Alice can only resolve) obviously yields to the same conclusions. The predicate for case 1 changes when we relax our

autonomous arbiters assumption. Case 2 stays the same. Remember, the resolution sets we consider here are static.

Case 1: Honest Alice *vs.* Malicious Bob: This case falls under Scenario 2, which requires special treatment when arbiters are not contacted simultaneously for aborting. For threshold-based mechanisms, every DAFE protocol needs to have $T_A \leq T_B - |F_B|$. For the most general case of DAFE protocols, we need $\forall B_R \in \mathcal{B}_{\mathcal{R}}\ B_R \not\subseteq F_B$ AND $\exists A_R \in \mathcal{A}_{\mathcal{R}}\ s.t.\ A_R \cap F_B = \varnothing$ (see Lemma 7 in Appendix B.1).

Case 2: Malicious Alice *vs.* Honest Bob: This case falls under Scenario 1, which means any DAFE protocol needs to have $\forall A_R \in \mathcal{A}_{\mathcal{R}}\ \exists B_R \in \mathcal{B}_{\mathcal{R}}\ s.t.\ B_R \subseteq A_R - F_A$. Remember, Corollary 3 (using threshold-based mechanisms) require $T_B \leq T_A - |F_A|$.

Regarding DAFE protocols using threshold-based arbiter resolution mechanisms, we can conclude (from the two cases above) that no such meaningful protocol can exist ($T_A \leq T_B - |F_B|$ and $T_B \leq T_A - |F_A|$ gives $T_A \leq T_A - |F_A| - |F_B|$, which means all the arbiters need to be trusted). Hence, there is no point in distributing the arbiters in terms of trust. It is even worse since we need to trust every single arbiter, and the protocol cannot be fair even if only one arbiter is corrupt.

Regarding general set-based DAFE protocols, we cannot conclude an immediate impossibility. But following our discussion above, we conjecture that no such useful protocol can exist.

Unfortunately, as we have shown in Section 2.1, the versions of the state-of-the-art protocols we analyzed in Section 2.1 *without* any timeouts fall under this protocol category. So the impossibility with threshold-based mechanisms, and our conjecture apply to very common real cases, even when the arbiters are not contacted simultaneously by the honest party.

C Discussion: Timeouts and Dynamic Resolution Sets

As we have proved in Section 5, no realistic DAFE protocol can provide fairness, whereas Section 6 shows an existing DAFET protocol that employs timeouts. Therefore, we can conclude that timeouts play an important role in optimistic fair exchange protocols when we would like to employ multiple autonomous arbiters. Even without completely autonomous arbiters, Section B.2 shows an impossibility of DAFE protocols using threshold-based mechanisms, and even with set-based mechanisms, it is not clear how such a DAFE protocol can be constructed.

Timeouts are tied to the use of dynamic sets in general (as we did for DAFET protocols). When only one party can resolve before the timeout, static resolution sets lose their meaning since the resolution set for the party who cannot resolve before the timeout is empty until the timeout. That set gets defined only after the timeout, which results in that set being dynamic in a very basic sense. The dynamism prevents the adversary from coming up with a strategy that violates fairness. As shown in Section 6, this helps AV protocol achieve semantic fairness. Of course, a careful protocol design is still necessary since timeouts and dynamically changing sets by themselves do not mean that the protocol will be trivially fair. One may further argue that dynamically changing resolution sets is a more important concept that plays a big role in this (im)possibility result, but it is easy to see that timeouts are natural mechanisms to achieve this dynamism.

Speaker Recognition in Encrypted Voice Streams

Michael Backes[1,2], Goran Doychev[1], Markus Dürmuth[1], and Boris Köpf[2]

[1] Saarland University, Saarbrücken, Germany
[2] Max Planck Institute for Software Systems (MPI-SWS)

Abstract. Transmitting voice communication over untrusted networks puts personal information at risk. Although voice streams are typically encrypted to prevent unwanted eavesdropping, additional features of voice communication protocols might still allow eavesdroppers to discover information on the transmitted content and the speaker.

We develop a novel approach for unveiling the identity of speakers who participate in encrypted voice communication, solely by eavesdropping on the encrypted traffic. Our approach exploits the concept of voice activity detection (VAD), a widely used technique for reducing the bandwidth consumption of voice traffic. We show that the reduction of traffic caused by VAD techniques creates patterns in the encrypted traffic, which in turn reveal the patterns of pauses in the underlying voice stream. We show that these patterns are speaker-characteristic, and that they are sufficient to undermine the anonymity of the speaker in encrypted voice communication. In an empirical setup with 20 speakers our analysis is able to correctly identify an unknown speaker in about 48% of all cases. Our work extends and generalizes existing work that exploits variable bit-rate encoding for identifying the conversation language and content of encrypted voice streams.

1 Introduction

The past decades have brought dramatic changes in the way we live and work. The proliferation of networked devices, and the resulting abundance of exchanged information present significant opportunities, but also difficult conceptual and technical challenges in the design and analysis of the systems and programs that fuel these changes. A particularly important trend is the increasing need for protocols that run on open infrastructures such as wireless communication channels or the Internet and offer remote communication between different people anytime, anywhere. However, transmitting privacy-sensitive information over such open infrastructures raises serious privacy concerns. For instance, modern voice communication protocols should satisfy various security properties such as secrecy (the content of the voice communication should remain secret to eavesdroppers) or even anonymity (users participating in voice communications should remain anonymous to eavesdroppers in order to avoid being stigmatized or other negative repercussions). To achieve these security properties, voice communication is typically encrypted. For example, telephones based on the GSM [13] and UMTS [1] standards encrypt their voice data, and most implementations of VoIP

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 508–523, 2010.

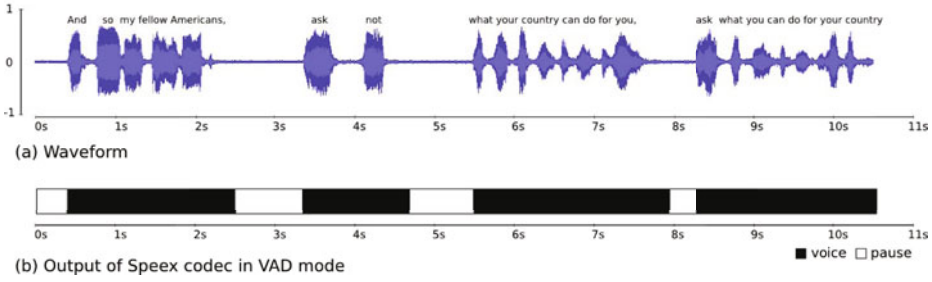


Fig. 1. Patterns of pauses in network traffic introduced when encoding an audio signal (above) with a VAD-enabled codec (below). Each audio packet is depicted as a 0.0145s long black- or white-colored bar, a black bar corresponding to a voice packet and a white bar corresponding to a pause packet. (Audio data: John F. Kennedy's inaugural address from January 20th, 1961).

telephony offer encryption on the application layer or support IPsec. The underlying rationale is that properly employing encryption hides both the content of the communication and the identity of the speaker, thereby ensuring both secrecy and anonymity. However, even properly deploying encryption does not exclude that additional features of voice communication protocols might still allow eavesdroppers to discover information about the transmitted content and the speaker.

1.1 Our Contribution

We develop a novel approach for unveiling the identity of speakers who participate in encrypted voice communication, solely by eavesdropping on the encrypted traffic. Our approach exploits the concept of *voice activity detection* (VAD), which is a common performance-enhancing technique to detect the presence or absence of human speech. VAD-based techniques are used to reduce the volume of the transmitted data and are prevalent in standards for transmitting voice streams. For example, the GSM and UMTS standards use a VAD-technique called discontinuous transmission (DTX) to stop the transmissions if a speaker is idle, thereby saving battery power and reducing interference. Moreover, VoIP clients such as Skype [29], Google Talk [12], and Microsoft Netmeeting [20], as well as the US Army's Land Warrior system, employ voice codecs that decrease the number and/or size of packets when a speaker is idle. This reduces network utilization, which is a primary concern in packet-switched computer networks.

We show that – even when traffic is encrypted – the reduction of traffic caused by VAD techniques creates patterns in the traffic, which in turn reveal patterns of pauses in the underlying voice stream (see Figure 1). We show that these patterns are speaker-characteristic, and that they are sufficient to undermine the anonymity of the speaker in encrypted voice communication.

Our approach relies on supervised learning and works as follows. In a preparation phase, we collect encrypted samples of voice stream data from a set of candidate speakers, the so-called *training data*. From this training data, we build a stochastic model of the pause characteristics for each candidate, i.e., the relative frequencies of durations of pauses and continuous speech. In the attack phase, we are faced with a sample of encrypted voice stream data of one of these candidates, the so-called *attack data*. We use the attack data to create a corresponding stochastic model of the pause characteristics of this candidate; after that, we employ standard and application-specific classifiers to perform a goodness-of-fit test between the stochastic models of each candidate and the one of the target candidate. The goodness-of-fit test determines the candidate whose model best matches the model derived from the attack data, yielding our guess for the target's identity.

We implemented our approach and conducted a series of experiments to evaluate its effectiveness. Our data set was composed of about 200 recorded speeches of 20 politicians. (We chose this data set because many speeches of all candidates are freely available on the web.) We encoded these speeches using the VAD-enabled voice codec Speex [36]. Speex is used by popular VoIP applications, such as Google Talk [12], TeamSpeak [30], Ekiga [9] and Microsoft Netmeeting [20]. We built the speaker models using the lengths of the plain (unencrypted) packets delivered by Speex. Our experiments showed that omitting encryption in building up these models does not affect our results, since there are large differences in length between packets corresponding to pauses and packets corresponding to speech (around a factor of six). These differences are not obscured by the largely length-preserving ciphers used for VoIP (which have a typical overhead of less than 10%). Our results show that – even in the presence of encryption – the information that VAD reveals is a serious threat to the identity of the speakers: In about 48% of all cases, we were able to correctly identify the speaker from the set of candidates.

1.2 Related Work

Most documented side-channel attacks against VoIP [35,34,17] target *variable bit-rate* (VBR) encoding. VBR is a bandwidth-saving encoding technique that is more specialized (and thus less often implemented) than VAD. When encoding a speech segment using VBR, the codec determines whether a high bit-rate is required for encoding the segment with sufficient audio quality, or if a lower bit-rate is already enough. The bit-rate used for encoding is reflected in the size of the resulting packets and is revealed despite encryption. The resulting patterns can be observed in the traffic of an encrypted VBR-encoded voice stream, as shown in Figure 2. For a comparison of VAD and VBR, observe that a VBR codec utilizes the lowest available bit-rate for encoding pauses; hence our attack against VAD also applies to VBR-codecs. However, in contrast to attacks against VBR, our attack also poses a threat in scenarios where pure VAD is used, e.g., in mobile communication.

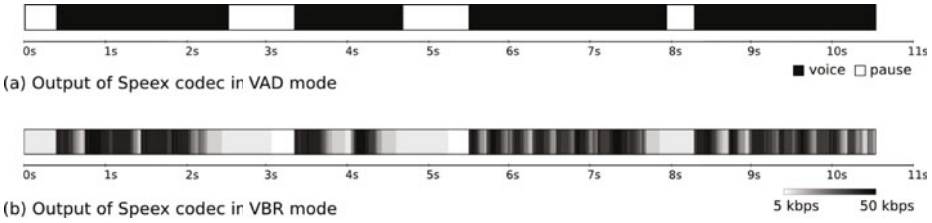


Fig. 2. Patterns in network traffic introduced when using a VAD-enabled codec (above) and a VBR-enabled codec (below). Each audio packet is depicted as a 0.0145s long colored bar, a lighter bar corresponding to a smaller bit-rate.

Wright et al. [35] exploit the patterns in encrypted VBR-encoded VoIP conversations to reveal which language is being spoken. In subsequent work [34] they even show that a sophisticated analysis of encrypted VBR-encoded VoIP traffic allows an attacker to (partially) uncover spoken phrases in a conversation. In contrast, our attack targets anonymity, i.e., it allows one to unveil the identity of the speaker.

Khan et al. [17] show how to reveal speaker identities from encrypted VoIP traffic. They exploit patterns introduced by VBR encoding, which is a much richer data source than the VAD encoding used in our work. Their speaker models are built using triples of packets and capture time intervals of approximately $60ms$, whereas our speaker models are based on triples of voice-/pause segments and capture time intervals of up to multiple seconds. What is particularly interesting is that the identification rates obtained by Khan et al. (75% for 10 speakers and 51% for 20 speakers) are comparable to ours (65% for 13 speakers and 48% for 20 speakers), even though their work is based on VBR instead of VAD.

In independent and concurrent work, Lu [19] considers the anonymity of speakers in encrypted VoIP communications. In contrast to our approach, she uses Hidden Markov Models (HMMs) for classification. The observed identification rates seem to be comparable to ours, however, her paper does not contain sufficient detail to allow for an in-depth comparison.

In the field of speaker recognition, there has been significant research on so-called *temporal features*, which include pause duration and frequency; phone, segmental, and word durations [11,23]; and patterns in the interaction between speakers, e.g., durations of turns in a conversation [23,25]. Besides speaker recognition, temporal features have been considered for other types of speaker classification. Pause duration and frequency, as well as syllable, consonant, vowel and sub-phonemic durations have been used for determining a speaker's age [26,27,18]. Word durations have been used for determining stress levels of a speaker [14]. Pauses in speech have also been used to identify deception [4].

For completeness, we briefly mention other known security issues in mobile and VoIP scenarios. Most importantly, a large number of weaknesses have been found in the underlying, often proprietary cipher algorithms (A5/1-3) that are

intended to ensure the secrecy of transmitted data in the GSM standard [5,3,8,7]. Moreover, there are a variety of known attacks against VoIP systems, e.g., denial of service attacks [37] and billing attacks [38]. We refer to [10] for a recent survey on such attacks.

1.3 Outline

The remainder of the paper is structured as follows. Section 2 explains how the speakers models are built. Section 3 introduces several measures for goodness-of-fit, i.e., for comparing speaker models. We present the empirical evaluations of our attack in Section 4 before we conclude in Section 5.

2 Building Speaker Profiles

In this section, we describe how a stochastic model of pause characteristics is built from the stream of packets that is delivered by a VAD-enabled voice codec. To this end, we distinguish speech from pauses by the duration of the packet. Short packets are assumed to be pauses. We then transform the packet sequence into a simpler *abstract voice stream* that retains only the lengths of runs of speech and pauses, where length is measured in terms of packet numbers. From these abstract voice streams we construct basic speaker profiles by determining the relative frequency of each pause duration. Lastly, we refine the basic speaker profiles by incorporating context information and applying clustering techniques.

2.1 Abstract Voice Streams

We assume that a voice stream is given as a sequence $v = [p_1, \ldots, p_n]$ of packets delivered by a VAD-enabled codec. As a first step, we transform the packet sequence v into an *abstract voice stream* $abs(v)$, which is the sequence of natural numbers that correspond to the maximal numbers of adjacent packets in v corresponding to pauses, speech phases, pauses, etc. For this, we assume a threshold packet size t that distinguishes between pause and speech packets; i.e., a packet p with $|p| \leq t$ is classified as a pause packet, and as a speech packet otherwise.[1] We formalize $abs(v)$ by the following recursive definition, where $+$ denotes list concatenation.

$$abs([]) := []$$
$$abs([p_1, \ldots, p_m] + w) := [m] + abs(w)$$

where m is the largest integer with

$$\forall i \in \{1, \ldots, m\} : \ |p_i| > t \quad \text{or} \quad \forall i \in \{1, \ldots, m\} : \ |p_i| \leq t \ .$$

[1] For example, using the Speex codec, the length of speech packets exceeds the length of pause packets by a factor of 6, and it is thus easy to find a suitable threshold t.

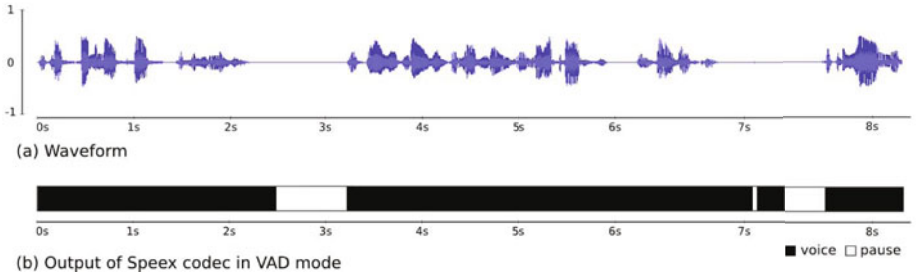


Fig. 3. Correlation between the audio signal (top), and the size of packets (bottom)

For example, for a sequence of packets v of sizes $[40, 45, 41, 2, 2, 3, 2, 50, 43]$ and a threshold of $t = 3$, we obtain the abstract voice stream $abs(v) = [3, 4, 2]$, which models the lengths of maximal sequences of speech, pause, and speech packets, respectively, in v. We will assume for simplicity that each abstract voice stream begins and ends with a speech phase, i.e., a stream has an odd length and the entries at odd positions correspond to speech phases.

For each entry d of an abstract voice stream we obtain the duration of the corresponding (speech or pause) phase in real time by dividing d by the packet frequency f, i.e., the number of packets per second. For the example of the Speex codec, we have a packet frequency of $f = 69\, s^{-1}$.

2.2 Adapting Abstract Voice Streams to the Codec's Characteristics

We have established a direct connection between packet numbers and durations of pause and speech phases using abstract voice streams. However, in order to capture the codec's characteristics, we must consider further extensions. Many codecs use a technique called *hangover* to avoid end-clipping of speeches and to bridge short pause segments such as those due to stop consonants [16]. When silence is detected, codecs delay for a *hangover period* before switching to pause mode. This delay can be either fixed or dynamic: dynamic hangover hardly influences the distribution of pause segments [16], and fixed hangover reduces the duration of pauses and increases the duration of voice segments. The hangover can be easily determined by comparing abstract voice streams to corresponding waveforms. In our experiments, we observed that the Speex codec uses a fixed hangover period of approx. 300ms ($\approx$ 21 packets) before switching to pause mode. (This can be seen at 2.5 sec. in Figure 3.) When the voice-signal starts again, there is (obviously) no delay for switching back to voice mode. (This can be seen at 3.2 sec. in Figure 3.)

Another artifact of the Speex codec can be observed when a very short noise signal occurs during a longer pause. In this case the codec immediately switches to voice, but switches back to pause mode earlier than after a long voice segment. (This can be seen at 7.1 sec. in Figure 3.)

To account for this codec behavior, we modify the abstract voice stream $abs(v) = [d_1, \ldots, d_k]$ as follows. First, we estimate the hangover period h; for Speex we obtain $h = 21$ packets; then

1. For each even i (corresponding to a pause), set $d_i := d_i + h$.
2. Update the previous (speech) entry d_{i-1} as follows. If $d_{i-1} > h$ then $d_{i-1} := d_{i-1} - h$. If $d_{i-1} \leq h$ then $d_{i-2} := d_{i-2} + d_{i-1} + d_i$ (ignore this assignment for $i = 2$), and delete the entries d_i and d_{i-1} from the sequence.

Modification (1) compensates for the hangover period h, as explained above. Modification (2) shortens the preceding speech entry accordingly and at the same time removes short noise signals from the stream.

Our experiments confirm that the resulting abstract voice stream more accurately captures the duration of pauses in the original voice stream, and we use it as the basis of our speaker profiles.

2.3 Basic Speaker Profiles

A basic speaker profile captures information about the relative frequencies of lengths of pauses or voice segments, respectively. As our experiments confirm, this information alone allows for good recognition results.

For an abstract voice stream $[d_1, \ldots, d_k]$, the relative frequency S^{pause} of pause durations d is defined as

$$S^{\text{pause}}[d] = \frac{\#\{j \mid d_{2j} = d\}}{(k-1)/2} \ .$$

Analogously, we define the relative frequency S^{voice} of the durations d of speech phases:

$$S^{\text{voice}}[d] := \frac{\#\{j \mid d_{2j+1} = d\}}{(k+1)/2} \ .$$

Given an abstract voice stream with packet lengths $[5, 10, 4, 7, 5, 7, 3]$ we obtain

$$S^{\text{pause}}[7] = \tfrac{2}{3} \quad S^{\text{pause}}[10] = \tfrac{1}{3}$$
$$S^{\text{voice}}[3] = \tfrac{1}{4} \quad S^{\text{voice}}[4] = \tfrac{1}{4} \quad S^{\text{voice}}[5] = \tfrac{1}{2}$$

By definition, S^{pause} and S^{voice} vanish for all packet sizes that do not occur in the abstract voice stream.

2.4 Advanced Speaker Profiles

The basic speaker profiles S^{pause} and S^{voice} presented above capture the relative frequencies of durations of pauses and continuous speech, respectively. As pauses and speech are considered in isolation, these models are oblivious of the context in which a pause or a speech phase occurs. To overcome this limitation, we construct a speaker profile based on the relative frequencies of three-tuples of durations of adjacent pause-voice-pause phases. By considering such three-tuples,

we incorporate interdependencies between sequences of pauses and speech into our model, which captures the context in which pauses occur.

We formally define S^3 as follows

$$S^3[(x,y,z)] = \frac{\#\{j \mid d_{2j-1} = x, d_{2j} = y, d_{2j+1} = z\}}{(k-1)/2} \ .$$

It is straightforward to generalize S^3 to arbitrary n-tuples. In our experiments, however, speaker profiles based on three-tuples have proven sufficient.

2.5 Clustering

The distributions of pause and voice durations are characteristic for a speaker. However, as with most natural processes, they are subject to small random disturbances. We therefore group the pause lengths to clusters: Given a sequence $[d_1, \ldots, d_k]$ we create a clustered version (with cluster-size s) of this sequence as

$$[\lceil d_1/s \rceil, \ldots, \lceil d_k/s \rceil] \ .$$

Unless otherwise specified, in the remainder of this paper we use a cluster-size of 80 (determined experimentally to yield good results). Applying this technique has the additional advantage of reducing the support of the distribution function. This is particularly relevant for the S^3 speaker model, as its support grows cubically in the number of observed durations.

3 Measuring Distance of Speaker Profiles

This section introduces three classifiers that serve as *goodness-of-fit* tests in this work; i.e., they compare how well the probability distribution over segment durations of the unknown speaker matches distributions over durations collected in the training phase. Thus these classifiers constitute tools to identify the victim of our attack from a set of candidate speakers.

3.1 The L_1-Distance

The simplest distance measure is the metric d_{L_1} induced by the L_1-norm. For probability distributions P, Q with finite support T, the metric d_{L_1} is defined as the sum of the absolute differences between the values of P and Q, i.e.,

$$d_{L_1}(P,Q) = \sum_{x \in T} |P[x] - Q[x]| \ .$$

Even though d_{L_1} is a rather simple measure, it performs reasonably well on our experimental data, as shown in Section 4.

3.2 The χ^2-Distance

A more sophisticated distance measure is the χ^2-distance, which is based on the χ^2-test. For two probability distributions P, Q with support T we define $d_{\chi^2}(P, Q)$ as the sum of the squared and normalized absolute differences between the values of P and Q, i.e.,

$$d_{\chi^2}(P, Q) = \sum_{x \in T} \frac{(P[x] - Q[x])^2}{Q[x]} \ .$$

Note that d_{χ^2} is not a metric in the mathematical sense, because it lacks symmetry. Besides this fact, the measure d_{χ^2} shows two main differences from the metric d_{L_1}. First, squaring the numerator for χ^2 gives more weight to large differences in the relative frequency of a given packet size. Second, dividing by the trained probability $Q[x]$ amplifies differences whenever $Q[x]$ is small, effectively giving the relative difference rather than the absolute difference. In our experiments, of the three classifiers, the χ^2-distance has shown the most robust performance.

3.3 The *K-S*-Distance

Finally, we derived a distance measure based on the Kolmogorov-Smirnov test, which is known to outperform the χ^2 on samples that are small or that are sparsely distributed throughout a large number of discrete categories [21]. We define the K-S-distance of two probability distributions P, Q with support $T = \{t_1, \ldots, t_n\}$ and $t_i \leq t_j$ whenever $i < j$, by

$$d_{K\text{-}S}(P, Q) = \max_{l \leq n} \left\{ \left| \sum_{i=1}^{l} (P(t_i) - Q(t_i)) \right| \right\} \ .$$

The K-S test searches for the maximal difference between the cumulation of two distributions. In our experiments, the K-S distance performed well, but slightly worse than the χ^2-distance.

3.4 Classifier Evaluation

Using the classifiers described above to compare the unknown speaker's model to the N trained models, we obtain a vector of scores $\langle s_1, s_2, \ldots, s_N \rangle$, s_i corresponding to the score of the unknown speaker's model when compared to the model of speaker i. From this vector, we compute the *rank*, representing the position at which the correct speaker was ranked. In case of a score tie, we take the lowest ranking position among all speakers with the same score. After t trials, we obtain the ranks $r_1, r_2, \ldots, r_t$, where r_i is the rank in the i-th trial. In the following we present several techniques for evaluating the performance of the classifiers using those values.

Identification rate. The simplest evaluation metric we consider is *identification rate* (IR). It is computed as the percentage of the trials where the classifier guessed correctly the unknown speaker, i.e.,

$$\text{IR} := \frac{\#\{i|r_i = 1\}}{t} .$$

The identification rate is an intuitive measure for the accuracy of classifiers. However, it is a quite conservative measure, as it ignores all results where the speakers are not ranked first. For our purposes we are not only interested in the best-scored speaker, but in a subset of the highest-ranked speakers. For example, if a classifier constantly gives the unknown speaker a rank of 3 out of 15 speakers, this still leaks information about the speaker's identity.

Average rank. An evaluation method that takes into consideration all obtained ranks is the *average rank* (AR) over all obtained ranks, i.e.,

$$\text{AR} := \sum_{i=1}^{t} \frac{r_i}{t} .$$

The results of this measure are very intuitive since they depict which position is output by the classifier on average; results closer to position 1 are preferred. However, as we are only interested in the few highest ranks, the use of average ranks may not be appropriate, as it puts equal weight on higher and lower ranks.

Top x results. To overcome the shortcomings of average ranks, we could observe only the top x obtained ranks. Thus, we obtain the *top_x*-metric which measures the percentage of trials where the correct speaker was ranked x-th or better, i.e.,

$$\text{top\_}x := \frac{\#\{i|r_i \leq x\}}{t} .$$

The plot with the rank x on the horizontal axis and the top_x metric on the vertical axis is called *cumulative match characteristic* (CMC) curve in the literature (e.g., see [22]), and we use it to illustrate the results of our experiments in Section 4.

Discounted cumulative gain. Alternatively, we could use an adapted version of *discounted cumulative gain* (DCG), a scoring technique used mainly in information retrieval for rating web search engine algorithms [15].

Let for $i \in \{1, \ldots, N\}$, the relevance rel_i be defined as number of trials where the correct speaker was ranked i-th. The DCG-measure is defined as

$$\text{DCG} := \sum_{i=1}^{N} \frac{rel_i}{d(i)} ,$$

where $d(i)$ is called *discounting function* and usually $f(i) = \log_2(i+1)$ is applied. Using this measure, top-ranked speakers will have a higher weight than lower-ranked ones, but lower ranks will still have a relevance to the final score of a classifier.

4 Experimental Evaluation

In this section we report on experimental results where we evaluate the feasibility of breaking anonymity in encrypted voice streams. We first describe our experimental setup and proceed by discussing the results we obtained using the speaker profiles and distance measures presented in the previous sections.

4.1 Experimental Setup

We use speeches of 20 different politicians as a data basis for our experiments: Among those 20 speakers, 18 speakers are male and 7 languages are spoken, English being the best represented language with 12 speakers, see Table 1. This set of voice recordings is homogeneous with respect to the setting in which the speeches were given, as they are official addresses to the nation that were broadcast on radio or television. The collected speeches are available online, e.g. on [33], [2], [31] and [32]. The length of the collected audio data per speaker varied between 47 and 114 minutes; on average we have about ten speeches per speaker. The speeches for each speaker were recorded in the course of several months or even years in multiple recording situations.

We simulate a unidirectional voice conversation by encoding the speeches using Speex (version 1.2rc1). We build our speaker models based on (sequences of) the sizes of audio packets output by Speex. These packet sizes correspond to the lengths of the speech packets in encrypted VoIP traffic, except for a constant offset. To see this, note that the encryption schemes used for VoIP are largely length-preserving. Moreover, typical protocols for transmitting audio packets on the application layer add headers of constant size, e.g., the commonly used Real-time Transport Protocol (RTP) [28]. As a consequence, the speaker models built from sequences of plain audio packets are equivalent to the models built from real VoIP traffic.

4.2 Results and Discussion

We performed our experiments on the full set of 20 speakers and on a subset of 13 speakers.[2] We divided the voice data of each speaker into two halves, each consisting of several speeches; we used the first half for training and the second half for the attack and vice versa, resulting in a total of 26 experiments with 13 speakers and 40 experiments with 20 speakers, respectively. We performed experiments with all speaker models discussed in Section 2, i.e., based on sequences of pause lengths (S^{pause}), sequences of speech lengths (S^{voice}), and three-tuples thereof (S^3). Moreover, we considered variants of each model based on clustering, and we compensated for the hangover technique used by Speex, as discussed in Section 2. As distance measures, we used the L_1 distance, the χ^2 classifier,

[2] The 13 speakers were our data set for the initial version of this paper [6], which we extended to 20 speakers for the final version.

Table 1. Speech data used in the experiments

Speaker	Nationality	Language	Number speeches	Duration (mm:ss)
Angela Merkel	Germany	German	15	53:53
Barack Obama	USA	English	15	68:33
Cristina Fernández de Kirchner	Argentina	Spanish	5	99:04
Dmitry Medvedev	Russia	Russian	12	66:40
Donald Tusk	Poland	Polish	10	92:38
Dwight D. Eisenhower	USA	English	7	67:28
Franklin D. Roosevelt	USA	English	4	80:38
George W. Bush	USA	English	15	50:24
Harry S. Truman	USA	English	5	60:48
Jimmy Carter	USA	English	6	47:56
John F. Kennedy	USA	English	8	47:10
Kevin Rudd	Australia	English	16	68:55
Luiz Inácio Lula da Silva	Brazil	Portuguese	7	105:27
Lyndon B. Johnson	USA	English	8	50:25
Nicolas Sarkozy	France	French	5	102:58
Richard Nixon	USA	English	6	113:43
Ronald Reagan	USA	English	12	51:06
Stephan J. Harper	Canada	English/French	13	100:07
Vladimir Putin	Russia	Russian	13	113:55
William J. Clinton	USA	English	20	82:05

and the Kolmogorov-Smirnov (K-S) classifier. Moreover, we evaluated and compared the performance of these classifiers when conducting those experiments, i.e., we analyzed the classifiers' identification rate (IR), average rank (AR), and the discounted cumulative gain (DCG), as discussed in Section 3.4.

For a data set of 13 speakers, we obtained the following results. Using the speaker model S^{pause}, the identification rate ranged between 26.9% and 38.5% depending on the used classifier, see Table 2(a). Using the speaker model S^{voice}, the identification rate ranged between 30.8% and 50%, see Table 3(a).

For 13 speakers, our best results were obtained using the speaker model S^3 and applying a clustering with cluster size 80 to reduce the support of the distribution function. For S^3, the identification rate ranged between 30.8% and 65.4%, as shown in Table 4(a). For comparison, observe that the probability of randomly guessing the correct speaker is $\frac{1}{13} \approx 7.7\%$, i.e., we achieve an 8.5-fold improvement over random guessing.

For a data set of 20 speakers, we obtained the following results. Using the speaker model S^3, we obtained identification rates between 22.5% and 40% depending on the classifier, as shown in Table 4(b). As in the setting with 13 speakers, S^3 outperforms the speaker model S^{pause}. However, as opposed to the setting with 13 speakers, we obtained the best identification rates for 20 speakers using the S^{voice} model: with this model, the identification rate ranged between 32.5% and 47.5%, see Table 3(b). The probability of randomly guessing the correct speaker is $\frac{1}{20} = 5\%$, i.e., we achieve a 9.5-fold improvement over random

guessing. Although the identification rate decreases when considering 20 speakers instead of 13 (which was expected), the improvement over random guessing is almost constant for both data sets.

Our discussion so far has focused on identification rate as a metric for evaluating classifiers. The reason for choosing identification rate is its direct and intuitive interpretation. The results of evaluating classifiers and speaker models

Table 2. Experimental results with different classifiers using speaker models based on pauses (S^{pause}). (IR = identification rate, AR = average rank, DCG = discounted cumulative gain).

(a) Results with 13 speakers

Classifier	IR	AR	DCG
L_1	0.269	3.000	0.619
χ^2	0.385	2.423	0.697
K-S	0.308	3.615	0.613
Random	0.077	7	0.412
Best case	1	1	1

(b) Results with 20 speakers

Classifier	IR	AR	DCG
L_1	0.275	5.050	0.572
χ^2	0.175	4.475	0.545
K-S	0.225	5.525	0.542
Random	0.050	10.5	0.352
Best case	1	1	1

Table 3. Experimental results with different classifiers using speaker models based on voice segments (S^{voice}). (IR = identification rate, AR = average rank, DCG = discounted cumulative gain).

(a) Results with 13 speakers

Classifier	IR	AR	DCG
L_1	0.500	2.808	0.729
χ^2	0.577	2.692	0.763
K-S	0.308	3.731	0.611
Random	0.077	7	0.412
Best case	1	1	1

(b) Results with 20 speakers

Classifier	IR	AR	DCG
L_1	0.425	4.675	0.652
χ^2	0.475	4.625	0.679
K-S	0.325	5.050	0.594
Random	0.050	10.5	0.352
Best case	1	1	1

Table 4. Experimental results with different classifiers using speaker models based on three-tuples of pauses and voice segments (S^3). (IR = identification rate, AR = average rank, DCG = discounted cumulative gain).

(a) Results with 13 speakers

Classifier	IR	AR	DCG
L_1	0.654	2.692	0.789
χ^2	0.615	2.192	0.801
K-S	0.308	3.731	0.615
Random	0.077	7	0.412
Best case	1	1	1

(b) Results with 20 speakers

Classifier	IR	AR	DCG
L_1	0.300	4.725	0.619
χ^2	0.400	4.050	0.670
K-S	0.225	5.075	0.547
Random	0.050	10.5	0.352
Best case	1	1	1

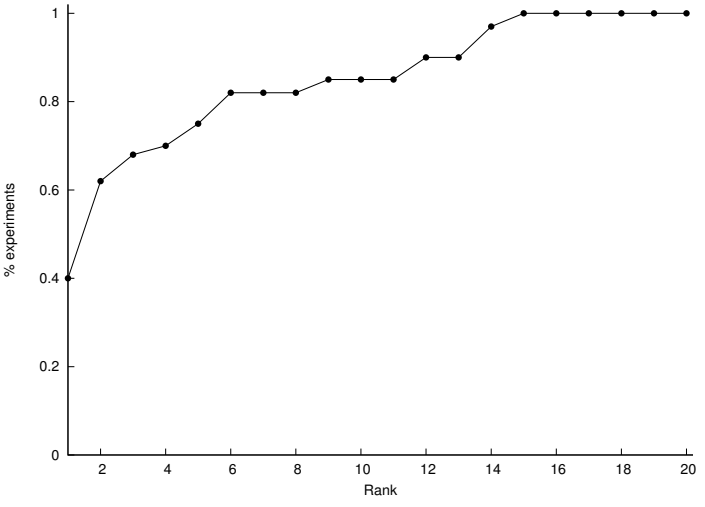


Fig. 4. Cumulative match characteristic (CMC) curve for the χ^2 classifier with 20 speakers, using speaker models based on three-tuples (S^3): the horizontal axis depicts the rank i assigned by the classifier; the vertical axis denotes the percentage of experiments in which the correct speaker was assigned at least rank i

using different metrics are also given in Tables 2, 3, 4, and Figure 4, respectively. We believe that these alternative metrics are relevant in terms of their security interpretation. For example, the top_x-metric seems to be closely connected to the notion of anonymity sets of size x [24]. We leave a thorough investigation of this connection to future work.

5 Conclusion

Performance-enhancing techniques such as voice activity detection create patterns in the volume of telephone traffic that are observable by eavesdroppers even if the traffic is encrypted. In turn, these patterns reveal patterns of pauses in the underlying voice stream. We have developed a novel approach for unveiling the identity of speakers who participate in encrypted voice communication: we show that these patterns are characteristic for different speakers, and that they are sufficient to undermine the anonymity of the speaker in encrypted voice communication. In an empirical setup with 20 speakers our analysis is able to correctly identify an unknown speaker in about 48% of all cases. This raises serious concerns about the anonymity in such conversations and is particularly relevant for communication from mobile and public devices.

References

1. 3GPP. The 3rd Generation Partnership Project, http://www.3gpp.org/
2. Administration of the President of the Russian Federation. Videoblog of the President of the Russian Federation, http://blog.kremlin.ru/

3. Barkan, E., Biham, E., Keller, N.: Instant ciphertext-only cryptanalysis of GSM encrypted communication. Journal of Cryptology 21(3), 392–429 (2008)
4. Benus, S., Enos, F., Hirschberg, J., Shriberg, E.: Pauses in deceptive speech. In: Proc. of ISCA 3rd International Conference on Speech Prosody (2006)
5. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
6. Doychev, G.: Speaker recognition in encrypted voice streams, Bachelor's thesis, Department of Computer Science, University of Saarland, Saarbrücken, Germany (December 2009)
7. Dunkelman, O., Keller, N., Shamir, A.: A practical-time attack on the A5/3 cryptosystem used in third generation GSM telephony. Cryptology ePrint Archive, Report 2010/013 (2010), http://eprint.iacr.org/
8. Ekdahl, P., Johansson, T.: Another attack on A5/1. IEEE Transactions on Information Theory 49(1), 284–289 (2003)
9. Ekiga, http://ekiga.org/
10. El-Moussa, F., Mudhar, P., Jones, A.: Overview of SIP attacks and countermeasures. In: Information Security and Digital Forensics. LNICST, pp. 82–91. Springer, Heidelberg (2010)
11. Ferrer, L., Bratt, H., Gadde, V.R.R., Kajarekar, S., Shriberg, E., Andreas, K.S., Venkataraman, S.A.: Modeling duration patterns for speaker recognition. In: Proc. of the EUROSPEECH, pp. 2017–2020 (2003)
12. Google Inc. Google Talk, http://www.google.com/talk/
13. GSM-Association. GSM - Global System for Mobile communications, http://www.gsmworld.com/
14. Hansen, J.H., Patil, S.: Speech under stress: Analysis, modeling and recognition. In: Müller, C. (ed.) Speaker Classification 2007. LNCS (LNAI), vol. 4343, pp. 108–137. Springer, Heidelberg (2007)
15. Järvelin, K., Kekäläinen, J.: IR evaluation methods for retrieving highly relevant documents. In: Proc. of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 41–48. ACM Press, New York (2000)
16. Jiang, W., Schulzrinne, H.: Analysis of on-off patterns in VoIP and their effect on voice traffic aggregation. In: Proc. of the 9th International Conference on Computer Communications and Networks (ICCCN 2000), pp. 82–87 (2000)
17. Khan, L., Baig, M., Youssef, A.M.: Speaker recognition from encrypted VoIP communications. Digital Investigation (2009) (in Press)
18. Linville, S.E.: Vocal Aging. Singular (2001)
19. Lu, Y.: On traffic analysis attacks to encrypted VoIP calls, Master's thesis, Cleveland State University (November 2009)
20. Microsoft Corporation. Microsoft Netmeeting, http://www.microsoft.com/downloads/details.aspx?FamilyID=26c9da7c-f778-4422-a6f4-efb8abba021e&displaylang=en
21. Mitchell, B.: A comparison of chi-square and Kolmogorov-Smirnov tests. Area 3, 237–241 (1971)
22. Moon, H., Phillips, P.J.: Computational and performance aspects of PCA-based face-recognition algorithms. Perception 30, 303–321 (2001)
23. Peskin, B., Navratil, J., Abramson, J., Jones, D., Klusacek, D., Reynolds, D.A., Xiang, B.: Using prosodic and conversational features for high-performance speaker recognition. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003), pp. 792–795 (2003)

24. Pfitzmann, A., Köhntopp, M.: Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 1–9. Springer, Heidelberg (2001)
25. Reynolds, D., Campbell, J., Campbell, B., Dunn, B., Gleason, T., Jones, D., Quatieri, T., Quillen, C., Sturim, D., Torres-Carrasquillo, P.: Beyond cepstra: Exploiting high-level information in speaker recognition. In: Proc. of the Workshop on Multimodal User Authentication, Santa Barbara, Calif, USA, pp. 223–229 (December 2003)
26. Schötz, S.: Acoustic analysis of adult speaker age. In: Müller, C. (ed.) Speaker Classification 2007. LNCS (LNAI), vol. 4343, pp. 88–107. Springer, Heidelberg (2007)
27. Schötz, S., Müller, C.: A study of acoustic correlates of speaker age. In: Müller, C. (ed.) Speaker Classifcation II. LNCS (LNAI), vol. 4441, pp. 1–9. Springer, Heidelberg (2007)
28. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A transport protocol for real-time applications (1996)
29. Skype Limited. Skype, http://www.skype.com/
30. TeamSpeak Systems GmbH. TeamSpeak, http://www.teamspeak.com/
31. The American Presidency Project. Audio/Video Archive, http://www.presidency.ucsb.edu/media.php
32. The Press and Information Office of the German Federal Government. Podcasts, http://www.bundeskanzlerin.de/Webs/BK/De/Aktuell/Podcasts/podcast.html
33. The White House. Your weekly address, http://www.whitehouse.gov/briefing-room/weekly-address
34. Wright, C.V., Ballard, L., Coull, S.E., Monrose, F., Masson, G.M.: Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In: Proc. of the 2008 IEEE Symposium on Security and Privacy, pp. 35–49. IEEE Computer Society Press, Los Alamitos (2008)
35. Wright, C.V., Ballard, L., Monrose, F., Masson, G.M.: Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In: Proc. of 16th USENIX Security Symposium on USENIX Security Symposium, pp. 1–12. USENIX Association (2007)
36. Xiph.Org. Speex: A free codec for free speech, http://speex.org/
37. Zhang, G., Ehlert, S., Magedanz, T., Sisalem, D.: Denial of service attack and prevention on SIP VoIP infrastructures using DNS flooding. In: Proc. of 1st International Conference on Principles, Systems and Applications of IP Telecommunications, pp. 57–66. ACM, New York (2007)
38. Zhang, R., Wang, X., Yang, X., Jiang, X.: Billing attacks on SIP-based VoIP systems. In: Proc. of the First USENIX Workshop on Offensive Technologies, pp. 1–8. USENIX Association (2007)

Evaluating Adversarial Partitions

Andreas Pashalidis and Stefan Schiffner

K.U. Leuven/IBBT, ESAT/SCD-COSIC
Kasteelpark Arenberg 10,
Leuven, Belgium
{andreas.pashalidis,stefan.schiffner}@esat.kuleuven.be

Abstract. In this paper, we introduce a framework for measuring unlinkability both per subject and for an entire system. The framework enables the evaluator to attach different sensitivities to individual items in the system, and to specify the severity of different types of error that an adversary can make. These parameters, as well as a threshold that defines what constitutes a privacy breach, may be varied for each subject in the system; the framework respects and combines these potentially differing parametrisations. It also makes use of graphs in a way that results in *intuitive* feedback of different levels of detail. We exhibit the behaviour of our measures in two experimental settings, namely that of adversaries that output randomly chosen partitions, and that of adversaries that launch attacks of different effectiveness.

1 Introduction

An adversary on unlinkability aims to divide a given set of elements into non-overlapping clusters, such that the elements in each cluster belong to the same subject or, more generally, share a well-defined property. In the electronic world, such elements are typically digital data items that arise as a result of online transactions, e.g. personal messages, shopping records, user attributes, protocol transcripts, or entries in an audit log. Measuring linkability is important because an adversary's ability to link elements that should be unlinkable constitutes a privacy breach. Moreover, successful attacks on linkability can lead to further privacy breaches such as the unauthorised (re-)identification of subjects [13]. We stress, however, that an adversary on unlinkability does not necessarily care about identifying subjects.

Linkability measurements are not straightforward. Consider, for example, the partition $\Pi = \{\{\circ, \circ, \circ, \circ, \circ\}, \{\bullet, \bullet\}\}$, and let us call its two clusters 'Alice's and Bob's items'. An adversary assuming that the correct partition is $\{\{\circ, \circ, \circ, \circ, \circ\}, \{\bullet\}, \{\bullet\}\}$, obviously failed to link Bob's items, but has linked Alice's items perfectly. It is, however, not obvious which of the partitions, $\Pi'_1 = \{\{\circ, \circ, \circ, \circ\}, \{\circ, \bullet, \bullet\}\}$, $\Pi'_2 = \{\{\circ, \circ, \circ, \bullet\}, \{\circ, \circ\}, \{\bullet\}\}$, and $\Pi'_3 = \{\{\circ, \circ, \circ\}, \{\circ\}, \{\circ\}, \{\bullet, \bullet\}\}$, for example, is a better approximation of Π, both overall and with respect to any given subject. In order to demonstrate this, we first focus on Alice's items but ignore, for the moment, Bob's. According to Π'_1, almost all her items have

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 524–539, 2010.

been identified as belonging together, with the exception of one, which belongs to a different cluster. Π'_2 also divides Alice's items into two clusters, but the second one contains two items rather than only one. Intuition therefore suggests that Π'_2 is a worse approximation than Π'_1, at least with respect to Alice's items. Π'_3 divides Alice's items into three clusters. However, one has to re-allocate the same amount of items (two) in both Π'_2 and Π'_3 to completely link Alice's items. This, however, does not necessarily mean that both clusters are equally good approximations; Π'_3 may still be considered worse than Π'_2 since, in Π'_2, a single cluster *merging* suffices to completely link Alice's items, while, in Π'_3, two such mergings are required. We now stop ignoring Bob's items, but still focus on unlinkability from Alice's point of view. While Π'_1 contains a cluster that mixes two of Bob's items with some of Alice's, Π'_2 mixes only one. Moreover, Π'_3 contains no cluster that mixes items of both Alice and Bob. This second viewpoint leads to opposite conclusions: Π'_3 is a better approximation of Π than Π'_2, and Π'_2 is better than Π'_1.

In order to decide which of the two viewpoints should prevail, it is important to know more about the concrete context. A loan seeker, for example, would like to prevent his bank from being able to link his negative credit ratings, but if linking happens anyway, then he is likely to prefer correct rather than incorrect inferences to be made.[1] A consumer, on the other hand, would like to prevent the direct marketing company from being able to aggregate his shopping behaviour into a detailed profile, but if this happens anyway, he is likely to prefer his profile to be 'contaminated' with the shopping histories of other people. Both the loan seeker and the consumer are therefore interested in both viewpoints, but have different preferred outcomes with respect to the latter viewpoint.

In this paper, we develop a novel evaluation framework for adversarial partitions that enables one to take these different viewpoints. The framework is flexible in multiple ways, as it enables the evaluator to attach different sensitivities to individual items in the system, to specify the severity of different types of error that an adversary can make, and to define a threshold that defines what constitutes a privacy breach. These parametrisations may vary for each subject in the system; the framework respects and combines them. Evaluations are communicated by means of intuitive graphs and, since they are performed on the subject level, can serve as the basis for further analysis such as fairness.

The rest of this paper is organised as follows. The next section surveys related work, and section 3 introduces our evaluation framework. Section 4 illustrates its application on a toy example. Section 5 compares the behaviour of our measures to that of other distance measures. Finally, section 6 concludes.

2 Related Work

Two research areas relate to our work, namely that of privacy and that of statistical classification. In particular, we build on ideas on measuring unlinkability

[1] We assume that the loan seeker is willing to pay back his own debts in order to erase negative credit ratings if necessary, but unwilling to do so for other people's debts.

from the former, and comparing clusterings from the latter area. Works from the first area deal with the question of how effectively certain privacy preserving systems protect the unlinkability of the elements that arise in the system. While [3,4,9,11], for example, measure unlinkability in general, [13,14] and [1] do so in the context of anonymous communication and attribute sharing, respectively. In addition, [11] considers fairness.

The literature on comparing clusterings, on the other hand, has a longer history, and many distance metrics on partitions have been defined and used over the years [5,15]. The Rand index [12], for example, considers the extent to which two partitions treat all element pairs similarly, the minimum transfer distance [2] considers the number of element *transfers* until two partitions are identical and the variation of information [7] uses information-theoretic primitives.

The overall goal of both research areas is to provide the basis to compare partitions. The difference is that, in the 'measuring unlinkability' area, these partitions represent attacks on specific privacy preserving systems, while, in the 'comparing clusterings' context, they represent algorithms that aim to classify the items of a given dataset in some useful way. Unfortunately, the approaches from both areas suffer from certain inflexibilities that limit their suitability when it comes to evaluating adversarial partitions in a privacy setting. These inflexibilities manifest themselves in three different ways, as follows.

Firstly, existing approaches do not take into account important aspects of adversarial partitions. While [14], for example, mainly focuses on the question of whether or not *two* given elements are linked, [3,9] as well as the literature on comparing clusterings, compute measurements over a given partition as a whole. Both approaches do not measure unlinkability on the crucial *subject* level. Other works, e.g. [4,11,14], also consider the (degree of) unlinkability of arbitrary element subsets; their focus is, however, on the extent to which the elements of the subset can be linked, while the extent to which foreign elements (i.e. elements not in the subset) 'contaminate' the adversary's view on the subset, are disregarded. (Here it should be mentioned that this is not entirely true for the 'white-box' analysis approach described in [3]; depending on the chosen partition distance metric, contamination levels may be taken into account. This, however, does not happen on the subject level, but rather on the overall partition level.) In this work, we ask the question 'how well is a given *solution cluster* hidden within the adversarial partition?'. That is, we perform separate measurements pertaining to each subject without ignoring foreign elements (i.e. elements belonging to other subjects), and, if necessary, combine these into an overall average only in the last step. This approach yields not only measurements on the subject level, but also natural ways to evaluate the *fairness* provided by the underlying system.

Secondly, existing approaches do not distinguish between the sensitivity that users or evaluators attach to the elements in a system. They also do not let the evaluator specify his sensitivity towards different types of error that an adversarial partition may exhibit. That is, existing approaches do not account for the fact that the same adversarial partition may represent attacks of different

seriousness in different contexts. Our evaluation framework enables the evaluator to formulate such sensitivities.

Thirdly, it is unclear how to construct supportive material, such as illustrations or graphics, that show, in a sufficiently intuitive way, how a given (un)linkability measurement comes about. While the graphs that represent an attacker's internal state as defined in [9] are certainly an exception, they do not convey the information of how well a given subject's cluster is hidden within the adversary's state (especially in the presence of 'transitivity contradictions'). Our evaluation framework uses graphs that depict meaningful quantities that individual subjects are likely to care about. In fact, in our framework, these graphs do not depict the final unlinkability measurements. Rather conversely, the final unlinkability measurements are *derived* from the graphs.

Our approach remains agnostic to specific applications, and combines ideas from both research areas above. From the literature of measuring unlinkability we follow the idea that unlinkability generally decreases as the adversary links more of a given subject's elements, and we take into account adversaries that output multiple, in their view probable partitions. From the literature on comparing clusterings, we adopt some (very) basic notions of [7]. Namely, the intersection of cluster pairs is an important quantity that defines, among other parameters, both 'miss' and 'include' error counts in our approach. These same intersections also play a central role in the 'variation of information' metric (see Equation 15 in [7]). However, it is not our goal to define distance metrics in the strict sense; therefore our framework does not aim to fulfill the axioms put forth in [8]. It does, however, fulfill the informal criteria listed in [3], namely taking into account both the certainty and the consistency of the adversary.

3 Evaluating Adversarial Partitions

Our evaluation methodology focuses on the errors made by an adversary, and distinguishes between primary and secondary errors. The motivation for this distinction lies in the 'sort of story' that the two error types tell: primary errors describe the adversary's current state, while secondary errors describe the risk that this state represents for the future. More precisely, primary errors describe how well a given solution cluster is *currently* hidden within a given adversarial partition, while secondary errors describe how well the cluster *remains* hidden if additional information would enable the adversary to further refine its current assessment. In the following, Π denotes the solution partition, and Π' the adversarial partition. We assume that both Π and Π' are set partitions of the *same* finite set of size $n = \sum_{C \in \Pi} |C|$ and use the notation C and C' to refer to individual clusters of Π and Π', respectively.

3.1 Primary Errors

Motivated by the example in the introduction, we consider two types of primary error that an adversary can make with respect to a solution cluster: a 'miss' error

occurs if the adversary fails to include an element that should be included in the cluster, and an 'include' error occurs if it includes an element that should not have been included. Formally, given a cluster pair C, C', the number of miss and include errors is defined as $m(C, C') = |C - C'|$ and $i(C, C') = |C' - C|$, respectively. The miss (resp. include) error counts can also be defined as $m(C, C') = |C| - |C \cap C'|$ (resp. $i(C, C') = |C'| - |C \cap C'|$). This makes explicit the intersection mentioned in section 2.

The evaluator may be more sensitive towards miss than he is towards include errors, or vice versa. We let the evaluator indicate this sensitivity by means of a 'policy parameter' $\alpha \in [0, 1]$: setting $\alpha = 0$ indicates that he cares exclusively about the extent to which the elements of interest are linked, while completely ignoring foreign elements. Setting $\alpha = 1$, on the other hand, means that the evaluator is agnostic to the extent to which the adversary has managed to link the elements, and is only interested in the extent to which foreign elements are being mixed together with the correct ones. If both error types are to be deemed equally important, then the evaluator has to set $\alpha = 1/2$.

We are interested to count errors per subject, i.e. *per solution cluster*. One approach would be, given a solution cluster, to average the error counts over all adversarial clusters. However, since only few adversarial clusters are likely to be related to any given solution cluster C, we do not follow this approach. Instead we use only the *most relevant* adversarial cluster as the basis to count primary errors. This cluster is one of the adversarial clusters that have at least one element in common with C, and we use the policy parameter α to identify which one it is. We proceed as follows. Given C, we first determine the subset of related clusters $L(C, \Pi') = \{C' \in \Pi' : |C \cap C'| \geq 1\}$. Then, we calculate the *priority* for each cluster in $L(C, \Pi')$ as $p_\alpha(C, C') = (\alpha m(C, C') + (1 - \alpha) i(C, C'))^{-1}$. The most relevant adversarial cluster is the one with the highest priority.

If two or more related adversarial clusters yield this maximum, then we take the average of their error counts. The complete formal procedure is therefore as follows. Given a solution cluster C, we first determine the set of most relevant adversarial clusters as $R_\alpha(C, \Pi') = \{C' \in L(C, \Pi') : p_\alpha(C, C') = \max_{C' \in L(C,\Pi')} p_\alpha(C, C')\}$. Then the error counts for solution cluster C with respect to an adversarial partition Π' are given by $m(C, \Pi', \alpha) = \sum_{C' \in R_\alpha(C,\Pi')} m(C,C')/|R_\alpha(C,\Pi')|$ and $i(C, \Pi', \alpha) = \sum_{C' \in R_\alpha(C,\Pi')} i(C,C')/|R_\alpha(C,\Pi')|$. Finally, the total error counts are obtained simply by adding up the per solution cluster counts: $m(\Pi, \Pi', \alpha) = \sum_{C \in \Pi} m(C, \Pi', \alpha)$ and $i(\Pi, \Pi', \alpha) = \sum_{C \in \Pi} i(C, \Pi', \alpha)$.

Remark 1. If $m(\Pi, \Pi', \alpha) > i(\Pi, \Pi', \alpha)$, then we call Π' 'conservative', if $m(\Pi, \Pi', \alpha) < i(\Pi, \Pi', \alpha)$ then we call it 'liberal', and 'neutral' in case of equality. If $\alpha = 0$ (resp. $\alpha = 1$), then the evaluator prefers conservative rather than liberal (resp. liberal rather than conservative) adversarial partitions. Consider an adversary thinking that, by default, all elements are unlinked (resp. linked), and links (resp. unlinks) elements only if it observes strong evidence in support of this. Roughly speaking, in the presence of uncertainty, such an adversary is likely to make more miss (resp. include) rather than include (resp. miss) errors, and thus end up with a conservative (resp. liberal) partition.

Remark 2. Note that $m(\Pi, \Pi', \alpha)$ and $i(\Pi, \Pi', \alpha)$ as well as their sum can be seen as distance measures between the two partitions. One should keep in mind, however, that they are asymmetric. If, for example, $\Pi = \{\{\circ, \bullet, \odot\}\}$ and $\Pi' = \{\{\circ, \bullet\}, \{\odot\}\}$, then $m(\Pi, \Pi', 1/2) = 1$ but $m(\Pi', \Pi, 1/2) = 0$, and $i(\Pi, \Pi', 1/2) = 0$ but $i(\Pi', \Pi, 1/2) = 3$. We do not consider this asymmetry to be a problem because we do not aim to specify a real distance metric over the partition space. It is nevertheless important that our measures behave consistently with real distance metrics; see section 5 for an examination in this respect.

Combining and normalising primary errors. For a given policy parameter α and solution cluster C, the combined error count with respect to an adversarial partition Π' is defined by $e(C, \Pi', \alpha) = \alpha m(C, \Pi', \alpha) + (1 - \alpha) i(C, \Pi', \alpha)$. In order to normalise this combined error count, we consider the worst case, i.e. the adversarial partition that maximises it. Assuming that $n \geq 2$, the worst case occurs if $\alpha = 0$, $|C| = 1$ and Π' is a singleton: in this case, there occur $m(C, \Pi', \alpha) = |C| - 1 = 0$ miss errors and $i(C, \Pi) = n - |C| = n - 1$ include errors, and, thus, $n - 1$ errors in total. We therefore define the normalised error count of Π' with respect to C as $e^*(C, \Pi', \alpha) = {}^{e(C,\Pi',\alpha)}/_{n-1}$. Note that $0 \leq e^*(C, \Pi', \alpha) \leq 1$; it is zero if the adversary has made no errors that the evaluator cares about, and one in the worst case just described. Also note that normalisation is possible only if $n \geq 2$. The overall normalised error count is the average $e^*(\Pi, \Pi', \alpha) = \sum_{C \in \Pi} {}^{e^*(C,\Pi',\alpha)}/_{|\Pi|}$.

3.2 Secondary Errors

If an adversarial partition contains many more clusters than the solution partition, then primary error counts are bound to give an incomplete picture, because they ignore the exact structure of all but the most relevant clusters. We are thus interested in more detailed evaluation, and, in particular, in the number of mergings that are required until a given percentage of elements of a particular solution cluster have been linked in the adversarial partition. The smaller the number of required mergings, the better the partition. However, every merging potentially brings with it foreign elements, and the sensitivity towards the presence of such elements is a matter of the evaluator's policy.

Moreover, we must decide in which order the mergings are performed until the target percentage is reached. Our `Plot` algorithm, shown in Fig. 1, makes use of the evaluator's policy parameter in order to establish this ordering. Given a solution cluster C, the adversarial partition Π', the policy parameter α, and a bit b that indicates whether or not contamination is desirable from the subject's point of view (if $b = 1$ then contamination is desirable)[2], the algorithm produces a plot of two graphs that communicates the quality of the adversarial partition with respect to C in an intuitive way. The *linked* graph, in particular, shows how quickly consecutive mergings lead to element linking, and the *mixed* graph shows how quickly foreign elements are mixed into the cluster as a result of

[2] The loan seeker (resp. consumer) from the introduction would set $b = 0$ (resp. $b = 1$).

the same mergings. The worst-case complexity of our implementation of the **PlotForCluster** algorithm is $O(n^2)$.

PlotForCluster (input: C, Π', α, b)

1. For all $C' \in L(C, \Pi')$, compute the priority $p_\alpha(C, C')$. Then order $L(C, \Pi')$ according to the priority, highest value first. Resolve ties by giving priority to clusters with fewer foreign elements. Resolve remaining ties arbitrarily.
2. Start with an empty cluster X.
3. For values of $j = 1$ until $|L(C, \Pi')|$, do the following.
   (a) Merge X with the jth cluster from $L(C, \Pi')$.
   (b) Plot the data point $(j-1, 1-({}^{m(C,X)}/_{|C|})$ in the 'linked' graph.
   (c) If $|C| = n$, set $y = 0$. Otherwise set $y = {}^{i(C,X)}/_{n-|C|}$.
   (d) If $b = 0$ (resp. $b = 1$), plot the data point $(j-1, y)$ (resp. $(j-1, 1-y)$) in the 'mixed' graph.

Fig. 1. Plotting the 'linked' and 'mixed' graphs for a given solution cluster

Based on these plots, we can measure how 'dangerously' the adversarial partition approaches any given threshold $\beta \in [0, 1]$, where β represents a percentage of to-be-linked elements of a given solution cluster C. This is done as follows. First, we let $y_{\lambda,C,\Pi'}(x)$ (resp. $y_{\mu,C,\Pi'}(x)$) denote the y-coordinate of the data point at x in that cluster's linked (resp. mixed) graph. If $\beta \leq y_{\lambda,C,\Pi'}(0)$, then the threshold β has already been surpassed by the partition Π'; no mergings are required to reach β. If $\beta > y_{\lambda,C,\Pi'}(0)$, on the other hand, then we proceed as follows. First, we draw a horizontal line starting at the point $(0, \beta)$. Using this line, we find the point $(x_{\lambda,C,\beta}, y_{\lambda,C,\beta})$ of the linked graph that corresponds to β. From there, we draw a vertical line and find the point $(x_{\mu,C,\beta}, y_{\mu,C,\beta})$ where this line meets the mixed graph. We call the vectors that start at the origin and point to $(x_{\lambda,C,\beta}, y_{\lambda,C,\beta})$ and $(x_{\mu,C,\beta}, y_{\mu,C,\beta})$, the '$\beta$-linked' and the '$\beta$-mixed' vectors, respectively. The *slope* of the β-linked vector, expressed as a percentage, is then used as a measure of how quickly the adversarial partition approaches the threshold.

Combining and normalising secondary errors. Unless $\alpha = 0$, the slope of the β-linked vector must be co-evaluated with the slope of the β-mixed vector, because the latter expresses how quickly foreign elements 'contaminate' the adversarial cluster as it approaches the given threshold. We again use the policy parameter α in order to combine the two slopes into a single measurement: the risk slope of a given solution cluster C with respect to an adversarial partition Π', a threshold $\beta > y_{\lambda,C,\Pi'}(0)$, and policy parameter α is defined as $\Delta(C, \Pi', \beta, \alpha) = {}^2/_\pi[\alpha \arctan(y_{\lambda,C,\beta}/x_{\lambda,C,\beta}) + (1-\alpha)\arctan(y_{\mu,C,\beta}/x_{\mu,C,\beta})]$. Note that $0 \leq \Delta(C, \Pi', \beta, \alpha) \leq 1$.

Evaluating the entire partition. The above computations and graphs measure the extent to which a given solution cluster is hidden within an adversarial partition. We would like to plot a single graph that summarises the situation

for the entire solution partition and that somehow conveys the extent to which 'the typical' solution cluster is hidden in the solution partition. Unfortunately, taking the straight-forward average $|\Pi|^{-1} \sum_{C \in \Pi} \Delta(C, \Pi', \beta, \alpha)$ is not an option, because the values of $y_{\lambda,C,\Pi'}(0)$ are likely to differ for each C and this forces this expression to remain undefined for all $\beta \leq \max_{C \in \Pi} y_{\lambda,C,\Pi'}(0)$.

We circumvent this problem by plotting the 'overall' linked and mixed graphs using the **PlotForPartition** algorithm shown in Fig. 2. Note that, in order to make the algorithm work, the quantities $y_{\lambda,C,\Pi'}(x)$ and $y_{\mu,C,\Pi'}(x)$ for values of x between $|L(C, \Pi')|$ and $|\Pi| - 1$ (inclusive) must first be defined; recall that, since both graphs in the plot for cluster C have exactly $|L(C, \Pi')|$ data points (one representing the adversary's current state, and one for every merging until the adversary has linked all elements in C), these quantities were defined above only for $x < |L(C, \Pi')|$. We use the following recursive flat definitions to define $y_{\lambda,C,\Pi'}(x)$ and $y_{\mu,C,\Pi'}(x)$ for the missing range: for all $x \geq |L(C, \Pi')|$, $y_{\lambda,C,\Pi'}(x) \stackrel{\text{def}}{=} y_{\lambda,C,\Pi'}(|L(C, \Pi')| - 1) = 1$ and $y_{\mu,C,\Pi'}(x) \stackrel{\text{def}}{=} y_{\mu,C,\Pi'}(|L(C, \Pi')| - 1)$. If, for example, a plot does not contain a data point for the third merging because the adversary can fully link the elements of the cluster in, say, two mergings – $|L(C, \Pi')| = 3$ in this case – , then we take into account the situation after the final (i.e. second) merging. That is, the y-coordinates of all data points for three or more mergings, are defined to be identical to the y-coordinates of the data points at the second merging. For the linked graph, these coordinates are always 1 (since it is the *final* merging). For the mixed graph, they are equal to the percentage of foreign elements that were present after the final merging.

Given this definition, the linked (resp. mixed) graph produced by **PlotForPartition** represents the average percentage of elements, over all subjects, that are linked (resp. mixed) after the adversary is given *an allowance* of performing up to x cluster mergings. Armed with these graphs, the risk slope $\Delta(\Pi, \Pi', \beta, \alpha)$ representing the 'average solution cluster' for a given threshold β can be computed in the same manner as $\Delta(C, \Pi', \beta, \alpha)$. The worst-case complexity of our implementation of **PlotForPartition** is $O(n^3)$.

PlotForPartition (input: Π, Π', α)

1. Run **PlotForCluster** for all clusters $C \in \Pi$.
2. For values of $j = 0$ until $|\Pi| - 1$, do the following.
   (a) Compute the averages $\hat{y}_l = \frac{\sum_{C \in \Pi} y_{\lambda,C,\Pi'}(j)}{|\Pi|}$ and $\hat{y}_m = \frac{\sum_{C \in \Pi} y_{\mu,C,\Pi'}(j)}{|\Pi|}$.
   (b) Plot $(j, \hat{y}_l)$ and $(j, \hat{y}_m)$ in the 'linked' and 'mixed' graphs, respectively.

Fig. 2. Plotting the 'linked' and 'mixed' graphs for the entire partition

3.3 Sensitive Elements

Merely counting errors presumes that all elements are equal. In reality, however, often only little harm is done if an adversary links some elements, as long as certain particularly sensitive ones remain unlinked. Similarly, one may be

not suffer much if an adversary links some (or all) of one's elements, as long as certain foreign elements, perhaps of a particularly desirable type, are being mixed according to the adversary's view. In order to account for different element sensitivities, we enable the evaluator to first attach a weight $w_\ell \in [0,1]$ to each element ℓ. The weights are required to represent the relative sensitivity of the elements. We therefore generalise the miss and error count formulas as $m(C,C') = \sum_{\ell \in \{C-C'\}} w_\ell$ and $i(C,C') = \sum_{\ell \in \{C'-C\}} w_\ell$; the remainder of our framework remains unchanged.

Note that subjects may disagree as to which elements are more sensitive than others. They may also disagree on the value that the policy parameter α should have. This is not a problem in our framework; the evaluator may assign both different sensitivities to the elements and different values to α and well as $b$for each solution cluster evaluation. That is, each cluster plot may have a different underlying sensitivities; since step 2 of **PlotForPartition** does not take sensitivities into account, divergent sensitivities cause no problem. On the contrary, they will cause the overall plot to represent more accurately the summary of the risk as perceived by each subject.

3.4 Adversarial Views over Partitions

So far we have assumed that the adversary outputs a single partition. It may, however, output a *probability distribution* over the space of partitions. Note that a computationally bounded adversary can only output a distribution that can be encoded in polynomial length. Without loss of generality, we assume that an adversary outputs a view $\mathcal{V} = \{(\Pi'_1, \Pr(\Pi'_1 = \Pi)), (\Pi'_2, \Pr(\Pi'_2 = \Pi)), \dots\}$ such that , for all $1 \leq i \leq |\mathcal{V}|$, $\Pr(\Pi'_i = \Pi) > 0$ and $\sum_i \Pr(\Pi'_i = \Pi) = 1$. The pair $(\Pi'_i, \Pr(\Pi'_i = \Pi))$ means that, according to the adversary's view, Π'_i is the correct partition with probability $\Pr(\Pi'_i = \Pi)$.

Primary errors. We define the average miss and include error counts for solution cluster C with respect to a view $\mathcal{V}$ as

$$m(C,\mathcal{V},\alpha) = \sum_{((\Pi',\Pr(\Pi'=\Pi))\in\mathcal{V}} \Pr(\Pi' = \Pi) m(C,\Pi',\alpha) \text{ and}$$

$$i(C,\mathcal{V},\alpha) = \sum_{((\Pi',\Pr(\Pi'=\Pi))\in\mathcal{V}} \Pr(\Pi' = \Pi) i(C,\Pi',\alpha),$$

respectively.[3] These formulas then replace $m(C,\Pi,\alpha)$ and $m(C,\Pi,\alpha)$, and the remainder of the primary error evaluation as described in section 3.1 remains unchanged.

[3] These definitions follow the spirit of Equation 3 in [3], which also weighs a particular partition-dependent quantity by the probability that the underlying partition is the correct one.

Secondary errors. In order to plot the linked and mixed graphs for a given solution cluster with respect to an adversarial view $\mathcal{V}$, we use the **PlotForClusterGivenView** algorithm shown in Fig. 3. This algorithm is very similar in spirit with the **PlotForPartition** algorithm; the difference is that the plotted values are not averages over clusters, but rather weighted averages over the partitions in the view. Note that, as expected, both **PlotForCluster** and **PlotForClusterGivenView** effectively yield identical plots for adversaries that output a single partition. Also note that the worst-case complexity of **PlotForClusterGivenView** is $O(n^2|\mathcal{V}|)$. That is, the evaluation of sufficiently lengthy adversarial views, i.e. such that $|\mathcal{V}| \gg n^2$, is roughly linear in their size.

PlotForClusterGivenView (input: $C, \mathcal{V}, \alpha$)

1. Run **PlotForCluster**(C, Π', α) for all $\Pi' \in \mathcal{V}$.
2. For values of $j = 0$ until $|\Pi'| - 1$, do the following.
   (a) Compute the weighted averages $\hat{y}_\lambda = \sum_{\Pi' \in \mathcal{V}} \Pr(\Pi' = \Pi) y_{\lambda, C, \Pi'}(j)$ and $\hat{y}_\mu = \sum_{\Pi' \in \mathcal{V}} \Pr(\Pi' = \Pi) y_{\mu, C, \Pi'}(j)$
   (b) Plot $(j, \hat{y}_l)$ and $(j, \hat{y}_m)$ in the 'linked' and 'mixed' graphs, respectively.

Fig. 3. Plotting the 'linked' and 'mixed' graphs of given solution cluster with respect to an adversarial view over the partition space

4 Example

Consider a set of 24 items and the solution partition Π that divides it into four clusters of equal size: triangles, squares, circles, and stars. Suppose that two adversaries, which are asked to partition the set, come up with the adversarial partitions shown in Fig. 4 (first row). Assuming that our policy parameter is $\alpha = 1/2$, the left partition Π'_1 is a conservative one, because it exhibits $m(\Pi, \Pi'_1, \alpha) = m(\text{triangles}, \Pi'_1, \alpha) + m(\text{squares}, \Pi'_1, \alpha) + m(\text{circles}, \Pi'_1, \alpha) + m(\text{stars}, \Pi'_1, \alpha) = 4 + 2 + 3 + 3 = 12$ miss errors, but only $i(\Pi, \Pi'_1, \alpha) = 4$ include errors. The right partition Π'_2 is a liberal one: it exhibits $m(\Pi, \Pi'_2, \alpha) = 7$ miss errors and $i(\Pi, \Pi'_2, \alpha) = 15$ include errors.

The star-star-circle cluster of Π'_1, for example, does not contribute to the primary error counts at all. We therefore evaluate the partitions with respect to secondary errors. The second row of Fig. 4 shows the output of **PlotForCluster** for the star cluster, and with respect to the two adversarial partitions Π'_1 (left) and Π'_2 (right). The left (resp. right) side plot also shows the 'linked' and 'mixed' vectors corresponding to the linkage of $\beta = 90\%$ (resp. $\beta = 70\%$) of the stars. We have that $\Delta(\Pi, \Pi'_1, 0.9, 1/2) \approx 1/2(0.364 + 0.100) \approx 23.20\%$ and $\Delta(\Pi, \Pi'_2, 0.7, 1/2) \approx 1/2(0.361 + 0.340) \approx 35.03\%$.

The third row of Fig. 4 shows the output of **PlotForPartition** with respect to the two adversarial partitions, for different values of the parameter α. Note that, for most parameter values, the linked and mixed graphs coincide partially or entirely. Finally, the fourth row of Fig. 4 shows the the risk slopes, i.e. the value of $\Delta(\Pi, \Pi', \beta, \alpha)$ as a function of the threshold percentage β.

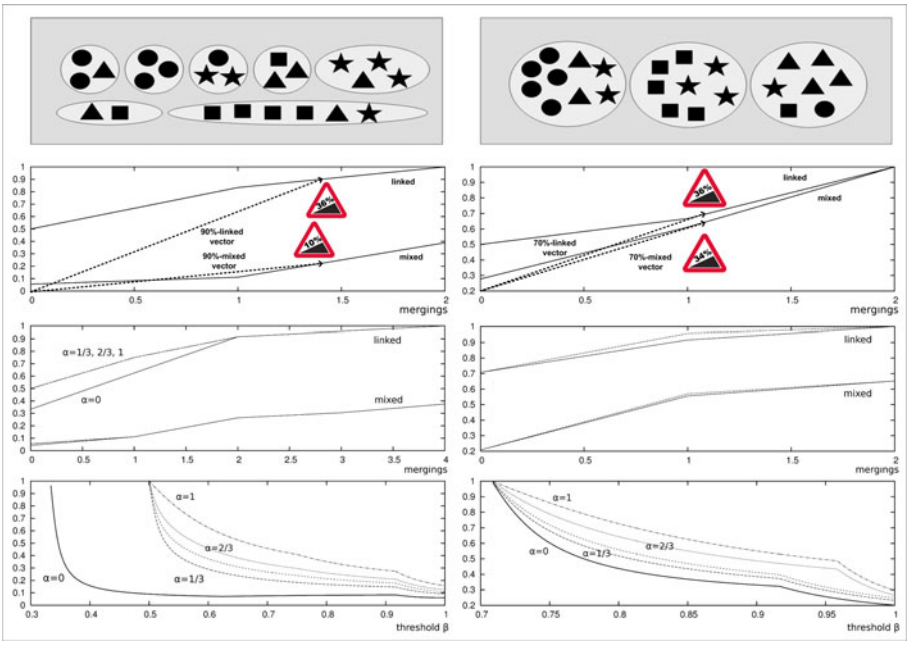


Fig. 4. First row: a conservative (left) and a liberal (right) adversarial partition. Second row: linked and mixed graphs for the star cluster and with respect to the adversarial partitions above ($\alpha = 1/2$). Third row: overall linked and mixed graphs, with respect to the adversarial partitions above, and for all $\alpha \in \{0, 1/3, 2/3, 1\}$. Fourth row: risk slopes $\Delta(\Pi, \Pi', \beta, \alpha)$ of the plots in the third row, as a function of the threshold β, for all $\alpha \in \{0, 1/3, 1/2, 2/3, 1\}$. Contamination is assumed to be undesirable (i.e. $b = 0$).

5 Simulated Attacks

This section examines the behaviour of our risk slope measure. Our motivation is to demonstrate that it behaves intuitively when viewed as a distance measure between partitions. We first take a brief look at its behaviour in the setting of *uniformly at random* chosen partitions. (For an efficient way to choose partitions in this way, see chapter 10 of [10].) Due to space constraints we only show results for the case where contamination is undesirable, i.e. where $b = 0$. Figure 5 shows and constrasts how three distance measures behave for uniformly at random chosen partitions as the number of elements n grows. The three measures shown are $\Delta(\Pi, \Pi', \beta, \alpha)$ (for $\alpha = 1/3, 2/3$ and $\beta = 0.7, 0.8$), the variation of information (VOI) [7], and the minimal transfer distance (MTD) [2].

We observe that, similarly to the MTD and VOI, $\Delta(\Pi, \Pi', \beta, \alpha)$ behaves smoothly as n grows. This is important because a weak dependence on n is preferable to a strong one (see section 5 of [7]). Moreover, while the MTD and the VOI measures only depend on Π and Π', $\Delta(\Pi, \Pi', \beta, \alpha)$ varies depending on the sensitivities specified by the evaluator. The figure demonstrates that

$\Delta(\Pi, \Pi', 0.7, \alpha) > \Delta(\Pi, \Pi', 0.8, \alpha)$. This matches our intuition that, since linking 70% of a subjet's elements is generally easier than linking 80%, the risk of this happening is higher, too. The figure also shows that $\Delta(\Pi, \Pi', \beta, 2/3) > \Delta(\Pi, \Pi', \beta, 1/3)$. This matches our intution that, since uniformly at random chosen partitions tend to be conservative (i.e. have many relatively small, rather than few very large clusters), attaching more importance to the presence of foreign elements results in lower risk levels. Finally, we observe that, as n grows, the MTD increases while the other measures decrease. The reasons for this lie as much with the measures themselves as with the nature of uniformly at random chosen partitions and the applied normalisations. Due to space constraints we do not analyse this further here. See appendix A for more information on the behaviour of our measures in the setting of random partitions.

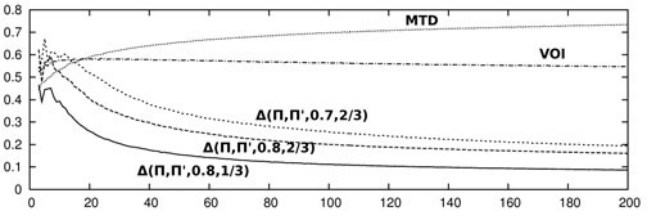


Fig. 5. $\Delta(\Pi, \Pi', \beta, \alpha)$, the variation of information normalised by $\log_2(n)$ (see section 5.1 of [7]), and the minimal transfer distance [2], normalised by $n - 1$. The shown results are averages of 1000 experiment repetitions.

In order to demonstrate that the behaviour of $\Delta(\Pi, \Pi', \beta, \alpha)$ is consistent and intuitive, we now compare it to the behaviour of the VOI distance in more detail. While we could use any reasonable measure as the basis for our comparison, we use the VOI because it has been shown to be a true metric [7]. We generate data for the comparison as follows. First, we choose a solution partition Π. Then we generate many partitions Π' that have different distances from Π. These partitions are generated by means of random walks of different lengths that start at Π and explore the solution space from there. Finally, we plot the $\Delta(\Pi, \Pi', \beta, \alpha)$ distance (vertical axis) against the VOI distance (horizontal axis) between Π and each Π'. Since the partitions Π' have different distances from Π, this partition generation method simulates attacks on unlinkability of different effectiveness.

The three plots in Figure 6 show how $\Delta(\Pi, \Pi', \beta, \alpha)$ behaves as the VOI distance between partitions over a set of $n = 200$ elements increases. They demonstrate that the risk slope decreases monotonically as the VOI distance between partitions increases. This matches our intuition that, as the distance between the true partition and the adversary's guess increases, the average risk of reaching a particular threshold, also decreases.

Figure 6(a) shows the effect of two selection methods for Π: the upper graph shows the simulation results when Π is chosen uniformly at random, and the

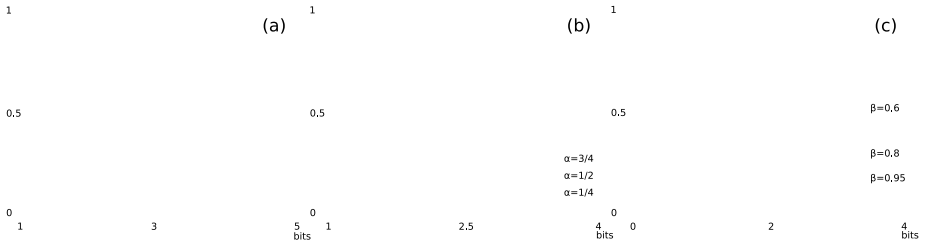


Fig. 6. The behaviour of $\Delta(\Pi, \Pi', \beta, \alpha)$ with respect to the VOI metric

lower graph shows the extreme case where Π is the singleton partition. For both graphs, parameters were set to $\alpha = 0.75$ and $\beta = 0.80$. Figure 6(b) demonstrates the influence of the policy parameter α. The graphs show the cases for $\alpha = 1/4, 1/2$, and $3/4$, respectively. The underlying partition Π' was chosen uniformly at random and the threshold parameter was fixed at $\beta = 0.8$. Note that, as α decreases, $\Delta(\Pi, \Pi', \beta, \alpha)$ decreases. As explained above, this is due to the conservative nature of uniformly at random chosen partitions. Finally, Figure 6(c) demonstrates the effect of β. The graphs show the cases for $\beta = 0.6, 0.8$, and 0.95, respectively. The underlying partition Π' was chosen uniformly at random and the policy parameter was fixed at $\alpha = 3/4$. The plot demonstrates the intuitive result that, as β increases, the risk slope decreases.

6 Concluding Remarks

We introduced a framework for the evaluation of adversarial partitions. The framework is flexible because it enables the evaluator to attach different levels of importance to the adversary's inability to link a given subject's elements, and its inability to distinguish the subject's elements from elements of other subjects. The evaluator may also specify, for each subject, different sensitivities for each element in the system as well as a threshold that represents what constitutes a privacy breach.

Our framework focuses on errors made by the adversary and distinguishes between primary and secondary errors. While primary errors measure how well a given subject's elements are currently linked and mixed with other subjects' elements, secondary errors project the risk that the adversary's current state represents for the future. Underlying this risk measurement is the implicit assumption that the adversary will obtain, in this future, information that enables a gradual merging of the clusters that contain some of that subject's elements, without at the same time filtering out foreign elements. In some settings, the adversary may be able to obtain information that leads to different ways of refining its current view. In such cases, secondary errors may turn out to be a less accurate representation of the real risk. We believe that, nevertheless, this does not invalidate the current approach because *any* refinement can be defined

as a set of cluster mergings and splittings: our linked graph accounts for the mergings, and our mixed graph partially accounts for the splittings.

Our evaluation framework does not, however, fully account for the number of splittings necessary to divide all elements into clusters. This is because, while the mixed graph represents the number of foreign elements, it does not take into account the exact number of subjects that correspond to these foreign elements. In some settings, for example [6], this number, as well as the requirement that each other subject should be represented by an approximately equal number of foreign elements, is important. Refining the evaluation framework in this respect while retaining its intuitive and flexible nature, is future work.

We envision our measures to be used for the evaluation of attacks on unlinkability in diverse settings including anonymous communication, online anonymous transactions, and identity management. We expect them to be useful because, by enabling the evaluator to play with the α and β parameters on the subject level, they offer the ability to evaluate attacks in more detail. Of course, in order to visualise and evaluate the adversary's state, this state must first be gathered. While this is typically not a problem in experimental settings, doing this without turning the data gatherer himself into the adversary remains a challenge in most real-world settings. Sometimes, however, for example in the setting of database sanitisation, the adversary's state is published. We expect our measures, just like other privacy measures, to be useful in pre-deployment analysis and in cases where reliable information about the adversary's state is available.

Acknowledgements

The authors are grateful to Filipe Beato and Markulf Kohlweiss for their insightful comments on an earlier version of this paper. This paper describes work undertaken partly in the context of the 'Trusted Architecture for Securely Shared Services' (TAS3) project (www.tas3.eu). TAS3 is a collaborative project supported by the 7th European Framework Programme, with contract number 216287.

References

1. Clauß, S.: A framework for quantification of linkability within a privacy-enhancing identity management system. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 191–205. Springer, Heidelberg (2006)
2. Denœud, L., Guénoche, A.: Comparison of distance indices between partitions. In: Batagelj, V., Bock, H.-H., Ferligoj, A., Ziberna, A. (eds.) Data Science and Classification, Mathematics and Statistics, pp. 21–28. Springer, Berlin (2006)
3. Fischer, L., Katzenbeisser, S., Eckert, C.: Measuring unlinkability revisited. In: Proceedings of the 2008 ACM Workshop on Privacy in the Electronic Society, WPES 2008, Alexandria, Virginia, USA, October 27, pp. 111–116. ACM Press, New York (2008)
4. Franz, M., Meyer, B., Pashalidis, A.: Attacking unlinkability: The importance of context. In: Borisov, N., Golle, P. (eds.) PET 2007. LNCS, vol. 4776, pp. 1–16. Springer, Heidelberg (2007)

5. Kogan, J.: Introduction to Clustering Large and High-Dimensional Data. Cambridge University Press, Cambridge (2007)
6. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data 1(1), 3 (2007)
7. Meilă, M.: Comparing clusterings—an information based distance. J. Multivar. Anal. 98(5), 873–895 (2007)
8. Meilă, M.: Comparing clusterings: an axiomatic view. In: ICML 2005: Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, pp. 577–584. ACM Press, New York (2005)
9. Neubauer, M.: Modelling of pseudonymity under probabilistic linkability attacks. In: IEEE International Conference on Computational Science and Engineering, vol. 3, pp. 160–167 (2009)
10. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms, 2nd edn. Academic Press Inc., London (1978)
11. Pashalidis, A.: Measuring the effectiveness and the fairness of relation hiding systems. In: Proceedings of the First International Workshop on Multimedia, Information Privacy and Intelligent Computing Systems (2008)
12. Rand, W.M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association 66(336), 846–850 (1971)
13. Schiffner, S., Clauß, S.: Using linkability information to attack mix-based anonymity services. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 94–107. Springer, Heidelberg (2009)
14. Steinbrecher, S., Köpsell, S.: Modelling unlinkability. In: Dingledine, R. (ed.) PET 2003. LNCS, vol. 2760, pp. 32–47. Springer, Heidelberg (2003)
15. Wagner, D., Wagner, S.: Comparing clusterings–an overview. Technical Report 2006-04, Faculty of Informatics, University of Karlsruhe, TH (2006)

A Random Partitions

This section experimentally examines how our measures behave for randomly chosen adversarial partitions. This examination provides reference points to attack evaluators which can be used to quantify by how much an adversary outperforms random guessing. We perform two experiments which we call the 'uniform' and the 'non-uniform' experiment, respectively. In both experiments, Π and Π' are randomly generated from the space of all set partitions of sets of size n. The experiments differ in the way the partitions are drawn from the space. In the uniform experiment, in particular, Π and Π' are chosen *uniformly* from the space of partitions. In the non-uniform experiment, Π and Π' are generated as follows. Initially, the to-be-generated partition consists of a 'cluster population' containing a single cluster that contains the first element. The remaining $n-1$ elements are then assigned to clusters, one by one, as follows. For each element, a fair coin is tossed. In case of heads, a new cluster is created, the element is assigned to that cluster, and the cluster is added to the cluster population; otherwise, a cluster already in the population is chosen uniformly at random and the element is assigned to it.

Fig. 7 shows some experiment results. As far as primary errors are concerned, in the uniform experiment occur, on average, slightly more errors than in the

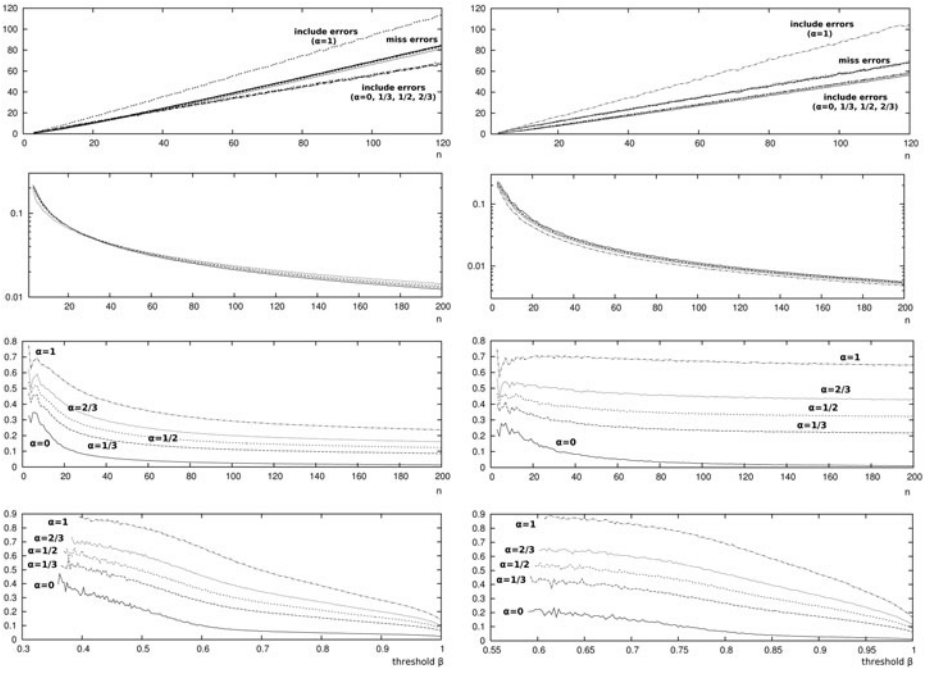


Fig. 7. First row: Average number of miss and include errors, i.e. $m(\Pi, \Pi', \alpha)$ and $i(\Pi, \Pi', \alpha)$, for varying n. Second row: normalised combined error counts $e^*(\Pi, \Pi', \alpha)$ on a logarithmic scale, for varying n. Third row: Risk slope $\Delta(\Pi, \Pi', 0.8, \alpha)$ for varying n. Fourth row: Risk slope $\Delta(\Pi, \Pi', \beta, \alpha)$, for constant $n = 50$ and varying threshold β. The results shown are for all parameter values $\alpha \in \{0, 1/3, 1/2, 2/3, 1\}$ and contamination is assumed to be undesirable ($b = 0$). Plots on the left and right hand side show results from the uniform and non-uniform experiment, respectively.

non-uniform experiment. Observe that, unless α takes very high values, in both experiments there occur less include than miss errors. For high values for α (say, above 90%), there occur more include rather than miss errors. This is because foreign elements are largely disregarded when determining the most relevant cluster and, as a result, that cluster has more foreign than correct elements. The average combined normalised error counts reflect the fact that there occur slightly fewer primary errors in the non-uniform than in the uniform experiment. Observe that, in our experiments, the choice of α has no significant impact on this measure.

The value of α has, on the other hand, a significant impact on the risk slopes $\Delta(\Pi, \Pi', \beta, \alpha)$; higher values of α yield higher risk slopes. Moreover, in the uniform experiment, the difference between the risk slopes for low and high values for α is much smaller than the corresponding difference in the non-uniform experiment. Finally, as the threshold β increases, the risk slopes converge. In the uniform experiment they converge slowly; in the non-uniform experiment they do not converge until $\beta \approx 0.7$, but then they converge fast.

Providing Mobile Users' Anonymity in Hybrid Networks*

Claudio A. Ardagna[1], Sushil Jajodia[2],
Pierangela Samarati[1], and Angelos Stavrou[2]

[1] DTI - Università degli Studi di Milano, 26013 Crema, Italia
{claudio.ardagna,pierangela.samarati}@unimi.it
[2] CSIS - George Mason University, Fairfax, VA 22030-4444, USA
{jajodia,astavrou}@gmu.edu

Abstract. We present a novel hybrid communication protocol that guarantees mobile users' k-anonymity against a wide-range of adversaries by exploiting the capability of handheld devices to connect to both WiFi and cellular networks. Unlike existing anonymity schemes, we consider all parties that can intercept communications between the mobile user and a server as potential privacy threats. We formally quantify the privacy exposure and the protection of our system in the presence of malicious neighboring peers, global WiFi eavesdroppers, and omniscient mobile network operators. We show how our system provides an automatic incentive for users to collaborate, since by forwarding packets for other peers users gain anonymity for their own traffic.

1 Introduction

We live in a globally interconnected society characterized by pervasive ubiquitous devices and communication technologies. The wide diffusion of the Internet, cellular networks, WiFi, low cost mobile devices, and the high availability of online services enable today's e-citizens to carry out tasks, access services, and stay connected virtually anywhere anytime. Unfortunately, the price we pay for this usability and convenience is an increased exposure of users' information and online activities. Organizations and individuals are slowly becoming aware of the privacy risks to which they are exposed. This scenario has sparked a renewed interest in the problem of providing privacy guarantees to users when operating in this brave new electronic world. Previous research has addressed different angles of the privacy problem. With respect to users' privacy, approaches like Mix-net [5], Onion Routing [8], and Crowds [19] were geared towards protecting the network anonymity of the users, preventing an adversary from linking

* This work was supported in part by the EU within the 7FP project "PrimeLife" under grant agreement 216483; by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4); by the National Science Foundation under grants CT-20013A, CT-0716567, CT-0716323, CT-0627493, and CCF-1037987; by the Air Force Office of Scientific Research under grants FA9550-07-1-0527, FA9550-09-1-0421, and FA9550-08-1-0157; and by the Army Research Office DURIP award W911NF-09-01-0352.

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 540–557, 2010.

the user to a service request. All these solutions were designed with traditional wired networks in mind, and shared the implicit assumptions on the stability of the routing configuration and network topology. In addition, many of them use the same path to route both user requests and responses. Existing solutions for wired networks are then not applicable in mobile networks where users can move fast in a short period of time and therefore cannot maintain a static communication path involving the same nodes between requests and responses. Approaches that have addressed the privacy problem in mobile ad-hoc networks (e.g., [14,17]) have been mostly aimed at providing anonymous routing protocols and have not considered protection of users' anonymity against the network operators; also, they typically rely on expensive multiparty computation and are therefore not suitable for mobile scenarios. On the other hand, privacy proposals for mobile networks (e.g., [7]) have addressed the problem of protecting users' anonymity against the services they access. These proposals, however, assume the mobile network operator to be in a privileged position and able to observe all the communications of the users.

In this paper, we study the privacy problem in hybrid networks where users, in addition to accessing online services via the cellular network, can communicate among each other over a WiFi network. Our goal is then to enable users to access online services using a cellular network in a privacy preserving way. To this end, we introduce a protocol that relies on the hybrid nature of mobile devices to create a local WiFi network which is impervious against global eavesdroppers that operate in the cellular network (e.g., mobile network operators). Our approach bases on the cooperation among peers in the WiFi network. An interesting aspect is that by collaborating in providing anonymity to others, peers gain themselves an immediate benefit on the anonymity of their communication. There is therefore an automatic incentive for peers to cooperate in the protocol.

Our approach represents an important paradigm shift, departing from the usual assumption of the mobile network operator as a trusted powerful entity able to know and observe all the traffic in the network. The mobile operator, while considered trustworthy with respect to the availability and working of the network, is restricted in terms of the view and traffic it can reconstruct. Addressing a novel threat and problem, our work is therefore complementary to existing solutions for privacy protection and could be applied in conjunction with them. Furthermore, we offer two important advantages over previous approaches. First, we do not rely on expensive communication or cryptographic operations including the use of multiparty computation, and we do not employ public key cryptography to convey the information to the server, beyond the connection establishment phase. Instead, we introduce a new fast packet filtering that leverages pseudo-random number generation to guarantee communication integrity. This aspect is particularly important to ensure applicability in a mobile environment, where low computation overhead and limited battery consumption are important requirements. Second, while guaranteeing privacy, we provide protection of the system against possible abuses of anonymity by maintaining the ability to block malicious traffic.

2 Problem Definition

Our reference model is a distributed and mobile infrastructure which forms a hybrid network, integrating *wireless*, *cellular*, and *wired* connections. The participating entities are: *1) mobile users*, that carry mobile devices supporting both GSM/3G and WiFi protocols for communication; *2) mobile network operators*, that manage radio cells of the cellular networks to provide wired network access to mobile users; and *3) servers*, that provide online services over the cellular network or the Internet. Mobile users can establish ad-hoc (WiFi) point-to-point connections with other mobile users in the network, resulting in several Mobile Ad-hoc NETworks (MANETs). Each mobile user, receiving signals from radio cells, is also registered with a given mobile network operator to access cellular functionalities. The cellular network acts as a gateway establishing a point-to-point connection with the user and the server. *Communication* is a bidirectional exchange of messages that involves a *user* u and a *server* s. Our goal is to provide a means for users to communicate with servers without giving the operator the ability to observe the communication profiles, that is, the pairs $\langle user,server \rangle$ describing service accesses. Protection is enacted by involving, in the communication with the mobile operator, other peers (users) with whom the user communicates via the WiFi network. Our approach guarantees that also participating peers will not be able to reconstruct the communication profile. We define the degree of anonymity protection enjoyed by a communication by modeling the uncertainty over the user and the server involved in it as follows.

Definition 1 ((k,h)-anonymity). *A communication is said to be (k,h)-anonymous against an adversary v, if v cannot relate the communication to less that k users and h servers.*

A communication is (k,h)-anonymous against an adversary, if the probability for the adversary of associating any u as the originating user is at most $\frac{1}{k}$ and the probability of associating any s as the server is at most $\frac{1}{h}$. A $*$ in place of a specific value for k (h, resp.) denotes that no inference can be drawn on the user (server, resp.) of a communication, which can therefore be any user (server, resp.) of the network. The degree of anonymity of a communication depends on the adversary. For each communication, user and server are known to each other, so their communications are $(1,1)$-anonymous to them. We assume the server of a communication to be always known to the mobile operator. With respect to a mobile operator, all communications will therefore be $(k,1)$-anonymous, where k defines the degree of k-anonymity [6,23] set by the user and provided by our protocol. Since the focus of our work is the protection of user's relations with servers against the mobile operator, our goal is to guarantee the k defined by the user. The reason for considering communication anonymity as a pair taking into consideration also the uncertainty on the server, is to model also the view of peers in the network (which do not know the servers to whom packets are being delivered). A communication between a user and a server is said to be completely exposed to an adversary if it is $(1,1)$-anonymous to her. It is considered protected if it is (k,h)-anonymous with $max(k,h) > 1$.

3 Rationale and Basics of Our Approach

The core idea of our approach is to empower users to anonymously involve other peers in sending a message to the server via a mobile operator using the WiFi network. Each message is split in k different packets and randomly distributed to k distinct peers in the WiFi network for their forwarding to the mobile network operator, that will then receive k indistinguishable packets from k different senders. Before introducing our communication protocol, we illustrate the basic knowledge that peers, operators, and servers participating in the network maintain or share.

Before any anonymous communication can be established, the user has to register and agree upon a secret key with the server. This pre-established secret key is used as a seed by the user to generate pseudo-random numbers to be associated with packets. All the servers, based on the seeds agreed with the users, jointly create a global table LEGITIMATE. LEGITIMATE consists of pairs (R^1, R^2) of pseudo-random numbers. This table can be either hosted by an external server accessible by the mobile network operators or alternatively stored by the operators themselves. Upon a packet arrival, the mobile network operator retrieves the pseudo random number attached to the packet and performs a lookup to the LEGITIMATE table to verify the validity of a packet. The cost of maintaining the LEGITIMATE table is manageable. For instance, assuming 128 pairs (R^1, R^2) of 64 bits of pseudo-random numbers to be used for packet verification and 1000 servers with 1 million users each, the storage requirements are approximately 1 TB which can be maintained by today off-the-shelf disks. The use of an external repository can then eliminate the need for a pre-storage of the random numbers since this repository can act as an intermediary between the individual servers and the mobile network operators. The size of R^1 and R^2 is chosen to be only 32 bits because each number is used only once and then discarded to avoid correlation attacks. The LEGITIMATE table acts as a blind firewall filter, allowing only packets tagged with an existing pseudo-random number (R^1) and having a valid encrypted message body to pass through.

Each peer p maintains the following tables: SENT_p contains the identifiers of the communications that the peer has helped distributing (including those originated by the peer) by forwarding a packet to the mobile operator; $\text{MYPRN}_{p,seed}$ contains the set of pseudo-random numbers $prn_i=(R_i^1, R_i^2)$ generated by p using the *seed* shared with the corresponding server. $\text{MYPRN}_{p,seed}$ contains the same *prn* generated by the server and is then a subset of the LEGITIMATE table. Each server s has a public/secret key pair $\langle P_s,S_s\rangle$. P_s is used by users when requesting connection establishment to encrypt the body of their message. This body includes a shared session key SK to be used by the user and the server for all further message exchanges in the session. Finally, each server s locally maintains a table ORIG_{sid} for each session *sid*, which stores the original set of peers (including user u) involved in the connection establishment.

To enforce integrity verification, we employ the UMAC [3] algorithm with R^2 as the key and the first 64 bits of the encrypted body of the message as a nonce for message authentication control. UMAC is designed to be very fast to compute

in software on contemporary uniprocessors with measured speeds as low as one cycle per byte [15] . In addition, the analysis of UMAC shows this scheme to have provable security, in the sense of modern cryptography, by way of tight reductions. Once a packet is forwarded to the server, the pseudo-random pair is removed from the table. Packets with invalid (i.e., *non-existing*) R^1 or invalid UMACs are discarded. The use of random numbers enables the protection of the servers against flooding attacks (mobile operators will discard packets that are found to be not genuine) preventing Denial-of-Service (DoS) attacks to servers.

4 Protocol

We present the working of the communication protocol distinguishing management of requests and responses. We use standard notation $E^s_K()$ and $D^s_K()$ to denote symmetric encryption and decryption operation with key K whereas $E^p_K()$ and $D^p_K()$ denote public key operations *used only for connection establishment.* Also, we will use $\mathcal{P}$, $\mathcal{O}$, and $\mathcal{S}$ to denote respectively the set of peers, mobile network operators, and servers in the hybrid network, and id_p and id_s as the identifiers of a peer and a server. Note that, to access a server, a user has first to establish a connection. In our protocol, connection requests and service access requests are indistinguishable to parties different from the initiating user and the server; all these parties (participating peers and mobile network operators) will simply observe packets without knowing whether they relate to a connection establishment or to a service access. The protocol and the behavior of the involved parties are the same for the two cases; the only differences are: *i)* in the set of peers selected, which contains user u, in the case of connection request; *ii)* within the content of the message, which contains the key for the session, in the case of connection request, and the id of the session, in the case of service request. Also, the body of the connection request packet is encrypted with the server's public key while the body of the service request packet is encrypted with the session key to which the request refers. Finally, for each service request, the response is also returned to peers in $\textsc{Orig}_{sid}$.

Figure 1 illustrates the protocol operations at the different participating parties. Figure 2 illustrates the distribution of packets among parties illustrating also how the content of the packets changes. Arcs with double lines refer to communications over the WiFi network (among peers), arcs with a single line refer to communications over the cellular network (between peers and mobile operators), arcs with a bold line refer to communications that can be carried on either over the wired or the cellular network (between the mobile operators and the servers), and arcs with a dotted line represent internal computations. Encrypted content is reported as a box with the encryption key appearing in the lower right corner of the box. Packets in Figure 2 refer to service access.

4.1 Request

For each session, a user can specify a privacy degree k to be guaranteed for all communications (connection and service requests related to the session) and

REQUEST ($u \rightarrow s$)
User $u \in \mathcal{P}$
u1.1 Let m be the message to be sent and *payload* its content.
Let k be the privacy preference, $(1 - P_f)$ the probability of forwarding to the operator, *cid* the communication id, UMAC_R a Universal Message Authentication Code (UMAC) using key R
u1.2 Generate a random message identifier *mid* and obtain the timestamp *tmp*
u1.3 Split *payload* in k parts $payload_i$, each with a sequence number seq_i, with i:=1...k
u1.4 **for** i:=1...k **do**
Generate $prn_i = (R_i^1, R_i^2)$ using the *seed*
$to_i := R_i^1$
if the message is a connection request
then generate session key *SK*
$body_i := E^p_{P_s}(id_u, seq_i, payload_i, SK, mid, tmp)$ /*connection establishment*/
else $body_i := E^s_{SK}(id_u, seq_i, payload_i, sid, mid, tmp)$ /*service access*/
u1.5 Wait until k peers are available
u1.6 **for** i:=2...k **do**
Choose a peer $p_i \in \mathcal{P}$
With random delay, send m_i:=[to_i,$body_i$,$\text{UMAC}_{R_i^2}\{body_i\}$,*cid*] to p_i in the WiFi network
u1.7 **if** (*cid* $\notin$ Sent_u)
then Sent_u:=$\text{Sent}_u \cup \{cid\}$
With random delay, forward [to_1,$body_1$,$\text{UMAC}_{R_1^2}\{body_1\}$] to o over the cellular network
else Send m_1:=[to_1,$body_1$,$\text{UMAC}_{R_1^2}\{body_1\}$,*cid*] to p_1 in the WiFi network
Peer $p \in \mathcal{P}$
Upon receiving a packet [*to*,*body*,$\text{UMAC}_{R^2}\{body\}$,*cid*]
p1.1 **if** (*cid* $\notin$ Sent_p)
then With probability $(1 - P_f)$: (Forward [*to*,*body*,$\text{UMAC}_{R^2}\{body\}$] to o over the cellular network; Sent_p:=$\text{Sent}_p \cup \{cid\}$; exit)
Send [*to*,*body*,$\text{UMAC}_{R^2}\{\text{body}\}$,*cid*] to a peer $p \in \mathcal{P}$
Operator $o \in \mathcal{O}$
Upon receiving [*to*,*body*,$\text{UMAC}_{R^2}\{body\}$] from peer p
o1.1 **if** ((*to* $\in$ Legitimate) and ($\text{UMAC}_{R^2}\{body\}$ is valid)
then Identify s using (*to*, R^2), remove (*to*, R^2) from Legitimate and forward [id_p,*to*,*body*] to s
else Drop the packet and exit
Server $s \in \mathcal{S}$
Upon receiving [id_p,*to*,*body*] from p via o
s1.1 Based on *to*, retrieve the content as $\text{D}^p_K(body) \vee \text{D}^s_K(body)$ with K:=$S_s \vee$ K:=*SK* respectively
s1.2 $\text{Orig}_{sid} = \text{Orig}_{sid} \cup \{[id_p, o]\}$ /*connection establishment*/
s1.3 Assemble the original message m with identifier *mid*

RESPONSE ($s \rightarrow u$)
Server $s \in \mathcal{S}$
Upon receiving all packets [id_p,*to*,*body*] for a request
s2.1 Let *payload* be the response, *sid* be the session id, and *SK* the session key
s2.2 **for** i:=1...k **do**
Let id_{p_i} and o_i be the peer id and the operator of the i-th packet of message *mid*
Generate the next random number from the seed prn_i=(R_i^1, R_i^2)
$body_i := E_{SK}(payload, sid, tmp)$
Send [id_{p_i},id_s,prn_i,$body_i$] to o_i
s2.3 **for each** $e_j \in \text{Orig}_{sid}$ with j=1...k /*service access*/
Generate the next random number from the seed prn_j=(R_j^1, R_j^2)
$body_j := E_{SK}(payload, sid, tmp)$
Send [$e_j.id_p$,id_s,prn_j,$body_j$] to $e_j.o$
Operator $o \in \mathcal{O}$
Upon receiving [id_p,id_s,*prn*,*body*] from s
o2.1 Remove *prn* from Legitimate and forward [id_s,*prn*,*body*] to p
User/Peer $p \in \mathcal{P}$
Upon receiving [id_s,*prn*,*body*]
up2.1 **if** ($prn \in \text{Myprn}_{p,seed}$)
then retrieve response as $\text{D}^s_{SK}(body)$
else drop the packet

Fig. 1. Communication protocol

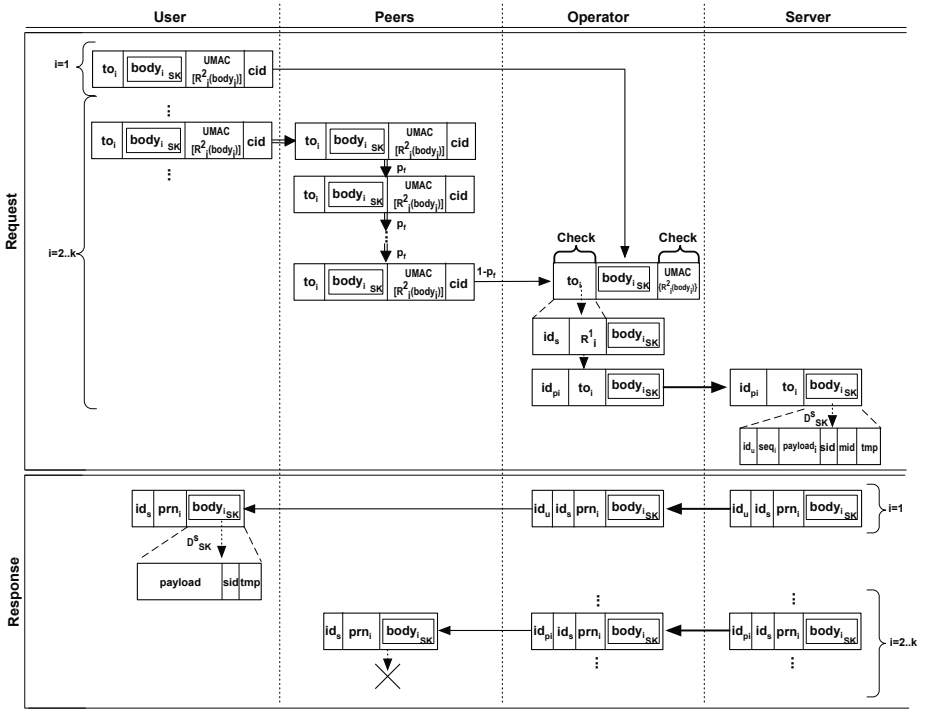


Fig. 2. Flow of packets within our protocol

the communication identifier *cid* to be used for all WiFi communications. The reason for *cid* is to limit to one the number of packets that a peer can send to the operator in each session (or in a window of time W like in Section 5).

User. Let m be a message with content *payload* to be sent by user u to server s. Let k be the privacy degree to be enforced, P_f and $(1 - P_f)$ the probability of forwarding to a peer in the communication range and to the operator, respectively, *cid* the communication identifier, and UMAC_R a Universal Message Authentication Code (UMAC) using key R. First, the user generates a random number *mid* that will be used as identifier for the message, and obtains the timestamp *tmp*. Then, the *payload* of the message is split into k different parts, $payload_1$,...,$payload_k$, each identified with its sequence number seq_i, to be sent via k different packets, composed as follows. For each packet m_i to be sent, to prove that the packet originates from a genuine user, the user generates, using the *seed* agreed with the server, a 64 bits pseudo-random number and splits it into two parts (i.e., $prn_i = (R_i^1, R_i^2)$). It then uses R_i^1 as the to_i field of packet m_i. The body $body_i$ of each packet to be sent, composed of the user id (id_u), sequence number of the packet (seq_i), packet payload ($payload_i$), message identifier *mid*, timestamp *tmp*, and either session key *SK* to be used for subsequent communication in the session (for connection requests), or session identifier *sid*

(for service access requests), is then encrypted. Encryption is performed with the server's public key P_s in case of connection requests and with the symmetric session key SK in case of service requests. In addition, a UMAC with R_i^2 as the key is used to produce the signature of the first 64 bits of the encrypted body, $\text{UMAC}_{R_i^2}\{\text{body}\}$, that is then appended at the end of the packet. Therefore, each packet m_i composed of $[to_i, body_i, \text{UMAC}_{R_i^2}\{body_i\}, cid]$, with $i := 1, \ldots, k$, is sent with a random delay to a different peer in the communication range. To avoid infinite loops in the distribution process, the user verifies through the WiFi channel if at least k peers (including u itself) are available in her proximity. If this is not the case, the user will not send the packet until enough peers become available. In the case of connection establishment (i.e., $cid \notin \text{SENT}_u$), the first packet m_1 is managed by u herself, that adds cid to SENT_u, keeping track of communications for which a packet has been forwarded to the operator; moreover, with a random delay, u forwards $m_1 = [to_1, body_1, \text{UMAC}_{R_1^2}\{body_1\}]$ to her operator o.

Peer. Upon receiving a packet $[to, body, \text{UMAC}_{R^2}\{body\}, cid]$, each peer p checks if it has already sent to the mobile operator any packet for the same communication (i.e., $cid \in \text{SENT}_p$). If it has not, the peer p sends the packet to its operator o with probability $(1 - P_f)$ and adds cid to SENT_p; while with probability P_f, it sends the packet unchanged to a peer in the communication range.

Operator. Upon receiving a packet $[to, body, \text{UMAC}_{R^2}\{\text{body}\}]$ from a peer p, the operator uses R^1 in the to field to retrieve the pair (R^1, R^2) in the global table LEGITIMATE, and checks the validity of $\text{UMAC}_{R^2}\{\text{body}\}$. If R^1 is a legitimate number (i.e., belongs to global table LEGITIMATE) and $\text{UMAC}_{R^2}\{\text{body}\}$ is a valid signature, the packet is genuine and the operator sends a message $[id_p, to, body]$ to server id_s. The remote server id_s is identified as the one that provided the pair associated with the packet. Also, the pair (R^1, R^2) is removed from the global table LEGITIMATE to ensure one-time use. If either R^1 is not in the table or the UMAC value of the body using R^2 is invalid, the packet is considered not genuine and dropped. Note that, the reason for including R^1 in each message to the servers, is to allow servers to quickly determine the key to be used in body decryption.

Server. Upon receiving a packet $[id_p, to, body]$ from operator o, using the field to, the server determines the encryption key K with which body was encrypted (server's public key P_s or session key SK), and decrypts $body$ accordingly (with server's private key S_s or session key SK, respectively). It then assembles the original message by merging the payloads in the bodies of the different packets. If the original message cannot be reconstructed, the communication is dropped and no response is returned to the user. In the case of connection establishment, for each received packet, the server adds $\{[id_p, o]\}$ to her local table ORIG_{sid}.

4.2 Response

Upon completion of the reception of all packets for the same message, the server determines the responses to be sent to different peers.

Server. Let *payload* be the response to be sent, *sid* be the session it refers to, and *SK* be the corresponding session key. For each packet related to message *mid*, received from peer p_i via operator o_i, the server generates $prn_i = (R_i^1, R_i^2)$ based on the *seed* shared with u. The body $body_i$ of the response is determined by encrypting, with session key *SK*: *payload* of the response, session identifier *sid*, and timestamp *tmp*. The server then sends $[id_{p_i}, id_s, prn_i, body_i]$ to operator o_i. Note that to make the body of responses referred to the same message different and indistinguishable from one another, the same body is encrypted i different times, by using a symmetric key encryption algorithm (e.g., 3DES, AES). In service access communication, a response $[e_j.id_p, id_s, prn_j, body_j]$ is also sent to each peer $e_j \in \text{ORIG}_{sid}$. As above, j different *prn* are used and j different *body* are generated by encrypting j individual times the plain message.

Operator. Upon receiving a response packet $[id_p, id_s, prn, body]$, the operator removes prn_i from table LEGITIMATE and forwards $[id_s, prn, body]$ to peer p.

User/Peer. Upon receiving a response packet $[id_s, prn, body]$ each peer p determines if *prn* belongs to one of her sets of pseudo random numbers ($prn \in \text{MYPRN}_{p,seed}$). If so, the peer was the initiating user u of the message to which the response refers, and can determine the decryption key thus retrieving *body* accordingly. Else, the peer drops the packet.

5 Assessing *k*-Anonymity

In our approach, a user establishing a connection needs to specify the number of peers whose cooperation it requests for achieving k-anonymity. In absence of previous history and in a non malicious environment, k-anonymity can be achieved by requesting cooperation of exactly k peers (as assumed in Section 4). However, the necessary number N of peers to involve to reach k-anonymity can decrease leveraging on previous communications in which the requester was involved, either as requester or participant on behalf of others, which introduce entropy. By contrast, it can increase in the presence of malicious peers and the consequent need to introduce redundancy in the system to provide resilience against them. In this section we discuss how a user can establish the number N of peers to involve in the protocol based on past communications and on a possible adversarial environment.

To prevent potential attacks from adversaries who try to subvert anonymity by using traffic analysis, we use a probabilistic path length and a multi-path approach. The expected path length L between a mobile user and the network operator (i.e., the number of hops taken by a packet in its path from a source to a destination) is randomly and exponentially distributed. In our multi-path configuration, a message originator or one of the peers forward each packet to a random next-hop peer with the same probability of forwarding P_f. Different packets of the same message follow different paths (that can be partially overlapped). The last peer on each path that has received a packet sends it to the network operator directly with the probability $(1 - P_f)$. Thus, like

in [19], we can derive the expected path length in non-malicious environment as: $L = (1 - P_f) \sum_{k=0}^{\infty} (k + 2) {P_f}^k = \frac{P_f}{(1-P_f)} + 2$.

Unfortunately, not all forwarded packets can be considered legitimate and not all neighboring peers are honest. To account for this, we define a threshold probability P_d of peers who misbehave. This probability includes peers moving out of the transmission range, dropping out of the network, acting maliciously by dropping or falsifying the packets they receive, or, in general, attempting to disrupt the normal operation of the system. Moreover, this probability threshold accounts for Sybil attacks [10] where a malicious peer can assume multiple false identities by pretending to have multiple WiFi physical occurrences. We assume that some fraction of peers in the WiFi network are malicious but the message originator is not. The expected path length in the presence of malicious peers that drop a packet can then be calculated as: $L = \frac{(1-P_d)P_f}{1-(1-P_d)P_f} + 2$.

In the remainder of this section, we analyze how the user can determine the number of peers to involve in the protocol to guarantee k-anonymity in case of communications involving a single request-response (Subsection 5.1) and multiple requests-responses (Subsection 5.2).

5.1 Single Request-Response

Each mobile user maintains the number M_s of packets forwarded for others to server s within a window of time W. In the protocol, to allow peers to calculate M_s, the server identifier id_s is declared in the response. The reason for declaring the server in the response, rather than in the request, is that the response traveling over the cellular network is not visible to WiFi eavesdroppers (see Section 6.1). While in Section 4 we assumed W to be equal to the session window, in the following, window W can be as large or as small as the mobile user prefers and is taken as a reference to evaluate the degree of anonymity. The value of W is not critical for single requests but becomes significant in case of consecutive requests to the same server. We envision that a typical value of W can range from a few seconds to several minutes. Assuming no malicious neighbors, the number of peers N, that user u needs to involve in a communication to achieve the required k-anonymity, is $N = k - M_s$, where M_s represents the number of packets the user has forwarded for other mobile peers to server s. In fact, if the user has participated in M_s previous communications, there must exist at least one peer that also participated in each of them.

Assuming the probability of malicious peers is at most P_d, to achieve the required k-anonymity for a request to server s in the window of time W, u must select at least N peers to satisfy the following formula:

$$k = \underbrace{\sum_{i}^{N} (1 - P_d)^L}_{N_f} + \underbrace{M_s \cdot (1 - P_d)}_{N_m} \tag{1}$$

Equation (1) has two contributing factors: $N_f = \sum_{i}^{N} (1 - P_d)^L$ indicates the expected number of successfully forwarded packets to the operator even in the

presence of a fraction of P_d malicious peers; $N_m = M_s \cdot (1 - P_d)$ accounts for the anonymity the user has gained by virtue of forwarding M_s packets for other mobile peers to server s. Of course, the mobile user will not be able to know the size UP of the set of unique peers that have communicated with the server but she can estimate that: $M_s \cdot (1 - P_d) \leq UP \leq \sum_{i=1}^{M_s} l_i \cdot (1 - P_d)$, where l_i is the number of peers that participate in the i-th communication. We consider the worst case scenario of having each l_i=1, and thus UP=$M_s \cdot$(1−P_d). Therefore, by forwarding packets for other mobile peers, a mobile user builds the necessary communication history that allows her to gain k-anonymity for her own traffic. In general, to determine the number N of necessary peers to involve in a communication, there are two extremes (see Equation 1): if u does not have any packet history within the window W, she needs to select a set of N peers that will successfully forward k packets to the server s, even if there is a fraction of P_d malicious peers. On the other hand, if u has already forwarded M_s packets to server s for other peers, if $N_m \geq k$, u can still enjoy k-anonymity without using any neighboring mobile peer, even assuming that P_d of them are malicious. A combination of the two extremes is also possible. In addition, based on the discussion in Section 4, a safe distribution process starts if and only if k peers (possibly including u) are available in users' proximity (i.e., path lenght L=1). Therefore, if we consider Equation 1 with probability of malicious users P_d, a user can safely start the communication if the number of available peers N satisfies $k = (N + M_s) \cdot (1 - P_d)$.

5.2 Multiple Requests-Responses

The analysis in Subsection 5.1 assumed that the communication between a user and server entails at most one message exchange. In practical applications, service access may require several messages between the involved user and the server. This opens the door to possible intersection attacks by which an observer can exploit the fact that a given user appears in different messages directed to a server. To counteract intersection attacks we ensure that both the requester as well as any other peer participate at most in the delivery of one message to the server in each given window W. The requester participates only in the first message exchange, but will receive all the responses since the server will send all responses to the original senders ORIG_{sid} (step s2.3). Also, peers participate in delivering a message only if they have not yet delivered any message for that communication (step p1.1). The important parameter is therefore the length of window W after which cid and ORIG_{sid} should be reset. Large sizes of W increase the potential level of anonymity but can decrease the ability of successfully concluding the communication. In fact peers that have participated in previous message delivery within a window become not usable anymore for forwarding packets to the operator and can then be modeled, with the formalization provided in Subsection 5.1, as malicious peers. The probability of packet dropping would then become $P_d = P_d + i\frac{k}{|\mathcal{P}|}$, where i is the number of request-response steps in the communication, $|\mathcal{P}|$ is the total number of peers in the network, and k is the preference of the requester. The probability P_d is then proportional to

the number of steps i; correspondingly, the probability of finding enough peers around u (i.e., N such that $k = (N + M_s) \cdot (1 - P_d)$ holds) decreases. By contrast, if W is small the probability of identifying the requester by means of intersection attacks increases. Each requester is in fact involved many times in a single communication and is more likely to be identified by an adversary. One limit case is when W is close to 0. In this case, the requester is involved in the packet forwarding of each request and thus she can be exposed with high probability.

6 Adversarial Analysis

Here, we present an analysis of our protocol against attacks by individual or colluding adversaries eavesdropping the communication as well as against timing and predecessor attacks. In addition, we point out the differences between wireless and wired networks and between full and k-anonymity.

6.1 Adversaries Eavesdropping the Communication

We assume that all participating parties in our system can play the role of adversary eavesdropping the communication and possibly collude.

Operator. A single o can only observe the communications involving peers that use o to forward their communications over the cellular network. Our system is designed to prevent o from identifying the originator u of a request below the k-anonymity threshold that the user selects. In fact, the originator u may not be subscribed to o, and then o is not able to observe the packets sent by u. Therefore, although o can relate the request to server s, it cannot deduct any information regarding u; hence, $(*, 1)$-anonymity is preserved.

Global WiFi eavesdropper. A Global WiFi eavesdropper can collect and analyze all WiFi traffic. Therefore, it can identify packets originating from mobile peers and potentially breach the requester's k-anonymity. However, a WiFi eavesdropper is not capable of identifying packets of the same message (i.e., with the same *mid*) in a short time interval. Moreover, it does not receive the responses from the server (which are communicated via the cellular network) and then it does not know the identity of server s. Therefore, a WiFi eavesdropper, short of breaking the cryptographic keys, cannot extract any information regarding o and s; hence $(1, *)$-anonymity is preserved.

But how easy is to create a WiFi eavesdropper? In WiFi communications peers establish point-to-point WiFi connections on ad-hoc channels. Hence, traditional WiFi providers are not able to simply use their access points to observe *all* WiFi communications. Rather, they need to employ ad-hoc antennas to cover *all* the area of interest and overhear on *all* point-to-point communications. Thus, the global WiFi eavesdropper scenario is possible in principle but difficult in practice. Another avenue of attack is to simulate a global WiFi eavesdropper employing "shadowing" neighbor peers that follow the victim. This attack is a special case of global WiFi eavesdropper. On their own, these nodes do not have

access to message content both in connection establishment and service access sub-protocols. In addition, every WiFi peer overhearing on the communications cannot assume that each packet forwarded by u, is originated by u himself, due to the hidden terminal problem that exists in all IEEE 802.11 communications [2]. This is a serious limitation and assumes that the WiFi nodes shadowing the victim will have to calculate and compensate for channel fading and signal loss due to physical objects. Finally, since packets need not to be manipulated by intermediate peers, there is no need to add identity or identifiable information to the packets in clear. Thus, the adversary is not able to infer who is the peer broadcasting a packet, unless there is a single peer in the communication range that is also physically visible by the adversary. $(k, *)$-anonymity is therefore preserved.

Colluding operators. This adversarial model results in an omniscient operator o that can observe all the traffic in the cellular network generated by mobile users using o to route their packets to the server. Our system does not attempt to protect the server's anonymity from such o and thus, o can observe all packets header information for a given time interval. Therefore, for each window of time W, o receives a set $M = \{m_{p,s,t}\}$ of packets from the cellular network where p denotes a mobile user, s is a server, and t is o's packet timestamp. Also, $\mathcal{P}$ denotes the set of mobile peers, $\mathcal{S}$ the set of servers, with $|\mathcal{S}| \leq |\mathcal{P}|$, and $K = \{k_1, \ldots, k_{|\mathcal{P}|}\}$ the set of peers' preferences. Operator o can place the observed packets in different sets, grouping packets having the same s, the same p, or the same pair (p, s). Given a server s, $M_{*,s}$=$\{m_{p',s',t'} \in M | s' = s\}$ is the set of all packets sent to the same server s, and $M_{p,s}$=$\{m_{p',s',t'} \in M | p' = p, s' = s\}$ is the set of packets sent from a peer p to a server s.

But, what can o extract from these sets that can be used for inference? There are two important metrics in each window of time W: *i)* the number of packets transmitted by unique peers to a server s, that is, $|\hat{M}_{*,s}|$, and *ii)* the maximum number of packet repetitions from a specific mobile peer p towards a specific server s. The first metric can be used to bound the maximum number of mobile peers that have *potentially* communicated with server s *assuming that o receives all the packets from all the mobile peers.* Let $max\{k_u\}$ be the greatest among all preferences of requesters u_i. If $max\{k_u\} \leq |\hat{M}_{*,s}|$, k-anonymity is preserved. This holds because if there are more than one requester communicating with the same server, then o will receive packets from all the peers involved in the communications. Our analysis in Section 5 (Equation 1) proves that $max\{k_u\}$ is the lower bound that we guarantee to all requesters. The second metric can be used to infer the lower bound on the number of communications that a server s received, that is, the maximum number of packet repetitions from a single peer. Although o may infer a lower bound to the number of communications, it will not be able to infer if a given user was the requester or a mere facilitator of the communication. Hence, $(k, 1)$-anonymity is preserved. In case that an omniscient operator employs a WiFi antenna, it could be able to observe both the cellular and WiFi channels in a given area, thus breaching the k-anonymity of the users in that area. However, the omniscient operator has to solve a much

more complex problem. This involves all challenges discussed in the Global WiFi eavesdropper scenario, including the hidden terminal problem and the difference in range between cellular and WiFi transmissions. To be successful, an adversary in the form of an omniscient operator has to employ many resources which make some of the attacks difficult to implement in practice: install WiFi antennas in strategic points for *all* areas of interest and utilize them solely for the purpose of eavedropping on *all* the available channels (each non-ovelapping channels requires yet another antenna). This constitues a significant investment of resources making the global WiFi eavesdropper a very expensive targeted attack with uncertain outcomes due to the user's mobility, the hidden terminal problem, and static or moving physical objects.

Colluding operators and WiFi eavesdroppers. This is the worst case scenario in which all parties are assumed to be malicious and colluding. In this case, we cannot provide any protection: all communications are monitored. Therefore information about both the cellular and the WiFi networks can be exposed. However, to be successful, this attack would require a malicious WiFi access point with enough range and capability of spectrum eaveasdropping or a fraction of malicious neighboring peers that shadow the user's every move. Although not infeasible, such sophisticated adversaries are highly unlikely to occur in practice for the large investements of resources they would require. Lastly, the higher the number of legitimate or non-cooperating neighboring peers, the more difficult it is for malicious peers to reach the required number of nodes to successfully breach k-anonymity.

6.2 Traditional Attacks

Our anonymity scheme can be further evaluated against attacks that have been primarily defined for wired networks. Two classes of such attacks are *timing attacks* [16] and *predecessor attacks* [24]. Timing attacks [16] focus on the analysis of the timing of network messages as they propagate through the system with the intent to link them back to the real user. This class of attacks has been successful in mix-based anonymity schemes for wired networks. They require the capability to manipulate the timing of packets and monitor its propagation on the victim's path. This usually requires at least one malicous node in the victim's path. In our scheme, there is no recurrent path due the definition of our protocol to the mobility of the users. Therefore, timing attacks are not effective against our approach. Indeed, the path and its length are generated probabilistically and change at each request. This makes practically infeasible for adversaries to setup a timing attack. Finally, the latency of each hop is intrinsically noisy: wireless communication performance can change due to weather conditions, interference by other devices, and physical obstacles. The predecessor attack [24] builds on the idea that by monitoring the communication for a given number of rounds (i.e., windows in this paper), a set of colluding attackers will receive messages with a higher rate from the real requesters. This is also based on the assumption that the real requesters communicate multiple times with the server and that are

part of anonymity groups (more or less stable). Our solution is not vulnerable to the predecessor attack since, by design, our protocol does not consider groups and assumes mobile users with ephemeral connectivity. A requester u that communicates on the mobile ad-hoc network moves fast and randomly during the communication. This makes it difficult for a set of adversaries to infer information about the requester by tracking her and intercepting her traffic. Moreover, the set of neighborhood peers around u may change at two consecutive time instants, and u may be involved in several other anonymous communications. Finally, to be successful in our settings, predecessor attacks must require the availability of a great fraction of corrupted peers, which follow the requester in her every move. This scenario is equivalent to the global WiFi eavesdropper discussed in Section 6.1. Note that, also in case of a static requester u, the sorrounding peers are not able to expose the identity of u, since the broadcasted packets do not contain identifiable information. Nevertheless, communication anonymity is preserved since peers do not know the server with whom u is communicating. If we change our view by considering a predecessor attack brought by an omniscient operator o, we can counteract this attack by tuning the length of the communication window. Contrary to Crowds [19], the "path reformation" (*i.e., the definition of a set of forwarding peers including* u) does not happen each time a peer joins or leaves the set of available peers but only at the end of the window. Moreover, while in Crowds the system is aware of peers joining or leaving, in our solution no involved parties (i.e., peers, mobile network operators) have knowledge of the length of the used window at any time. This leaves the adversary with guessing as the only option and increases dramatically the effort required to identify the window expiration time and protocol re-initialization.

7 Performance Evaluation

We have performed experiments to evaluate the performance of our protocol in terms of latency overhead imposed on the communication among parties. We measured the systems' performance using the Emulab (`http://www.emulab.net/`) and Orbit (`http://www.wirelessorbit.com/`) testbeds. In all of our experiments, we used devices equipped with standard IEEE 802.11 wireless network communication cards. All the results represent the average of multiple measurements (> 50) repeated over different periods of time to avoid wireless interference and transient effects from the wireless equipments. We varied the Signal-to-Noise Ratio (SNR) of the wireless link for single hop, peer-to-peer wireless connections, between 14 and 64, and we measured its impact on the link latency. As expected, our results show that the latency varies between 1ms and 52ms when the SNR is greater than 16. To characterize the behavior of a multi-hop wireless ad-hoc network, we employed the Random Waypoint [22] and the Orbit Mobility Framework [12] using city models for pedestrians. These models take into consideration mobility and interference which can degrade the signal quality. Then, we employed node mobility scenarios consisting of tens of nodes ($5-30$). For mobility scenarios, we varied the SNR between 24 and 64 using a timed event script, and we measured the impact of our anonymity protocol

on the end-to-end latency. The overhead trend is linear with the number of hops and ranges between 28.9ms for 2-hops and 127.9 for 6-hops in average. This overhead includes both the *communication and the computational costs*. The worst case scenario, in terms of overhead we observed, was for a 6-hop network. The maximum increase in latency overhead was approximately 150ms, which is acceptable for the majority of time-sensitive streaming applications. The latency impact when selecting a 3-hop or 4-hop network is relatively low (about 50ms and 70ms, respectively). The latency results indicate that our solution does not incur prohibitive overhead or packet losses.

8 Related Work

Past research addressing communication privacy in mobile networks [4,17,20] has been inspired by works in wired networks. Traditional solutions like TOR [8] for route anonymity and Crowds [19] for Web-communication anonymity usually assume a known network topology to create meaningful routes and use the path generated by the sender for both the request and the response. In addition, they often rely on trusted third parties (e.g., mix, onion router, blender) and on heavy multiparty computation. Other systems including I2P [13], MorphMix [21] take a different approach and provide P2P-based solutions for network anonymity. I2P [13] is an anonymizing network for secure communication that relies on tunnels and garlic routing to route data anonymously. I2P does not rely on centralized resources and does not use the same path for both the request and the response. MorphMix [21] is a P2P system for Internet-based anonymous communications, where each node is also a mix and can contribute to the anonymization process. Both I2P and MorphMix are based on heavy multiparty computation, consider wired networks, and are not able to manage mobility of the users. In general, all the above solutions are not applicable in a mobile scenario, where users move fast, form networks of arbitrary topology, and use devices with limited capabilities. Some solutions using mixes however have been designed for protecting privacy in mobile scenarios with constrained devices [11,18]. They focused on location management and protection, rather than on identity protection, and assume the existence of trusted parties.

Existing research in the context of mobile networks mainly focused on protecting privacy in mobile and vehicular ad-hoc networks [9,17,20,25], and mobile hybrid networks [1,4]. Dong et al. [9] propose an anonymous protocol for mobile ad-hoc networks that does not rely on topological information to protect identities and the locations of the nodes. Data packets are forwarded in real and fake routes to assure random route transmission and confuse adversaries, at a price of an increased communication overhead. GSIS [17] presents a protocol, based on Group Signature and Identity-based Signature techniques, used to protect security and privacy in vehicular networks. Capkun et al. [4] provide a scheme for secure and privacy-preserving communications in hybrid ad-hoc networks based on pseudonyms and cryptographic keys. Differently from the above works, our solution does not rely on multiparty computation, preserves the privacy of

the requester also from the mobile network operators, and provides an anonymous mechanism to verify the legitimacy of the traffic produced by mobile users. Ardagna et al. [1] present a multi-path approach for k-anonymity in mobile hybrid networks. The system in this paper considerably extends and improves the work presented in [1] by: *i)* removing public key encryption except for connection establishment; *ii)* extending the communication protocol to make it resistant to intersection attacks and suitable for multiple rounds of requests-responses, *iii)* allowing the requester to assess if there is enough entropy in the system to build communication anonymity, before start sending the message, and *iv)* providing a deep analysis and evaluation of the attacker model. Similarly to our approach, the work by Ren and Lou [20] is aimed at providing a privacy yet accountable security framework. This solution, however, is based on multiparty computation and groups of users established a priori, and assumes a semi-trusted group manager and network operator.

9 Conclusions

We proposed a protocol for protecting users' privacy that harness the availability of both mobile and WiFi connectivity in current phones creating a hybrid network. Differently from traditional solutions that offer privacy protection against servers and other peers only, we assumed mobile network operators as a potential source of privacy threats. The intuition behind our approach is that while users can trust the mobile operators to properly provide network accessibility, they want at the same time to be maintained free to act in the network without feeling their activities are constantly monitored. Therefore, our solution protects the privacy of the requester from all parties involved in a communication.

References

1. Ardagna, C., Jajodia, S., Samarati, P., Stavrou, A.: Privacy preservation over untrusted mobile networks. In: Bettini, Jajodia, S., Samarati, P., Wang, S. (eds.) Privacy in Location Based Applications. Springer, Heidelberg (2009)
2. Bianchi, G.: Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on selected areas in communications 18(3), 535–547 (2000)
3. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and secure message authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 216. Springer, Heidelberg (1999)
4. Capkun, S., Hubaux, J.-P., Jakobsson, M.: Secure and Privacy-Preserving Communication in Hybrid Ad Hoc Networks, Tech. Rep. IC/2004/10, EPFL-IC, Lausanne, Switzerland (January 2004)
5. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–88 (1981)
6. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Samarati, P.: k-Anonymity. In: Yu, T., Jajodia, S. (eds.) Secure Data Management in Decentralized Systems. Springer, Heidelberg (2007)

7. Cornelius, C., Kapadia, A., Kotz, D., Peebles, D., Shin, M., Triandopoulos, N.: Anonysense: privacy-aware people-centric sensing. In: Proc. of MobiSys 2008, Breckenridge, CO, USA (June 2008)
8. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proc. of the 13th USENIX Security Symposium, San Diego, CA, USA (August 2004)
9. Dong, Y., Chim, T., Li, V., Yiu, S., Hui, C.: ARMR: Anonymous routing protocol with multiple routes for communications in mobile ad hoc networks. Ad Hoc Networks 7(8), 1536–1550 (2009)
10. Douceur, J.: The sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 251. Springer, Heidelberg (2002)
11. Federrath, H., Jerichow, A., Pfitzmann, A.: Mixes in mobile communication systems: Location management with privacy. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, Springer, Heidelberg (1996)
12. Hong, X., Kwon, T., Gerla, M., Gu, D., Pei, G.: A mobility framework for ad hoc wireless networks. In: Tan, K.-L., Franklin, M.J., Lui, J.C.-S. (eds.) MDM 2001. LNCS, vol. 1987, pp. 185–196. Springer, Heidelberg (2000)
13. I2P Anonymous Network, `http://www.i2p2.de/`
14. Kong, J., Hong, X.: ANODR: Anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In: Proc. of MobiHoc 2003, Annapolis, MD, USA (June 2003)
15. Krovetz, T.: UMAC: Message authentication code using universal hashing. RFC 4418 (Informational) (March 2006)
16. Levine, B., Reiter, M., Wang, C., Wright, M.: Timing attacks in low-latency mix systems (extended abstract). In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 251–265. Springer, Heidelberg (2004)
17. Lin, X., Sun, X., Ho, P.-H., Shen, X.: GSIS: A secure and privacy preserving protocol for vehicular communications. IEEE Transaction on Vehicular Technology 56(6), 3442–3456 (2007)
18. Reed, M., Syverson, P., Goldschlag, D.: Protocols using anonymous connections: Mobile applications. In: Christianson, B., Lomas, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, Springer, Heidelberg (1998)
19. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM TISSEC 1(1), 66–92 (1998)
20. Ren, K., Lou, W.: A sophisticated privacy-enhanced yet accountable security framework for metropolitan wireless mesh networks. In: Proc. of ICDCS 2008, Beijing, China (June 2008)
21. Rennhard, M., Plattner, B.: Introducing MorphMix: peer-to-peer based anonymous internet usage with collusion detection. In: Proc. of WPES 2002, Washington, DC, USA (November 2002)
22. Saha, A., Johnson, D.: Modeling mobility for vehicular ad-hoc networks. In: Proc. of VANET 2004, Philadelphia, PA, USA (October 2004)
23. Samarati, P.: Protecting respondents' identities in microdata release. IEEE Transactions on Knowledge and Data Engineering 13(6), 1010–1027 (2001)
24. Wright, M., Adler, M., Levine, B.N., Shields, C.: The predecessor attack: An analysis of a threat to anonymous communications systems. ACM TISSEC 7(4), 489–522 (2004)
25. Zhang, Y., Liu, W., Lou, W., Fang, Y.: Mask: Anonymous on-demand routing in mobile ad hoc networks. IEEE Transaction on Wireless Communications 5(9), 2376–2385 (2006)

Complexity of Anonymity for Security Protocols

Ferucio Laurenţiu Ţiplea, Loredana Vamanu, and Cosmin Vârlan

Department of Computer Science
"Al.I.Cuza" University of Iaşi
Iaşi 700506, Romania
{fltiplea,loredana.vamanu,vcosmin}@info.uaic.ro

Abstract. Anonymity, as an instance of information hiding, is one of the security properties intensively studied nowadays due to its applications to various fields such as e-voting, e-commerce, e-mail, e -cash, and so on. In this paper we study the decidability and complexity status of the anonymity property in security protocols. We show that anonymity is undecidable for unrestricted security protocols, is NEXPTIME-complete for bounded security protocols, and it is NP-complete for 1-session bounded security protocols. In order to reach these objectives, an epistemic language and logic to reason about anonymity properties for security protocols under an active intruder, are provided. Agent states are endowed with facts derived from actions performed by agents in protocol executions, and an inference system is provided. To define anonymity, an observational equivalence is used, which is shown to be decidable in deterministic polynomial time.

1 Introduction

Anonymity, as an instance of *information hiding*, is one of the security properties intensively studied nowadays due to its applications to various fields such as electronic voting, electronic commerce, electronic mail, electronic cash and so on. It embraces many forms, such as *sender* or *receiver anonymity*, and it is closely related to *unlinkability*, *indistinguishability*, and *role interchangeability* [12, 10, 17]. The intuition behind anonymity is that an agent who performed some action is not "identifiable" by some observer of the system. "Non-identifiability" might mean that the observer is not able to see that the agent performed that action, or he saw that many other agents performed that action.

Several approaches to model anonymity have been proposed, such as [15,16,8, 7,5,9]. The approach in [15] is CSP-based, while the ones in [16,7] are based on epistemic logics. The authors in [16] show, in an epistemic logic framework, how the agent states can be augmented with information about actions performed by agents during protocol computations, and then propose an inference mechanism by which more information can be deduced. Several anonymity concepts are then proposed and discussed. The epistemic approach in [7] models anonymity in a multi-agent system framework. This is a very nice and general approach to talk about anonymity-related properties and many other papers on anonymity built

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 558–572, 2010.

on it [10,17]. Based on the concept of a *function view* as a concise representation of the intruder's partial knowledge about a function, Hughes and Shmatikov have proposed a rich variety of anonymity-related properties in [8]. The *cryptographic protocol logic* (CPL) in [9] came as an ambitious general framework for formalizing a very large class of security properties, including anonymity as well. While CPL seems very expressive, the model checking problem for it is undecidable and not too much about decidable fragments and proof systems for the core CPL is known.

From a computational point of view, the anonymity problem for security protocols is a decision problem: it is the problem to decide, given a security protocol and an action of it, whether or not the action is anonymous with respect to some agent. None of the papers mentioned above discusses the decidability and complexity status of this problem. As anonymity is not a *trace-based property* but it is based on an *observational equivalence* on protocol states, it is expected that anonymity is harder than secrecy or authentication. This is because, given a state of the protocol which is to be checked against some property, it might be the case that all observationally equivalent states are needed to be analyzed in order to decide the property.

In this paper we study the decidability and complexity status of the anonymity property for security protocols. Thus, we show that anonymity is undecidable for unrestricted security protocols, is NEXPTIME-complete for bounded security protocols, and it is NP-complete for 1-session bounded security protocols. In order to reach these objectives we enrich the security protocol model in [13, 19] by adding *facts* to agent states. Then we develop an inference system by which agents can infer more properties from facts. This inference system has special constructs, mainly due to the fact that in our approach the intruder is active, and this makes it different from the one in [16] (if the intruder is passive, then any receiver knows exactly from whom the message he received comes). To define anonymity, an *observational equivalence* is used, which is decidable in deterministic polynomial time.

The paper is organized in five sections. The formal model we use in this paper for security protocols is introduced in Section 2. Facts, as a way to cope with information about actions performed by agents in a security protocol, are introduced in Section 3, together with an inference system. Our observational equivalence is also a topic of this section, as well as the anonymity concepts we use in the paper. It is shown that the observational equivalence is decidable in deterministic polynomial time. Section 4 presents the main results of the paper. We conclude in Section 5.

2 Modeling Security Protocols

We recall the formalism in [13] with slight modifications [19], and use it in order to develop the main results of the paper.

Protocol signatures and terms. A *security protocol signature* is a 3-tuple $\mathcal{S} = (\mathcal{A}, \mathcal{K}, \mathcal{N})$ consisting of a finite set $\mathcal{A}$ of *agent names* (or shortly, *agents*)

and two at most countable sets $\mathcal{K}$ and $\mathcal{N}$ of *keys* and *nonces*, respectively. It is assumed that:

- $\mathcal{A}$ contains a special element denoted by I and called the *intruder*. All the other elements are called *honest agents* and Ho denotes their set;
- $\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1$, where $\mathcal{K}_0$ is the set of *short-term keys* and $\mathcal{K}_1$ is a finite set of *long-term keys*. The elements of $\mathcal{K}_1$ are of the form K_A^e (A's public key), or K_A^d (A's private key), or K_{AB} (shared key by A and B), where A and B are distinct agents;
- some honest agents A may be provided from the beginning with some *secret information* $Secret_A \subseteq \mathcal{K}_0 \cup \mathcal{N}$, not known to the intruder. $Secret_A$ does not contain long-term keys because they will never be communicated by agents during the runs;
- the intruder is provided from the beginning with a set of nonces $\mathcal{N}_I \subseteq \mathcal{N}$ and a set of short-term keys $\mathcal{K}_{0,I} \subseteq \mathcal{K}_0$. It is assumed that no elements in $\mathcal{N}_I \cup \mathcal{K}_{0,I}$ can be generated by honest agents.

The set of *basic terms* is $\mathcal{T}_0 = \mathcal{A} \cup \mathcal{K} \cup \mathcal{N}$. The set $\mathcal{T}$ of *terms* is defined inductively: every basic term is a term; if t_1 and t_2 are terms, then (t_1, t_2) is a term; if t is a term and K is a key, then $\{t\}_K$ is a term. We extend the construct (t_1, t_2) to $(t_1, \ldots, t_n)$ as usual by letting $(t_1, \ldots, t_n) = ((t_1, \ldots, t_{n-1}), t_n)$, for all $n \geq 3$. Sometimes, parenthesis will be omitted. Given a term t, $Sub(t)$ is the set of all *subterms* of t (defined as usual). This notation is extended to sets of terms by union.

The length of a term is defined as usual, by taking into consideration that pairing and encryption are operations. Thus, $|t| = 1$ for any $t \in \mathcal{T}_0$, $|(t_1, t_2)| = |t_1| + |t_2| + 1$, for any terms t_1 and t_2, and $|\{t\}_K| = |t| + 2$, for any term t and key K.

The *perfect encryption assumption* we adopt [1] states that a message encrypted with a key K can be decrypted only by an agent who knows the corresponding inverse of K (denoted K^{-1}), and the only way to compute $\{t\}_K$ is by encrypting t with K.

Actions. There are two types of actions, send and receive. A *send action* is of the form $A!B : (M)t$, and a *receive action* is of the form $A?B : t$. In both cases, A is assumed an honest agent who *performs the action*, $A \neq B$, $t \in \mathcal{T}$ is the *term of the action*, and $M \subseteq Sub(t) \cap (\mathcal{N} \cup \mathcal{K}_0)$ is the *set of new terms of the action*.

$M(a)$ denotes M, if $a = A!B : (M)t$, and the empty set, if $a = A?B : t$; $t(a)$ stands for the term of a. When $M = \emptyset$ we will simply write $A!B : t$. For a sequence of actions $w = a_1 \cdots a_l$ and an agent A, define the *restriction of w to A*, denoted $w|_A$, as being the sequence obtained from w by removing all actions not performed by A. The notations $M(a)$ and $t(a)$ are extended to sequences of actions by union.

Protocols. A *security protocol* (or simply, *protocol*) is a triple $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$, where $\mathcal{S}$ is a security protocol signature, $\mathcal{C}$ is a subset of $\mathcal{T}_0$, called the set of *constants* of $\mathcal{P}$, and w is a non-empty sequence of actions, called the *body* of the

protocol, such that no action in w contains the intruder. Constants are publicly known elements in the protocol that cannot be re-instantiated (as it will be explained below). As usual, $\mathcal{C}$ does not include private keys, elements in $Secret_A$ for any honest agent A, or elements in $\mathcal{N}_I$, $\mathcal{K}_{0,I}$ and $M(w)$.

Any non-empty sequence $w|_A$, where A is an agent, is called a *role* of the protocol. A role specifies the actions a participant should perform in a protocol, and the order of these actions.

Substitutions and events. Instantiations of a protocol are given by *substitutions*, which are functions σ that map agents to agents, nonces to arbitrary terms, short-term keys to short-term keys, and long-term keys to long-term keys. Moreover, for long-term keys, σ should satisfy $\sigma(K_A^e) = K_{\sigma(A)}^e$, $\sigma(K_A^d) = K_{\sigma(A)}^d$, and $\sigma(K_{AB}) = K_{\sigma(A)\sigma(B)}$, for any distinct agents A and B.

Substitutions are homomorphically extended to terms, actions, and sequences of actions. A substitution σ is called *suitable for an action* $a = AxB : y$ if $\sigma(A)$ is an honest agent, $\sigma(A) \neq \sigma(B)$, and σ maps distinct nonces from $M(a)$ into distinct nonces, distinct keys into distinct keys, and it has disjoint ranges for $M(a)$ and $Sub(t(a)) - M(a)$. σ is *suitable for a sequence of actions* if it is suitable for each action in the sequence, and σ is *suitable for a subset* $C \subseteq \mathcal{T}_0$ if it is the identity on C.

An *event* of a protocol $\mathcal{P} = (\mathcal{S}, \mathcal{C}, w)$ is any triple $e_i = (u, \sigma, i)$, where $u = a_1 \cdots a_l$ is a role of $\mathcal{P}$, σ is a substitution suitable for u and $\mathcal{C}$, and $1 \leq i \leq l$. $\sigma(a_i)$ is the *action of the event* e_i. As usual, $act(e_i)$ ($t(e_i)$, $M(e_i)$) stands for the the action of e_i (term of e_i, set of new terms of e_i). The *local precedence relation* on events is defined by $(u, \sigma, i) \rightarrow (u', \sigma', i')$ if and only if $u' = u$, $\sigma' = \sigma$, and $i' = i+1$, provided that $i < |u|$. $\stackrel{+}{\rightarrow}$ is the transitive closure of $\rightarrow$. Given an event e, ${}^\bullet e$ stands for the *set of all local predecessors of* e, i.e., ${}^\bullet e = \{e' | e' \stackrel{+}{\rightarrow} e\}$.

Message generation rules. Given X a set of terms, $analz(X)$ stands for the least set which includes X, contains t_1 and t_2 whenever it contains (t_1, t_2), and contains t whenever it contains $\{\{t\}_K\}_{K^{-1}}$ or $\{t\}_K$ and K^{-1}. By $synth(X)$ we denote the least set which includes X, contains (t_1, t_2), for any terms $t_1, t_2 \in synth(X)$, and contains $\{t\}_K$, for any term t and key K in $synth(X)$. Moreover, $\overline{X}$ stands for $synth(analz(X))$.

States and runs. A *state* of a protocol $\mathcal{P}$ is an indexed set $s = (s_A | A \in \mathcal{A})$, where $s_A \subseteq \mathcal{T}$, for any agent A. The *initial state* is $s_0 = (s_{0A} | A \in \mathcal{A})$, where $s_{0A} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_A \cup Secret_A$ for any $A \in Ho$, $s_{0I} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_I \cup \mathcal{N}_I \cup \mathcal{K}_{0,I}$, and $\mathcal{K}_X$ is the set of long-term keys known by $X \in \mathcal{A}$.

For two states s and s' and an action a, we write $s[a\rangle s'$ if and only if:

1. if a is of the form $A!B : (M)t$, then:
   (a) $t \in \overline{s_A \cup M}$ and $M \cap Sub(s) = \emptyset$; (*enabling condition*)
   (b) $s'_A = s_A \cup M \cup \{t\}$, $s'_I = s_I \cup \{t\}$, and $s'_C = s_C$ for any $C \in \mathcal{A} - \{A, I\}$;
2. if a is of the form $A?B : t$, then:
   (a) $t \in \overline{s_I}$; (*enabling condition*)
   (b) $s'_A = s_A \cup \{t\}$ and $s'_C = s_C$, for all $C \neq A$.

We extend the notation "$[\cdot\rangle$" to events by letting $s[e\rangle s'$ whenever $s[act(e)\rangle s'$, and we call $s[e\rangle s'$ a *computation step*. A *computation* or *run* of a security protocol is any sequence $s_0[e_1\rangle s_1 \cdots s_{k-1}[e_k\rangle s_k$ of computation steps, also written as $s_0[e_1 \cdots e_k\rangle s$ or even $e_1 \cdots e_k$, such that $s_{i-1}[e_i\rangle s_i$ for any $1 \leq i \leq k$, and ${}^\bullet e_i \subseteq \{e_1, \ldots, e_{i-1}\}$ for any $1 \leq i \leq k$ (for $i = 1$, ${}^\bullet e_i$ should be empty).

3 Anonymity-Related Security Properties

In this section we show how the model presented in the previous section can be endowed with information necessary to define and reason about anonymity properties in security protocols. The main idea is to add *facts* to agent states once the agents perform actions in the protocol. Each agent may then deduce new facts by using his knowledge at some point in the protocol. Although the idea of endowing agent states by facts was already used in [16], our approach is different. We endow the agent states with less information but sufficient to define and reason about a large spectrum of anonymity properties. While [16] assumes a passive intruder, in our approach the intruder is active. This asks for special deduction rules, making the deduction process more complex.

To define anonymity, a state-based observational equivalence is used in our paper. Two states are observationally equivalent w.r.t. an agent if the agent can derive the same "meaningful information" from each of the states. The anonymity concepts in [16] are not based on any observational equivalence. Halpern and O'Neill's approach to anonymity [7] is a very general one, so the observational equivalence is not precisely defined in their paper. Precise observational equivalences have been proposed, but for particular classes of anonymous communication [3]. The observational equivalence in [5] is trace-based. However, anonymity is not a *trace-based property* (or, at least, it cannot be naturally defined as a trace-based property such as secrecy or authentication).

3.1 Augmenting Agent States with Facts

When an agent in a security protocol performs a send or a receive action, he may record a number of important pieces of information. These pieces of information can be formalized by using *facts*[1], that is, sentences of the form $P(t_1, \ldots, t_i)$, where P is a predicate symbol of arity at least one and $t_1, \ldots, t_i$ are message terms (facts beginning by the same predicate symbol P will also be called P-*facts*).

In order to exemplify this we shall consider a running example. In the protocol in Figure 1, the agent A asks B for a ticket to access some network service H guarded by some agent C. Once A gets the authenticated ticket from B, it sends it to C together with an encrypted copy for H. C checks the ticket and then sends the encrypted copy to H.

Four classes of information pieces are to be recorded by agents in our formalism:

[1] Later in this section, facts will be considered *primitive propositions* for defining the epistemic logic we use to talk about anonymity properties.

$$
\begin{aligned}
A\,!\,B &: (\{N_A, K\})\ \{A, B, H, N_A, K\}_{K_B^e} \\
B\,?\,A &: \{A, B, H, N_A, K\}_{K_B^e} \\
B\,!\,A &: \{N_A, B, Ticket\}_K, \{N_A, B, Ticket\}_{K_B^d} \\
A\,?\,B &: \{N_A, B, Ticket\}_K, \{N_A, B, Ticket\}_{K_B^d} \\
A\,!\,C &: \{Ticket, \{Ticket\}_{K_{AH}}\}_{K_{AC}} \\
C\,?\,A &: \{Ticket, \{Ticket\}_{K_{AH}}\}_{K_{AC}} \\
C\,!\,H &: \{\{Ticket\}_{K_{AH}}\}_{K_{CH}} \\
H\,?\,C &: \{\{Ticket\}_{K_{AH}}\}_{K_{CH}}
\end{aligned}
$$

Fig. 1. A running example

1. *sent*-facts. Each agent X who sends a message t to some agent Y records a fact $sent(X, t, Y)$. For instance, when the first action of our example will be performed, A records $sent(A, \{A, B, H, N_A, K\}_{K_B^e}, B)$;
2. *rec*-facts. According to the intruder type, two cases are to be considered:
   (a) *passive intruder*. If an action $X\,?\,Y : t$ was performed by X, then X may safely record a fact $rec(X, t, Y)$ because he knows that the message he received is from Y. For instance, if action two in our running example was performed in some computation, then B may record the fact $rec(B, \{A, B, H, N_A, K\}_{K_B^e}, A)$;
   (b) *active intruder*. If an action $X\,?\,Y : t$ was performed by X, then X might not be sure whether t comes from Y or from the intruder. In such a case X records a fact $rec(X, t, (Y, I))$ showing that t may be from Y or from I. For instance, if action two in our running example was performed in some computation, then B records the fact $rec(B, \{A, B, H, N_A, K\}_{K_B^e}, (A, I))$;
3. *gen*-facts. The message in the first action of our running example is *generated by A for B* because it is encrypted by B's public key; denoted this by $gen(A, \{A, B, H, N_A, K\}_{K_B^e}, B)$ and record it in A's state. Similarly, $\{Ticket\}_{K_{AH}}$ in the fifth action is *generated by A for H* because it is encrypted by a key shared by A and H. Therefore, $gen(A, \{Ticket\}_{K_{AH}}, H)$ will be recorded in A's state;
4. *auth*-facts. In the third action of the protocol, the message sent by B to A contains a sub-message of the form $\{N_A, B, Ticket\}_{K_B^d}$. This is in fact B's digital signature on the message $(N_A, B, Ticket)$; we denote this by $auth(B, (N_A, B, Ticket, \{N_A, B, Ticket\}_{K_B^d}))$ and record it in B's state.

We will now formalize our discussion above. First, we extend the concept of an agent state from Section 2 as follows. A state of an agent A is a pair of sets $s_A = (s_{A,m}, s_{A,f})$, where $s_{A,m}$ is a set of messages and $s_{A,f}$ is a set of facts. Intuitively, $s_{A,m}$ represents the set of all messages the agent A sent or received in some computation ξ from the initial state to the state s_A, and $s_{A,f}$ represents the set of facts which give information about the actions the agent A performed in ξ.

Then, a protocol state is of the form $s = (s_A | A \in \mathcal{A})$, where each s_A has the form $s_A = (s_{A,m}, s_{A,f})$. We naturally extend the notation Sub for terms and

sets of terms to agent states by $Sub(s_A) = Sub(s_{A,m})$, and to protocol states by $Sub(s) = \bigcup_{A \in \mathcal{A} - \{I\}} Sub(s_A)$.

The protocol computation rule has to be changed accordingly. Given two states s and s' and an action a, we write $s[a\rangle s'$ if and only if:

1. if a is of the form $A!B : (M)t$, then:
   (a) $t \in \overline{s_{A,m} \cup M}$ and $M \cap Sub(s) = \emptyset$;
   (b) $s'_{A,m} = s_{A,m} \cup M \cup \{t\}$, $s'_{I,m} = s_{I,m} \cup \{t\}$, and $s'_{C,m} = s_{C,m}$ for any $C \in \mathcal{A} - \{A, I\}$;
   (c) the facts in s' are obtained as follows:
      i. add $sent(A, t, B)$ to $s_{A,f}$ and $s_{I,f}$;
      ii. if some term $t_1 = \{t'\}_{K_{AC}}$ or $t_1 = \{t'\}_{K^e_C}$ has been built by A in order to build t, then add $gen(A, t_1, C)$ to $s_{A,f}$;
      iii. if some term $t_1 = (t', \{t'\}_{K^d_A})$ has been built by A in order to build t, then add $auth(A, t_1)$ to $s_{A,f}$;
      iv. $s'_{C,f} = s_{C,f}$, for any $C \in \mathcal{A} - \{A, I\}$;
2. if a is of the form $A?B : t$, then:
   (a) $t \in \overline{s_{I,m}}$;
   (b) $s'_{A,m} = s_{A,m} \cup \{t\}$ and $s'_{C,m} = s_{C,m}$, for all $C \in \mathcal{A} - \{A\}$;
   (c) the facts in s' are obtained as follows:
      i. add $rec(A, t, (B, I))$ to $s_{A,f}$ and $s_{I,f}$;
      ii. $s'_{C,f} = s_{C,f}$, for any $C \in \mathcal{A} - \{A, I\}$.

In the case of a passive intruder (2a) should be "$t \in \overline{s_{B,m}}$" and (2ci) above should be "add $rec(A, t, B)$ to $s_{A,f}$ and $s_{I,f}$". All the other concepts, such as *computation step* or *computation* (*run*), remain unchanged.

3.2 Fact Derivation

At each point in the evolution of a protocol, each agent may derive new facts from the facts he owns at that point. For instance, when A performs the first action in our running example and sends $\{A, B, H, N_A, K\}_{K^e_B}$ to B, A records the fact $sent(A, \{A, B, H, N_A, K\}_{K^e_B}, B)$ in his state. As A built this message for B, he knows all the "ingredients" he used to built it and, therefore, A may think that he sent to B each such ingredient. Therefore, from $sent(A, \{A, B, H, N_A, K\}_{K^e_B}, B)$ the agent A should be able to derive $sent(A, N_A, B)$, or $sent(A, K, B)$, and so on. Even more, A should be able to derive facts like $sent(A, N_A)$ ("A sent N_A to some agent") or $sent(A)$ ("A sent some message") or $sent(N_A, B)$ ("N_A was sent to B") or $sent(A, B)$ ("A sent some message to B") or $sent(N_A)$ ("N_A was sent by some agent"). In order not to overload the notation we have used the same predicate symbol "*sent*" to denote these new facts; the distinction will always be clear from the context (alternatively, one may use the notation $sent(A, N_A, \_)$, $sent(A, \_, \_)$, and so on).

The derivation process sketched above is guided by deduction rules. Some of these rules are based on the *trace of a message with respect to an agent state*. Intuitively, the trace of t w.r.t. $s = (s_m, s_f)$, denoted $trace(t, s)$, is the set of all messages an agent in state s could use in order to build t.

Definition 1. A message t is called *decomposable* over an agent state $s = (s_m, s_f)$ if $t \in \mathcal{T}_0$, or $t = (t_1, t_2)$ for some messages t_1 and t_2, or $t = \{t'\}_K$ for some message t' and key K with $K^{-1} \in analz(s_m)$, or $gen(A, t, B) \in s_f$ for some honest agents A and B.

"$gen(A, t, B)$" in Definition 1 covers the case when A generates t for B by encrypting some message by B's public key (A does not know B's corresponding private key but knows how he built t and, from this point of view, we may say that t is decomposable).

Definition 2. The function $trace(t, s)$, where t is a message and $s = (s_m, s_f)$ is an agent state, is given by:

- $trace(t, s) = \{t\}$, if $t \in \mathcal{T}_0$;
- $trace(t, s) = \{t\} \cup trace(t_1, s) \cup trace(t_2, s)$, if $t = (t_1, t_2)$ for some terms t_1 and t_2;
- $trace(t, s) = \{t\}$, if t is not decomposable over s;
- $trace(t, s) = \{t\} \cup trace(t', s)$, if $t = \{t'\}_K$ is an encrypted but decomposable message over s.

We are now in a position to present our deduction rules:

- *fact simplification rules*

$$(S1)\ \frac{sent(A,t,B)}{sent(A,t),sent(A,B),sent(t,B)} \qquad (S2)\ \frac{sent(A,B)}{sent(A)}$$

$$(S3)\ \frac{sent(A,t)}{sent(A),sent(t)} \qquad (S4)\ \frac{sent(t,B)}{sent(t)}$$

$$(R1)\ \frac{rec(A,t,x)}{rec(A,t),rec(A,x),rec(t,x)} \qquad (R2)\ \frac{rec(A,x)}{rec(A)}$$

$$(R3)\ \frac{rec(A,t)}{rec(A),rec(t)} \qquad (R4)\ \frac{rec(t,x)}{rec(t)}$$

  where x is B or (B, I), and B is an honest agent different than A (if "A sent t to B" then we may also say that "A send t", or "A sent some message to B", or "the message t was sent to B", and so on);
- *message simplification rules*

$$(S5)\ \frac{sent(A,t,B),\ t' \in trace(t,s)}{sent(A,t',B)} \qquad (R5)\ \frac{rec(A,t,B),\ t' \in trace(t,s)}{rec(A,t',B)}$$

$$(R5')\ \frac{rec(A,t,(B,I)),\ t' \in trace(t,s)}{rec(A,t',(B,I))}$$

  where s is an agent state (if "A sent t to B" and t' was used by A to build t, then "A sent t' to B", and so on);
- *from rec-facts to gen- and auth-facts*

$$(RG)\ \frac{rec(B,\{t\}_{K_{AB}})}{gen(A,\{t\}_{K_{AB}},B)} \qquad (RA)\ \frac{rec(B,(t,\{t\}_{K_A^d}))}{auth(A,(t,\{t\}_{K_A^d}))}$$

  (if B received $\{t\}_{K_{AB}}$, then B knows that A is the only agent who could generate this message for him. If B verifies the signature on t and it turns out to be A's signature, then B knows that A authenticated the message t);

– *from rec-facts to sent-facts*

$$(RS1)\ \frac{rec(A,t,B)}{sent(B,t,A)} \qquad (RS1')\ \frac{rec(A,t,(B,I))}{sent(B)}$$

(if A knows that he received t from B, then B sent t to A; however, if A is not sure whether he received t from B, then what he knows is that B sent some message)

$$(RGS)\ \frac{rec(A,t),\ gen(C,t,A)}{sent(C,t,A)} \qquad (RAS)\ \frac{rec(A,t),\ auth(C,t)}{sent(C,t)}$$

(if A received some message t that was generated for him by C, then A can conclude that C sent the message to him. If A received an authentic message to C, then he can conclude that C sent the message);

– *from rec-facts to rec-facts*

$$(RGR)\ \frac{rec(A,t,(B,I)),\ gen(B,t,A)}{rec(A,t,B)} \qquad (RAR)\ \frac{rec(A,t,(B,I)),\ auth(B,t)}{rec(A,t,B)}$$

(if A is not sure whether he received the message t from B or from intruder, but the message t turns out to be generated by B for A or it is an authentic message to B, then A should be sure that the message t comes from B);

– *from sent-facts to sent-facts*

$$(SGS)\ \frac{sent(A,t),\ gen(A,t,B)}{sent(A,t,B)}$$

(if A sent t and generated it for B, then A sent t for B);

– *from sent-facts to rec-facts*

$$(SGR)\ \frac{sent(A,t,B),\ gen(C,t,B)}{rec(A,t,C)}$$

(if A sent t to B, and t was generated by C for B, then A received t from C).

As an example of deduction, one can easily derive from (SGR) and $(RS1)$ the following rule:

$$(RGS')\ \frac{rec(A,t,B),\ gen(C,t,A)}{sent(C,t,B)}$$

(RGS') captures a situation like the one in the Kerberos protocol (Figure 2) where C sends a ticket $\{t\}_{K_{AC}}$ to A via B. In this case, from the facts $rec(A,t,B)$ and $gen(C,t,A)$, the agent A is able to deduce $sent(C,t,A)$ (by using (RGS), $(S1)$, and (SGS)).

$$C \xrightarrow{\{\cdots,\{t\}_{K_{AC}}\}_{K_{BC}}} B \xrightarrow{\{\cdots\},\{t\}_{K_{AC}}} A$$

Fig. 2. Deduction rule (RGS')

The rule (RGS') can be used with our running example and allows H to deduce $sent(A,\{Ticket\}_{K_{AH}},C)$ at some state in the protocol (i.e., H will learn that A is the one who sent him the ticket $Ticket$).

Given a set M of messages and a set F of facts, denote by $Analz(M, F)$ the set of all facts that can be inferred from F and M. If $s = (s_m, s_f)$ is an agent state, then $Analz(s)$ stands for $Analz(s_m, s_f)$.

We note the difference between "*analz*" (Section 2) and "*Analz*".

3.3 Observational Equivalence

Anonymity, and other similar properties, are crucially based on what agents are able to "observe". If two distinct messages can be decomposed into the same atomic messages or both are encrypted by keys the agent A does not know, then the two messages are "observationally equivalent" from A's point of view in the sense that none of them reveals more "meaningful information" to A than the other. This can be extended to facts and agent states as follows.

Given a pair of agent states (s, s') define the binary relation $\sim_{s,s'}$ on message terms by:

- $t \sim_{s,s'} t$, for any $t \in \mathcal{T}_0$;
- $t \sim_{s,s'} t'$, for any term t undecomposable over s and any term t' undecomposable over s';
- $(t_1, t_2) \sim_{s,s'} (t'_1, t'_2)$, for any terms t_1, t_2, t'_1, and t'_2 with $t_1 \sim_{s,s'} t'_1$ and $t_2 \sim_{s,s'} t'_2$;
- $\{t\}_K \sim_{s,s'} \{t'\}_K$, for any terms t and t' and any key K with $t \sim_{s,s'} t'$ and $K^{-1} \in analz(s_m) \cap analz(s'_m)$.

Component-wise extend the relation $\sim_{s,s'}$ to facts:

$$P(t_1, \ldots, t_i) \sim_{s,s'} P(t'_1, \ldots, t'_i) \quad \Leftrightarrow \quad (\forall 1 \leq j \leq i)(t_j \sim_{s,s'} t'_j).$$

Definition 3. Two agent states $s = (s_m, s_f)$ and $s' = (s'_m, s'_f)$ are *observationally equivalent*, denoted $s \sim s'$, if the following hold:

- $analz(s_m) \cap \mathcal{T}_0 = analz(s'_m) \cap \mathcal{T}_0$;
- for any $\varphi \in Analz(s)$ there is $\varphi' \in Analz(s')$ such that $\varphi \sim_{s,s'} \varphi'$;
- for any $\varphi' \in Analz(s')$ there is $\varphi \in Analz(s)$ such that $\varphi' \sim_{s',s} \varphi$.

Roughly speaking, Definition 3 says that if $s = (s_m, s_f)$ and $s' = (s'_m, s'_f)$ are two observationally equivalent states of an agent, then the agent can derive the same meaningful information from any of these two states. Or, in other words, these two states are *indistinguishable*.

Let $s_m = \{\{N_C\}_K\}$, $s_f = \{rec(A, \{N_C\}_K, B)\}$, $s'_m = \{\{C, N_C\}_K\}$, and $s'_f = \{rec(A, \{C, N_C\}_K, B)\}$, where K is a symmetric key. According to Definition 3, $s = (s_m, s_f)$ and $s' = (s'_m, s'_f)$ are observationally equivalent. If we replace s_m above by $\{\{N_C\}_K, C, K\}$ and s'_m by $\{\{C, N_C\}_K, K\}$, then s and s' are not anymore observationally equivalent because from $rec(A, \{C, N_C\}_K, B)$ and s'_m one can infer $rec(A, C, B)$, and this fact cannot be inferred from $rec(A, \{N_C\}_K, B)$ and s_m.

Proposition 1. The observational equivalence on agent states is an equivalence relation decidable in $\mathcal{O}(f^4 l^4)$ time complexity, where f is the maximum number of facts in the states, and l is the maximum length of the messages in the states.

Recall that a protocol state is a tuple $s = (s_A | A \in \mathcal{A})$. We extend the equivalence relation defined above to protocol states on coordinates, that is, two protocol states s and s' are *observationally equivalent with respect to an agent* A, denoted $s \sim^A s'$ if $s_A \sim s'_A$. From Proposition 1 it follows that $\sim^A$ is an equivalence relation on protocol states, for any agent A.

3.4 Anonymity

We use the epistemic logic in [2,7] to reason about anonymity, tailored to our paper as follows:

$$\varphi ::= p \,|\, \varphi \wedge \varphi \,|\, \neg\varphi \,|\, \mathtt{K}_A\varphi$$

where A ranges over a non-empty finite set $\mathcal{A}$ of agent names and p ranges over a set Φ of *sent*-, *rec*-, *gen*-, and *auth*-facts such that no *rec*-fact contains terms of the form (B, I).

The anonymity concepts we will define make use of only one occurrence of the operator $\mathtt{K}$ in any formula and so, the *truth value of a formula* φ in a security protocol $\mathcal{P}$ is defined inductively as follows:

- $\mathcal{P} \models \varphi$ iff $(\mathcal{P}, s) \models \varphi$, for any reachable state s in $\mathcal{P}$;
- $(\mathcal{P}, s) \models p$ iff $(\mathcal{P}, s_A) \models p$, for some agent $A \neq I$;
- $(\mathcal{P}, s_X) \models p$ iff $p \in Analz(s_X)$, where $X \in \mathcal{A}$;
- $(\mathcal{P}, s) \models \neg\varphi$ iff $(\mathcal{P}, s) \not\models \varphi$;
- $(\mathcal{P}, s) \models \varphi \wedge \psi$ iff $(\mathcal{P}, s) \models \varphi$ and $(\mathcal{P}, s) \models \psi$;
- $(\mathcal{P}, s) \models \mathtt{K}_A\varphi$ iff $(\mathcal{P}, s'_A) \models \varphi$, for any reachable state s' with $s' \sim^A s$.

The formula $\mathtt{K}_A\varphi$ means "agent A knows φ". As usual, we use $\mathtt{P}_A\varphi$ as an abbreviation for $\neg\mathtt{K}_A\neg\varphi$. $\mathtt{P}_A\varphi$ means "agent A thinks that φ is possible". We shall simply write $s \models \varphi$ instead of $(\mathcal{P}, s) \models \varphi$, whenever the protocol $\mathcal{P}$ is understood from the context.

Anonymity for security protocols will be defined for actions performed by agents. By an *action* we will understand a *sent*-fact (these are also called *sent-actions*), or a *rec*-fact that does not contain terms of the form (B, I) (these are also called *rec-actions*). Therefore, the *sent*-actions are of the form $sent(A, t, B)$, $sent(A, t)$, $sent(A, B)$, $sent(A)$, $sent(t)$, or $sent(t, B)$, while the *rec*-actions are of the form $rec(A, t, B)$, $rec(A, t)$, $rec(A, B)$, $rec(A)$, $rec(t)$, or $rec(t, B)$. By *act* we will denote a generic action of the one of the forms above.

Now, following [7], define minimal anonymity for security protocols.

Definition 4. Let $\mathcal{P}$ be a security protocol and X an agent in $\mathcal{P}$ (X may be an honest agent H or the intruder I). An action *act* of $\mathcal{P}$ is *minimally anonymous w.r.t.* X if $\mathcal{P} \models act \Rightarrow \neg\mathtt{K}_X act$.

As we can see, we have defined anonymity not only with respect to an honest agent but also with respect to the intruder. This is motivated by the fact that the intruder is an observer of the entire protocol execution and, in spite of the fact that he records all send and receive actions, he might not be able to see precisely the action performed by some agent. For instance, the intruder may be able to see that A performed a send action but he might not be able to see that A sent some specific message. On the other side, honest agents may have more deduction power than the intruder, but might not observe all send and receive actions performed in the protocol. Therefore, from the anonymity point of view, honest agents and the intruder have incomparable powers. This makes the study of anonymity with respect to the intruder very appealing.

The action $sent(B, Ticket, A)$ in our running example is minimally anonymous w.r.t. C because, whenever this action is performed, C is not able to deduce it from his knowledge. On the other side, the action $sent(A, \{Ticket\}_{K_{AH}}, C)$ is not minimally anonymous w.r.t. H because H can learn it by the deduction rule (RGS'), but it is minimally anonymous w.r.t. I because I cannot learn it.

Remark 1. We want to emphasize that the anonymity of an action which contains messages, such as $sent(A, t)$, should not be confused with the secrecy of t. The minimal anonymity of $sent(A, t)$ w.r.t. H means that H was not able to observe at some point that the agent A performed the "action of sending the message t" (although H might knew t).

Remark 2. The anonymity of an action within a group of agents (anonymity set) as defined in ([7], Definition 3.4) can be expressed in our formalism as well, and the results in Section 4 obtained for minimal anonymity hold for this kind of anonymity too. However, the lack of space does not allow us to go into details.

4 Complexity of Anonymity

In this section we establish several complexity results for the anonymity problem in security protocols. First, we fix a few notations.

Each action has a *type* which is a tuple. For instance, $sent(A, t, B)$ has type (s, a, m, a), where s stands for *sent*, a for "agent", and m for "message". Similarly, $sent(t, B)$ has type (s, m, a), $rec(A, t)$ has type (r, a, m), where r stands for *rec*, and so on.

Each action type τ induces two decision problems w.r.t. anonymity:

1. the *minimal anonymity problem for actions of type τ w.r.t. an honest agent* (abbreviated $MAP(\tau)$), which is the problem to decide, given a security protocol $\mathcal{P}$, an action *act* of type τ, and an honest agent H, whether *act* is minimally anonymous w.r.t. H in $\mathcal{P}$;
2. the *minimal anonymity problem for actions of type τ w.r.t. the intruder* (abbreviated $MAPI(\tau)$), which is the problem to decide, given a security protocol $\mathcal{P}$ and an action *act* of type τ, whether *act* is minimally anonymous w.r.t. the intruder in $\mathcal{P}$.

Minimal anonymity w.r.t. honest agents in unrestricted security protocols is undecidable. This can be obtained by reducing the halting problem for counter machines to the complement of the minimal anonymity problem. The reduction follows, somehow, a classical line for simulating counter machines [14]. When the machine halts, some action in the security protocol simulating the machine will not be minimally anonymous w.r.t. some honest agent, and this happens only when the machine halts.

Theorem 1. $MAP(\tau)$ is undecidable in unrestricted security protocols, for any action type τ.

The undecidability result in Theorem 1 can be extended to minimal anonymity w.r.t. the intruder, but not for all action types.

Theorem 2. $MAPI(\tau)$ is undecidable in unrestricted security protocols, for any action type τ except for (r, a, a), (r, m, a), and (r, a, m, a).

If we focus on bounded security protocols then the anonymity is decidable. Recall that a bounded security protocol [19] is a security protocol whose message terms are built over some finite set of basic terms and whose length do not exceed some constant k. As a conclusion, the state space of a bounded security protocol is finite and so, we should be able to decide whether an action *act* is minimally anonymous w.r.t. some agent X (honest or the intruder). An obvious algorithm for checking whether an action *act* is minimally anonymous w.r.t. X would be the following:

for *any reachable state* s *with* $s \models act$ **do**
  if *there exists a reachable state* s' *with* $s' \sim^X s$ *and* $s'_X \not\models act$ **then**
    act is minimally anonymous w.r.t. X
  else *act* is not minimally anonymous w.r.t. X
end

This algorithm searches the state space twice: once for reachable states s with $s \models act$ and then, if such a state is found, for a state s' with $s' \sim^X s$ and $s'_X \not\models act$. As the number of events of a bounded security protocol is exponential w.r.t. the size of the protocol [19], this algorithm has a very high time complexity (w.r.t. the size of the protocol).

The complexity can be cut down if we restrict the minimal anonymity problem to basic-term actions. An action *act* of a security protocol is called a *basic-term action* if all terms in the action are basic terms. For instance, $sent(A, N_A, B)$, where N_A is a nonce, is a basic-term action, whereas $sent(A, \{N_A\}_K, B)$ is not. For basic-term actions the following property holds: if $s' \sim^X s$ then $s'_X \not\models act$ if and only if $s_X \not\models act$. Therefore, for basic-term actions, the above algorithm can be simplified by replacing the test in the if-statement by the simpler one "$s_X \not\models act$". Thus, we obtain the following result.

Theorem 3. $MAP(\tau)$ and $MAPI(\tau)$ are in $NEXPTIME$ for any τ if they are restricted to basic-term actions of type τ and bounded security protocols. Moreover, except for $MAPI(r, a, a)$, $MAPI(r, m, a)$, and $MAPI(r, a, m, a)$, all

the other minimal anonymity problems restricted as above are complete for $NEXPTIME$.

If we restrict more bounded security protocols by allowing only 1-session runs, then we obtain the following complexity results.

Theorem 4. $MAP(\tau)$ and $MAPI(\tau)$ are in NP for any τ if they are restricted to basic-term actions of type τ and 1-session bounded security protocols. Moreover, except for $MAPI(r, a, a)$, $MAPI(r, m, a)$, and $MAPI(r, a, m, a)$, all the other minimal anonymity problems restricted as above are complete for NP.

5 Conclusions

Using an epistemic logic framework, we have considered in this paper a large variety of anonymity-related concepts for security protocols: six variants of sender anonymity and six variants of receiver anonymity. All of them were formulated both w.r.t. an honest agent and w.r.t. the intruder, and are based on an observational equivalence on protocol states, which is decidable in deterministic polynomial time.

We have shown that the decision problems induced by them are undecidable in unrestricted security protocols under an active intruder. For bounded (1-session bounded) security protocols we have shown that some of these decision problems are complete for NEXPTIME (NP). The status of the others is left open.

We have obtained similar results to those in Section 4 for other types of anonymity, such as the one in ([7], Definition 3.4), but they could not have been included here due to the lack of space.

References

1. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 29, 198–208 (1983)
2. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press, Cambridge (2003)
3. Feigenbaum, J., Johnson, A., Syverson, P.: A model of onion routing with provable anonymity. In: Proceedings of the 11th International Conference on Financial Cryptography and 1st International Conference on Usable Security, Scarborough, Trinidad and Tobago, February 12-16 (2007)
4. Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter Machines and Counter Languages. Mathematical System Theory 2, 265–283 (1968)
5. Garcia, F.D., Hasuo, I., Pieters, W., van Rossum, P.: Provable Anonymity. In: Proceedings of the 3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code, FMSE 2005, Alexandria, USA (2005)
6. Greibach, S.A.: Remarks on Blind and Partially Blind One-way Multicounter Machines. Theoretical Computer Science 7, 311–324 (1978)
7. Halpern, J.Y., O'Neill, K.R.: Anonymity and Information Hiding in Multi-agent Systems. Journal of Computer Security 13(3), 483–514 (2005)

8. Hughes, D., Shmatikov, V.: Information Hiding, Anonymity and Privacy: A Modular Approach. Journal of Computer Security 12(1), 3–36 (2004)
9. Kramer, S.: Cryptographic Protocol Logic: Satisfaction for (Timed) Dolev-Yao Cryptography. The Journal of Logic and Algebraic Programming 77, 60–91 (2008)
10. Mano, K., Kawabe, Y., Sakurada, H., Tsukada, Y.: Role Interchangibility and Verification of Electronic Voting. In: The 2006 Symposium on Cryptography and Information Security, Hiroshima, Japan (2006)
11. Minsky, M.L.: "Recursvive" Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines. Annals of Mathematics 74(3) (1961)
12. Pfitzmann, A., Hansen, M.: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management – A Consolidated Proposal for Terminology. Technical Report, Technische Universität Dresden (2008)
13. Ramanujam, R., Suresh, S.P.: A Decidable Subclass of Unbounded Security Protocols. In: Proceedings of Workshop on Issues in the Theory of Security (WITS 2001), pp. 11–20 (2003)
14. Ramanujam, R., Suresh, S.P.: Undecidability of Secrecy for Security Protocols. Manuscript (2003) http://www.imsc.res.in/~jam/
15. Schneider, P., Sidiropoulos, A.: CSP and Anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)
16. Syverson, P.F., Stubblebine, S.G.: Group Principals and the Formalization of Anonymity. In: Wing, J.M., Woodcock, J.C.P., Davies, J. (eds.) FM 1999, vol. 1708, pp. 314–333. Springer, Heidelberg (1999)
17. Tsukada, Y., Mano, K., Sakurada, H., Kawabe, Y.: Anonymity, Privacy, Onymity, and Identity: A Modal Logic Approach. In: Proceedings of the 2009 IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT 2009), pp. 42–51 (2009)
18. Ţiplea, F.L., Bîrjoveanu, C.V., Enea, C.: Complexity of the Secrecy for Bounded Security Protocols. In: Proceedings of the NATO Advanced Research Workshop on Information Security in Wireless Networks, Suceava, România (2006)
19. Ţiplea, F.L., Bîrjoveanu, C.V., Enea, C., Boureanu, I.: Secrecy for Bounded Protocols with Freshness Check is NEXPTIME-complete. Journal of Computer Security 16(6), 689–712 (2008)

k-Zero Day Safety: Measuring the Security Risk of Networks against Unknown Attacks

Lingyu Wang[1], Sushil Jajodia[2], Anoop Singhal[3], and Steven Noel[2]

[1] Concordia Institute for Information Systems Engineering, Concordia University
wang@ciise.concordia.ca
[2] Center for Secure Information Systems, George Mason University
{jajodia,snoel}@gmu.edu
[3] Computer Security Division, National Institute of Standards and Technology
anoop.singhal@nist.gov

Abstract. The security risk of a network against unknown zero day attacks has been considered as something unmeasurable since software flaws are less predictable than hardware faults and the process of finding such flaws and developing exploits seems to be chaotic [10]. In this paper, we propose a novel security metric, *k-zero day safety*, based on the number of unknown zero day vulnerabilities. That is, the metric simply counts how many unknown vulnerabilities would be required for compromising a network asset, regardless of what vulnerabilities those might be. We formally define the metric based on an abstract model of networks and attacks. We then devise algorithms for computing the metric. Finally, we show the metric can quantify many existing practices in hardening a network.

1 Introduction

Today's critical infrastructures and enterprises increasingly rely on networked computer systems. Such systems must thus be secured against potential network intrusions. However, before we can improve the security of a network, we must be able to measure it, since *you cannot improve what you cannot measure*. A network security metric is desirable since it will allow for a direct measurement of how secure a network currently is, and how secure it would be after introducing new security mechanisms or configuration changes. Such a capability will make the effort of network hardening a science rather than an art.

Emerging efforts on network security metrics (Section 5 will review related work) typically assign numeric scores to vulnerabilities as their relative exploitability or likelihood. The assignment is usually based on known facts about each vulnerability (e.g., whether it requires an authenticated user account). However, such a methodology is no longer applicable when considering zero day vulnerabilities about which we have no prior knowledge or experience. In fact, a major criticism of existing efforts on security metrics is that unknown zero day vulnerabilities are unmeasurable [10]. First, the knowledge about a software system itself is not likely to help because unlike hardware faults, software flaws leading to vulnerabilities are known to be much less predictable. Second, modeling adversaries is not feasible either, because the process of finding flaws

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 573–587, 2010.

and developing exploits is believed to be chaotic. Third, existing metrics for known vulnerabilities are not helpful, because they measure the difficulty of *exploiting* a known vulnerability but not that of *finding* a zero day vulnerability.

The incapability of measuring unknown zero day vulnerabilities can potentially diminish the value of security mechanisms since an attacker can *simply step outside the implementation and do as he pleases* [10]. What is the value of a more secure configuration, if it is equally susceptible to zero day attacks? We thus fall into the agnosticism that security is not quantifiable until we can fix all security flaws (by then we certainly do not need any security metric, either).

We propose a novel security metric, *k-zero day safety*, to address this issue. Instead of attempting to measure *which* zero day vulnerability is more likely, our metric counts *how many* distinct zero day vulnerabilities are required to compromise a network asset[1]. A larger number will indicate a relatively more secure network, since the likelihood of having more unknown vulnerabilities all available at the same time, applicable to the same network, and exploitable by the same attacker, will be lower. Based on an abstract model of networks and attacks, we formally define the metric and prove it to satisfy the three algebraic properties of a metric function. We then design algorithms for computing the metric. Finally, we show the metric can quantify many existing practices in network hardening and discuss practical issues in instantiating the model.

The contribution of this work is twofold. First, to the best of our knowledge, this is the first effort capable of quantifying the security risk of a network against unknown zero day attacks. Second, we believe the metric would bring about new opportunities to the evaluation, hardening, and design of secure networks.

In the rest of this paper, we first build intuitions through a running example. We then present a model and define the metric in Section 2, design and analyze algorithms in Section 3, discuss network hardening and model instantiation in Section 4, review related work in Section 5, and finally conclude the paper in Section 6. Due to space limitations, the proof of theorems is given in [21].

1.1 Motivating Example

The left-hand side of Figure 1 shows a toy example where host 1 provides an HTTP service ($http$) and a secure shell service (ssh), and host 2 provides only ssh. The firewall allows traffic to and from host 1, but only connections originated from host 2. Assume the main security concern is over the root privilege on host 2. Clearly, if all the services are free of known vulnerabilities, a vulnerability scanner or attack graph will both lead to the same conclusion, that is, the network is secure (an attacker on host 0 can never obtain the root privilege on host 2), and no additional network hardening effort is necessary.

However, we shall reach a different conclusion by considering *how many distinct zero day attacks the network can resist*. The upper-right corner of Figure 1 shows three sequences of zero day attacks leading to $\langle root, 2\rangle$ (each pair denotes a condition and each triple inside oval denotes the exploitation of a zero day vulnerability): An attacker on host 0 can exploit a zero day vulnerability in either $http$ or ssh on host 1 to obtain

[1] In our model, an asset is a general concept that may encompass one or more aspects of security, such as confidentiality, integrity, and availability.

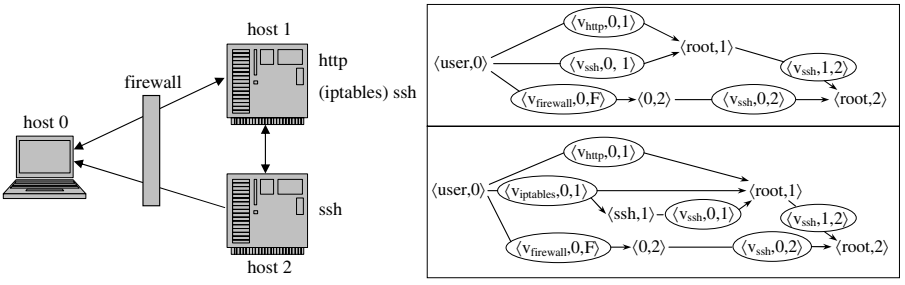


Fig. 1. Network Configuration and Sequences of Zero Day Attacks

the root privilege; using host 1 as a stepping stone, he/she can exploit a zero day vulnerability in ssh on host 2 to reach $\langle root, 2\rangle$; alternatively, he/she can exploit a zero day vulnerability in the firewall (e.g., a weak password in its Web-base remote administration interface) to re-establish the blocked connection to host 2 and then exploit ssh on host 2. The network can resist at most one zero day attack since the second sequence only requires one unique zero day vulnerability in ssh (on both host 1 and 2).

Now consider hardening the network by using iptables rules ($iptables$) to allow only specific hosts, not including host 0, to connect to ssh on host 1. The lower-right corner of Figure 1 shows four sequences of zero day attacks (the two new sequences indicate exploiting a zero day vulnerability in $iptables$ to either connect to ssh, or obtain the root privilege, on host 1). It can be observed that all four sequences now require two distinct zero day vulnerabilities. The seemingly unnecessary hardening effort thus allows the network to resist one more zero day attack. The hardened network can be considered relatively more secure, since the likelihood of having more zero day vulnerabilities available at the same time, in the same network, and exploitable by the same attacker, will be lower[2]. Therefore, the number of distinct zero day vulnerabilities can be used to measure the relative security risk of different networks, which may otherwise be indistinguishable by existing techniques. Those discussions, however, clearly oversimplify many issues, which will be addressed in the rest of this paper.

2 Modeling k-Zero Day Safety

In this section, we define the k-zero day safety metric based on an abstract model of network components. We shall delay to Section 4 the discussion of practical issues in instantiating the abstract model based on a real world network.

2.1 The Network Model

Definition 1 gives an abstract model of network components relevant to measuring zero day attacks (all notations will later be summarized in Table 1). The model will allow us to formally define and reason about the proposed metric.

[2] This likelihood would decrease exponentially in the number of vulnerabilities if such vulnerabilities can be modeled as i.i.d. random variables, but we shall not assume any specific model since the process of developing exploits is believed to be chaotic [10].

Definition 1 (Network). *Our network model has the following components:*

- H, S, *and* P, *which denotes the set of hosts (computers and networking devices), services, and privileges, respectively.*
- $serv(.) : H \rightarrow 2^S$ *and* $priv(.) : H \rightarrow 2^P$, *which denotes a function that maps each host to a set of services and that of privileges, respectively.*
- $conn \subseteq H \times H$, *and* $\preceq \subseteq priv(h) \times priv(h)$, *which denotes a connectivity relation and a privilege hierarchy relation, respectively.*

Here hosts are meant to also include networking devices because such devices are vulnerable to zero day attacks, and a compromised device may re-enable accesses to blocked services (e.g., the firewall in Figure 1). Also, tightly-coupled systems (e.g., a server hosting multiple replicas of a virtual host under the Byzantine-Fault Tolerance algorithm [3]) should be regarded as a single host, since we shall only consider causal relationships between hosts.

A service in the model is either remotely accessible over the network, in which case called a *remote service*, or used to disable a remote service or network connection, in which case called a *security service*. The model does not include services or applications that can only be exploited locally for a privilege escalation (modeling such applications may not be feasible at all considering that an attacker may install his/her own applications after obtaining accesses to a host). On the other hand, the model includes remote services and connectivity currently disabled by security services, since the former may be re-enabled through zero day attacks on the latter (e.g., ssh behind $iptables$ in Figure 1).

In the model, privileges are meant to include those under which services are running and those that can potentially be obtained through a privilege escalation. The purpose of including the latter is not to model privilege escalation itself but to model the strength of isolation techniques (e.g., sandboxing or virtual machines) that may prevent such an escalation, as we shall elaborate shortly.

Example 1. In Figure 1, we have

- $H = \{0, 1, 2, F\}$ (F denotes the firewall),
- $conn = \{\langle 0, F\rangle, \langle 0, 1\rangle, \langle 0, 2\rangle, \langle 1, F\rangle, \langle 1, 0\rangle, \langle 1, 2\rangle, \langle 2, F\rangle, \langle 2, 0\rangle, \langle 2, 1\rangle\}$ (we include $\langle 0, 2\rangle$ since it can be enabled by a zero day attack on the firewall),
- $serv(1) = \{http, ssh, iptables\}$, $serv(2) = \{ssh\}$, and $serv(F) = \{\ firewall\}$ ($firewall$ is a security service and it disables connection $\langle 0, 2\rangle$),
- $priv(1) = priv(2) = \{user, root\}$.

2.2 The Zero Day Attack Model

The very notion of *unknown* zero day vulnerability means we cannot assume any vulnerability-specific property, such as the likelihood or severity. We can, however, assume generic properties common to vulnerabilities, as in Definition 2.

Definition 2 (Zero Day Vulnerability). *A zero day vulnerability is a vulnerability whose details are unknown except that it satisfies the following*[3].

1. *It cannot be exploited unless*
   (a) *a network connection exists between the source and destination hosts,*
   (b) *a remote service with the vulnerability exists on the destination host,*
   (c) *and the attacker already has a privilege on the source host.*
2. *Its exploitation can potentially yield any privilege on the destination host.*

The assumptions essentially depict a worst-case scenario about the pre- and post-conditions, respectively, of exploiting a zero day vulnerability. That is, a particular zero day vulnerability may in reality require stronger pre-conditions while implying weaker post-conditions than those stated above. This fact ensures our metric to always yield a conservative result (the metric can be extended to benefit from weaker assumptions when they can be safely made). For a similar purpose, we shall assign one zero day vulnerability to each service although in reality a service may have more vulnerabilities (note that a more conservative result of a metric is one that requires less zero day vulnerabilities).

We more formally state above assumptions in Definition 3 and 4. In Definition 3, the zero day exploit of a privilege will act as a placeholder when we later model isolation techniques. In Definition 4, unlike the exploit of a known vulnerability which has its unique pre- and post-conditions, all zero day exploits share the same hard-coded conditions, as assumed above. Also note that the zero day exploit of each security service has additional post-conditions, which indicates the exploit will re-enable the disabled conditions. For zero day exploits of a privilege p, the pre-conditions include the privilege of every service, unless if that privilege already implies p (in which case including it will result in redundancy). This follows from our assumption that a zero day exploit may potentially lead to any priviledge.

Definition 3 (Zero Day Exploit). *For each $h \in H$ and $x \in (serv(h) \cup priv(h))$, denote by v_x a zero day vulnerability. A zero day exploit is the triple*

- $\langle v_s, h, h' \rangle$ *where* $\langle h, h' \rangle \in conn$ *and* $s \in serv(h')$, *or*
- $\langle v_p, h, h \rangle$ *where* $p \in priv(h)$.

Definition 4 (Condition). *Denote by E_0 the set of all zero day exploits, C_0 the set of conditions $(conn \cup \{\langle x, h \rangle : h \in H, x \in serv(h) \cup priv(h)\})$, and define functions $pre(.) : E_0 \rightarrow C_0$ and $post(.) : E_0 \rightarrow C_0$ as*

- $pre(\langle v_s, h, h' \rangle) = \{\langle h, h' \rangle, \langle s, h' \rangle, \langle p_{min}, h \rangle\}$ *for each* $s \in serv(h)$, *where* p_{min} *is the least privilege on* h.
- $pre(\langle v_p, h, h \rangle) = \{p_s : s \in serv(h), \neg(p \preceq p_s)\}$ *for each* $p \in priv(h)$.
- $post(\langle v_s, h, h' \rangle) = \{p_s\}$ *for each remote service* s *with privilege* p_s.
- $post(\langle v_s, h, h' \rangle) = \{p_s\} \cup C_s$ *for each security service* s, *where* C_s *is the set of conditions disabled by* s.
- $post(\langle v_p, h, h \rangle) = \{\langle p, h \rangle\}$ *for each* $p \in priv(h)$.

[3] While we shall focus on such a restrictive model of zero-day vulnerabilities in this paper, an interesting future direction is to extend the model to address other types of zero-day vulnerabilities, such as a time bomb whose exploitation does not require a network connection.

In Definition 5, a zero day attack graph is composed by relating both exploits of known vulnerabilities and zero day exploits through common pre- and post-conditions. In a zero day attack graph, the exploits of known vulnerabilities can be considered as *shortcuts* that help attackers to satisfy a condition with less zero day exploits. Therefore, exploits of known vulnerabilities here may also be a trust relationship or a misconfigured application, as long as they serve the same purpose of a shortcut for bypassing zero day exploits.

Definition 5 (Zero Day Attack Graph). *Given the set of exploits of known vulnerabilities E_1 and their pre- and post-conditions C_1, let $E = E_0 \cup E_1$, $C = C_0 \cup C_1$, and extend $pre(.)$ and $post(.)$ to $E \rightarrow C$ (as the union of relations). The directed graph $G = \langle E \cup C, \{\langle x, y\rangle : (y \in E \wedge x \in pre(y)) \vee (x \in E \wedge y \in post(x))\}\rangle$ is called a zero day attack graph.*

In Definition 6, the notion of *initial condition* serves two purposes. First, it includes all conditions that are not post-conditions of any exploit (which is the usual interpretation of the notion). Second, it is meant to also include conditions that may be satisfied as the result of insider attacks or user mistakes. In another word, the effect of such attacks or mistakes is modeled as the capability of satisfying post-conditions of an exploit without first executing the exploit[4]. Also note that in the definition, an attack sequence is defined as a total order, which means multiple attack sequences may lead to the same asset. However, this is not a limitation since our metric will not require the attack sequence to be unique, as we shall show.

Instead of the usual way of modeling an asset as a single condition, we take a more general approach. The logical connectives $\wedge$, $\vee$, and $\neg$ respectively model cases where multiple conditions must be satisfied altogether to cause a damage (e.g., the availability of a file with multiple backups on different hosts), cases where satisfying at least one condition will cause the damage (e.g., the confidentiality of the aforementioned file), and cases where conditions are not to be satisfied during an attack (for example, conditions that will trigger an alarm). The asset value is introduced as the relative weight of independent assets.

Definition 6 (Initial Condition, Attack Sequence, and Asset). *Given a zero day attack graph G,*

- *the set of initial conditions is given as any $C_I \subseteq C$ satisfying $C_I \supseteq \{c : (\forall e \in E)(c \notin post(e))\}$,*
- *an attack sequence is any sequence of exploits $e_1, e_2, \ldots, e_j$ satisfying $(\forall i \in [1, j])$ $(\forall c \in pre(e_i))$ $(c \in C_I) \vee (\exists x \in [1, i-1]$ $c \in post(e_x))$,*
- *an asset a is any logical proposition composed of conditions and the logic connectives $\wedge$, $\vee$, and $\neg$ for which an asset value $v(a)$ is given through a function $v(.) : A \rightarrow [0, \infty)$ where A denotes the set of all assets, and*
- *define a function $seq(.) : A \rightarrow 2^Q$ as $seq(a) = \{e_1, e_2, \ldots, e_j : a \in post(e_j)\}$ where Q denotes the set of all attack sequences.*

[4] In a broader sense, we should improve robustness of the model such that it will fail gracefully when assumptions fail, which is beyond the scope of this paper.

Example 2. Figure 2 shows the zero day attack graph of our running example,

- if we do not consider insider attacks or user mistakes, the following attack sequences will lead to the asset $\langle root, 2\rangle$.
  1. $\langle v_{http}, 0, 1\rangle, \langle v_{ssh}, 1, 2\rangle, \langle v_{root}, 2, 2\rangle$
  2. $\langle v_{iptables}, 0, 1\rangle, \langle v_{ssh}, 1, 2\rangle, \langle v_{root}, 2, 2\rangle$
  3. $\langle v_{iptables}, 0, 1\rangle, \langle v_{ssh}, 0, 1\rangle, \langle v_{ssh}, 1, 2\rangle, \langle v_{root}, 2, 2\rangle$
  4. $\langle v_{firewall}, 0, F\rangle, \langle v_{ssh}, 0, 2\rangle, \langle v_{root}, 2, 2\rangle$
- if we consider insider attacks on host 1, only sequence $\langle v_{ssh}, 1, 2\rangle, \langle v_{root}, 2, 2\rangle$ and the fourth attack sequence above will be needed to compromise $\langle root, 2\rangle$.
- if we consider a different asset $\langle root, 1\rangle \wedge \langle root, 2\rangle$, then only the first three attack sequences above can compromise this asset.

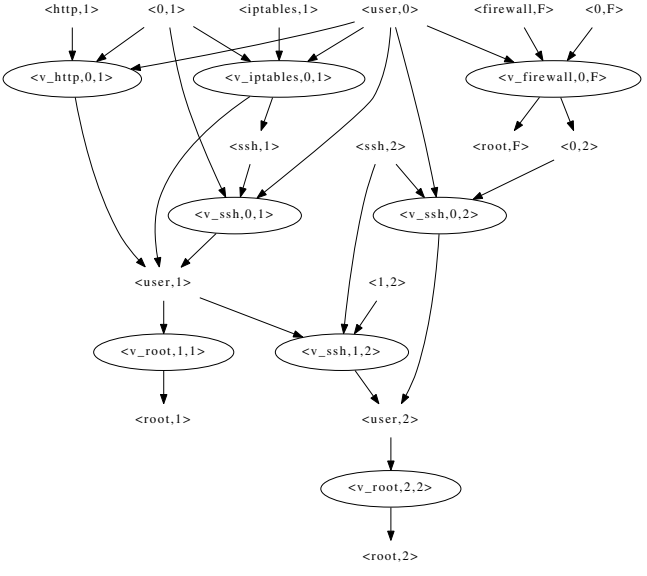


Fig. 2. An Example of Zero Day Attack Graph

2.3 The k-Zero Day Safety Model

In Definition 7, the relation $\equiv_v$ models two distinct cases in which two zero day exploits should only be counted once. First, both exploits involve the same zero day vulnerability. Second, the exploit of a service is related to the exploit of a privilege to indicate that the former will directly yield the privilege due to the lack of isolation between the two (note that we do not model the details of any involved privilege escalation). A probability can be associated to relation $\equiv_v$ to indicate the degree of similarity or isolation, when such information is available. Although the relationship between exploits has distinct meanings in those two cases, the effect of such a relationship towards our metric will be the same. Therefore, the relation $\equiv_v$ models such relationships in a unified way.

Given two sets of zero day exploits, the function $k0d(.)$ counts how many exploits in their symmetric difference are distinct (that is, these exploits cannot be related through $\equiv_v$). In particular, if one of the sets is empty, then the function $k0d(.)$ will yield the number of distinct zero day exploits in the other set. When a probabilistic approach is adopted in defining the relation $\equiv_v$, the function $k0d(.)$ can be revised to give the expected value (mean). The reason of defining the function over the symmetric difference of two sets is given in Theorem 1.

Definition 7 (Relation $\equiv_v$ and Metric Function $k0d(.)$)

- *Define a relation $\equiv_v \subseteq E_0 \times E_0$ such that $e \equiv_v e'$ indicates either e and e' are exploits of the same zero day vulnerability, or $e = \langle v_s, h_1, h_2 \rangle$, $e' = \langle v_p, h_2, h_2 \rangle$ and exploiting s yields p. We say e and e' are distinct if $e \not\equiv_v e'$.*
- *Define a function $k0d(.) : 2^{E_0} \times 2^{E_0} \rightarrow [0, \infty]$ as $k0d(F, F') = max(\{\ |F''| : F'' \subseteq (F \triangle F'), (\forall e_1, e_2 \in F'')\ (e_1 \not\equiv_v e_2)\})$ where $|F''|$ denotes the cardinality of F'', $max(.)$ denotes the maximum value in a set, and $F \triangle F'$ denotes the symmetric difference (that is, $(F \setminus F') \cup (F' \setminus F)$).*

Theorem 1. *The function $k0d(.)$ is a metric.*

In Definition 8, we apply the metric $k0d(.)$ to assets, sets of assets, and a network. First, $k0d(a)$ indicates the minimum number of distinct zero day exploits required to compromise a. This number is unique for each asset, although multiple attack sequences may compromise the asset. The empty set in the definition can be interpreted as the conjunction of all initial conditions (which can always be compromised without any zero day exploit). Second, the metric is applied to a set of independent assets by taking the weighted average with asset values as the weight. Finally, by applying the metric to all assets, we obtain a measurement of a network's resistance to potential zero day attacks.

Definition 8 (k-Zero Day Safety). *Given a zero day attack graph G, the set of initial conditions C_I, and the set of assets A,*

- *for any $a \in A$, we use $k0d(a)$ for $min(\{k0d(q \cap E_0, \phi) : q \in seq(a)\})$ where $min(.)$ denotes the minimum value in a set and q stands for both a sequence and a set. For any $k \in [0, kod(a))$, we say a is k-zero day safe.*
- *given any $A' \subseteq A$, we use $k0d(A')$ for $\sum_{a \in A'}(k0d(a) \cdot v(a))/\sum_{a \in A'} v(a)$. For any $k \in [0, kod(A'))$, we say A' is k-zero day safe.*
- *in particular, when $A' = A$, we say the network is k-zero day safe.*

Example 3. For the running example, suppose all exploits of services involve distinct vulnerabilities except $\langle v_{ssh}, 0, 1 \rangle$, $\langle v_{ssh}, 1, 2 \rangle$, and $\langle v_{ssh}, 0, 2 \rangle$. Assume ssh and $http$ are not protected by isolation but $iptables$ is protected. Then, the relation $\equiv_v$ is shown in the left-hand side of Table 1 where 1 indicates two exploits are related and 0 the opposite (or, by adopting a probabilistic approach, these can be regarded as the probabilities associated with the relation $\equiv_v$).

3 Computing k-Zero Day Safety

This section presents algorithms for computing the k-zero day safety.

3.1 Computing the Value of k

To compute the k-zero day safety of a network, we first derive a logic proposition of each asset in terms of exploits. Then, each conjunctive clause in the disjunctive normal form (DNF) of the derived proposition will correspond to a minimal set of exploits that jointly compromise the asset. The value of k can then be decided by applying the metric $k0d(.)$ to each such conjunctive clause.

Table 1. An Example of Relation $\equiv_v$ (Left) and the Notation Table (Right)

	$\langle v_{iptables}, 0, 1\rangle$	$\langle v_{http}, 0, 1\rangle$	$\langle v_{ssh}, 0, 1\rangle$	$\langle v_{root}, 1, 1\rangle$	$\langle v_{ssh}, 1, 2\rangle$	$\langle v_{firewall}, 0, F\rangle$	$\langle v_{ssh}, 0, 2\rangle$	$\langle v_{root}, 2, 2\rangle$
$\langle v_{iptables}, 0, 1\rangle$	1	0	0	0	0	0	0	0
$\langle v_{http}, 0, 1\rangle$	0	1	0	1	0	0	0	0
$\langle v_{ssh}, 0, 1\rangle$	0	0	1	1	1	0	1	0
$\langle v_{root}, 1, 1\rangle$	0	1	1	1	0	0	0	0
$\langle v_{ssh}, 1, 2\rangle$	0	0	1	0	1	0	1	1
$\langle v_{firewall}, 0, F\rangle$	0	0	0	0	0	1	0	0
$\langle v_{ssh}, 0, 2\rangle$	0	0	1	0	1	0	1	1
$\langle v_{root}, 2, 2\rangle$	0	0	0	0	1	0	1	1

Notation	Explanation
H, h	A set of hosts, a host
S, s	A set of services, a service
P, p	A set of privileges, a privilege
$serv(.)$	Services on a host
$priv(.)$	Privileges on a host
$conn$	Connectivity
v_s, v_p	Zero day vulnerability
$\langle v_x, h, h'\rangle$	Zero day exploit
$pre(.), post(.)$	Pre- and post-conditions
G	Zero day attack graph
C_I	Initial conditions
$e_1, e_2, \ldots, e_j$	Attack sequence
A	Assets
$seq(a)$	Attack sequences compromising a
$\equiv_v$	Relation of non-distinct exploits
$k0d(.)$	The k-zero day safety metric

More precisely, we interpret a given zero day attack graph as a logic program by regarding each exploit or condition as a Boolean variable and by having a logic proposition $c \leftarrow .$ for each initial condition c, a proposition $e \leftarrow \bigwedge_{c \in pre(e)} c$ and a set of propositions $\{c \leftarrow e : c \in post(e)\}$ for each pre- and post-condition relationship, respectively. We can then apply Procedure *k0d_Bwd* shown in Figure 3 to obtain the value of k. The main loop (lines 1-8) computes the k-zero day safety for each asset. The results of all iterations are aggregated as the final output (line 9). The inner loop (lines 3-6) repetitively applies the afore-mentioned logic propositions to derive a formula, which is converted into its DNF (line 7) from which the k-zero day safety is computed (line 8).

Complexity. The procedure's worst-case complexity is exponential in the size of the zero day attack graph. Specifically, the complexity is dominated by the size of the derived proposition L and its DNF; both may be exponential. Indeed, Theorem 2 shows that the problem of computing k-zero day safety is NP-hard.

Theorem 2. *Given a zero day attack graph and an asset a, finding an attack sequence $q \in seq(a)$ to minimize $k0d(q \cap E_0, \phi)$ is NP-complete.*

Procedure $k0d\_Bwd$
Input: A zero day attack graph G, a set of assets A with the valuation function $v(.)$
Output: A non-negative real number k
Method:
1. **For** each asset $a \in A$
2. **Let** L be the logic proposition representing a
3. **While** at least one of the following is possible, do
4. **Replace** each initial condition c with $TRUE$
5. **Replace** each condition c with $\bigvee_{e \in \{e' : c \in post(e')\}} e$
6. **Replace** each non-negated exploit e with $e \wedge (\bigwedge_{c \in pre(e)} c)$
7. **Let** $L_1 \vee L_2 \vee \ldots L_n$ be the DNF of L
8. **Let** $k_a = min(\{k0d(F_i \cap E_0, \phi) : F_i$ is the set of non-negated exploits in $L_i, 1 \leq i \leq n\})$
9. **Return** $\sum_{a \in A}(k_a \cdot v(a))/\sum_{a \in A} v(a)$

Fig. 3. Computing the Value of k

Note that the intractability result here only implies that a single algorithm is not likely to be found to efficiently determine k for all possible inputs (that is, arbitrary zero day attack graphs). However, efficient solutions still exist for practical purposes. We shall examine such a case in the following.

3.2 Determining k-Zero Day Safety for a Given Small k

For many practical purposes, it may suffice to know that every asset in a network is k-zero day safe for a given value of k, even though the network may in reality be k'-zero day safe for some unknown $k' > k$ (note that we have shown determining k' to be intractable). We now describe a solution whose complexity is polynomial in the size of a zero day attack graph if k is a constant compared to this size. Roughly speaking, we attempt to compromise each asset with less than k distinct zero day exploits through a forward search of limited depth. The asset is not k-zero day safe if any branch of the search succeeds, and vice versa.

Specifically, Figure 4 shows the recursive Procedure $k0d\_Fwd$ with two base cases (lines 1-2 and 3-4, respectively) and one recursive case (lines 5-9). In the first base case, the procedure returns $FALSE$ when asset a can be compromised with less than k distinct zero day exploits in T_e. The Sub-Procedure $k0d\_Reachable$ expands T_e with all reachable known exploits since they do not count in terms of the $k0d(.)$ metric. In the second base case, the procedure returns $TRUE$ when the set T_e already has more than k distinct zero day exploits (regardless of whether a can be satisfied with T_c).

The main procedure enters the recursive case only when T_e includes less than k distinct zero day exploits and a cannot be satisfied with T_c. In this case, the Sub-Procedure $k0d\_Reachable$ must have already added all known exploits and their post-conditions to T_e and T_c, respectively. Now the main procedure iteratively visits each zero day exploit e reachable from T_c (line 6), and starts a recursive search from e (line 7). If no such e exists, the procedure will return $TRUE$ indicating the end of a sequence is reached (line 9). If any branch of the search succeeds, $FALSE$ will be recursively returned to indicate a is not k-zero day safe (line 8); otherwise, $TRUE$ is returned (line 9).

Complexity. To find reachable known exploits from E_1, the sub-procedure will check the pre-conditions of each known exploit, which takes time $O(|C| \cdot |E_1|)$. This will be repeated upon adding an exploit to T_e and its post-conditions to T_c. Therefore,

Procedure $k0d\_Fwd$
Input: A zero day attack graph G, an asset a, a real number $k > 0, T_e = \phi, T_c = C_I$
//T_e and T_c include the exploits and conditions visited so far, respectively
Output: $TRUE$, if $k0d(a) > k$; $FALSE$, otherwise
Method:
1. **If** $k0d\_reachable(T_e, T_c) \wedge k0d(T_e) < k$
2. **Return** $FALSE$
3. **ElseIf** $k0d(T_e) \geq k$
4. **Return** $TRUE$
5. **Else**
6. **For** each $e \in E_0 \setminus T_e$ satisfying $pre(e) \subseteq T_c$
7. **If** $\neg\, k0d\_Fwd(G, a, k, T_e \cup \{e\}, T_c \cup post(e))$
8. **Return** $FALSE$
9. **Return** $TRUE$

Sub-Procedure $k0d\_Reachable$
Input: T_e, T_c
Output: $TRUE$ or $FALSE$
Method:
10. **While** $(\exists e \in E_1 \setminus T_e)(pre(e) \subseteq T_c)$
11. **Let** $T_e = T_e \cup \{e\}$
12. **Let** $T_c = T_c \cup post(e)$
13. **Return** $(\bigwedge_{c \in T_c} c \rightarrow a)$

Fig. 4. Determining k-Zero Day Safety for a Given k

$k0d\_Reachable$ takes time $O(|C| \cdot |E_1|^2)$, which is also the complexity for the base cases of the main procedure since it dominates the complexity of other steps. For the recursive case, we have the recurrence formula $t = O(|C| \cdot |E_1|^2) + |E_0| \cdot t'$ where t and t' denote the complexity of the recursive case and that of each recursive call. Since the recursive case cannot be entered unless $k0d(T_e) < k$ and each recursive call will add one more zero day exploit to T_e, the maximum layers of recursion can be written as $l = max(\{|q| : q \text{ is an attack sequence satisfying } k0d(q, \phi) < k + 1\})$. Solving the recurrence formula, we have that $t = |C| \cdot |E_1|^2 \cdot |E_0|^l$. Therefore, the complexity is polynomial in the size of the zero day attack graph if k is a constant.

4 Discussions

In this section, we demonstrate the power of our metric through an example application, network hardening, and discuss issues in instantiating the model.

Network Hardening Using the Metric. Based on the proposed metric, *network hardening* can be defined as making a network k-zero day safe for a larger k. Such a concept generalizes the existing qualitative approach in [22], which essentially achieves $k > 0$. Moreover, the metric immediately imply a collection of hardening options. To see this, we first unfold k based on the model:

$$k = k0d(A) = \sum_{a \in A} (k0d(a) \cdot v(a)) / \sum_{a \in A} v(a) \quad (1)$$

$$k0d(a) = min(\{k0d(q \cap E_0, \phi) : q \in seq(a)\}) \quad (2)$$

$$k0d(q \cap E_0, \phi') = max(\{\, |F| \, : F \subseteq q \cap E_0, \, (\forall e_1, e_2 \in F) \, (e_1 \not\equiv_v e_2)\}) \quad (3)$$

$$seq(a) = \{e_1, e_2, \ldots, e_j : a \in post(e_j), \quad (4)$$

$$(\forall i \in [1, j]) \, (\forall c \in pre(e_i)) \, (c \in C_I) \vee (\exists x \in [1, i-1] \, c \in post(e_x))\} \quad (5)$$

Therefore, k can be increased by:

- Increasing the diversity of services such that exploits will involve more distinct zero-day vulnerabilities (and no longer related by $\equiv_v$) in Equation (3).
- Strengthening isolation techniques for a similar effect as above.
- Disabling initial conditions (e.g., removing a service or a connection) in C_I to yield longer attack sequences in above line (5) (part of Equation (4)).
- Enforcing more strict access control policies to lessen the risk of insider attacks or user mistakes (thus removing conditions from C_I in line (5)).

- Protecting assets with backups (conjunction of conditions) and detection efforts (negation of conditions) to yield a longer sequence in Equation (4).
- Introducing more security services to regulate accesses to remote services in such a way that a longer sequence can be obtained in Equation (4).
- Patching known vulnerabilities such that less shortcuts for bypassing zero day exploits yield a longer sequence in Equation (4).
- Prioritizing the above options based on the asset values in Equation (1) and shortest attack sequences in Equation (2).

Clearly, these hardening options closely match current practices, such as the so-called *layered defense*, *defense in depth*, *security through virtualization*, and *security through diversity* approaches. However, their effectiveness [5] can now be *quantified* in a simple, intuitive way; their cost can now be more easily justified, not based upon speculation or good will, but simply with a larger k.

Instantiating the Model. Since the proposed metric and algorithms are based on an abstract model of networks, how to instantiate the model for given networks is an equally important (and admittedly difficult) issue. We now address several key aspects of the issue while leaving more research to future work.

- While instantiating the model, an uncertain situation can be dealt with by either taking a conservative assumption under which the metric yields a lower k (e.g., any host should be included unless it is believed to be absolutely immune from zero day attacks) or by taking a probabilistic approach (e.g., we have discussed how associating a probability to relation $\equiv_v$ can help to model the degree of similarity in vulnerabilities and strength of isolation). Our future work will further explore such probabilistic approaches.
- An extremely conservative assumption may yield a trivial result (e.g., no network is 1-zero day safe, if insider attacks are considered possible on every host). While such an assumption may be the safest, it is also the least helpful in terms of improving the security since nothing would be helpful.
- The remote services and network connectivity must be identified by examining hosts' configuration. A network scanning is insufficient since it will not reveal services or connectivity currently disabled by security services (e.g., ssh behind $iptables$ in Figure 1). The model is thus more concerned about the existence, instead of the current reachability, of a service or host.
- A zero day attack graph cannot be obtained by injecting zero day exploits into an existing attack graph of known vulnerabilities. The reason is that some unreachable exploits may be discarded in generating an attack graph of known vulnerabilities [1], whereas such exploits may indeed serve as shortcuts for bypassing zero day exploits in a zero day attack graph.
- The model itself does not provide a means for determining which conditions are likely to be subject to insider attacks or user mistakes, which should be determined based on knowledge about access control polices (which users are allowed to do what on which hosts) and how trustworthy each user is.

[5] None of options can always guarantee improved security, which is why we need a metric.

5 Related Work

Standardization efforts on vulnerability assessment include the Common Vulnerability Scoring System (CVSS) [12] which measures vulnerabilities in isolation. The NIST's efforts on standardizing security metrics are also given in [13] and more recently in [19]. The research on security metrics has attracted much attention lately [7]. Earlier work include the a metric in terms of time and efforts based on a Markov model [4]. More recently, several security metrics are proposed by combining CVSS scores based on attack graphs [20,5]. The minimum efforts required for executing each exploit is used as a metric in [2,15]. A mean time-to-compromise metric is proposed based on the predator state-space model (SSM) used in the biological sciences in [9]. The cost of network hardening is quantified in [22]. Security metrics are also developed for specific applications, such as IDSs [8] and distributed trust managment [17].

More closely related to our work, *attack surface* measures how likely a software is vulnerable to attacks based on the degree of exposure [14]. Our work borrows from attack surface the idea of focusing on interfaces, instead of internal details, of a system. However, we apply the idea to a network of computer systems instead of a single software system. Parallel to the study of security metrics, fault tolerance algorithms rely on replication and diversity to improve the availability of services [3]. Our metric provides a means for measuring the effectiveness of systems running such algorithms in the context of a network. Our work is partially inspired by the well known data privacy metric k-anonymity [16] which measures the amount of privacy using an integer regardless of specific application semantic. In our study, we adopt the graph-based representation of attack graphs proposed in [1], which avoids the state explosion problem that may face a model checking-based approach [18].

To the best of our knowledge, few work exist on measuring zero day attacks. An empirical study of the total number of zero day vulnerabilities available on a single day is given based on existing data [11]. If such a result can be obtained or estimated in real time, it may be incorporated into our metric by dynamically adjusting the value of k (a larger k is needed when more vulnerabilities are available). Another recent effort orders different applications in a system by the seriousness of consequences of having a single zero day vulnerability [6]. In contrast to our work, it has a different focus (applications instead of networks) and metric (seriousness of consequences instead of number of vulnerabilities).

6 Conclusion

We have proposed k-zero day safety as a novel security metric for measuring the relative security of networks against potential zero day attacks. In doing so, we have transformed the unmeasureability of unknown vulnerabilities from a commonly perceived obstacle to an opportunity for security metrics. While the general problem of computing the metric is intractable, we have demonstrated that practical security issues can be formulated and solved in polynomial time. For future work, we shall extend the model to address various limitations mentioned in this paper; we shall also integrate the proposed algorithms into existing attack graph-based security tools so to validate their real world effectiveness.

Acknowledgements. The authors thank the anonymous reviewers for their valuable comments. This material is based upon work supported by the National Institute of Standards and Technology Computer Security Division under grant 60NANB9D9192; by the Army Research Office MURI award number W911NF-09-1-0525 administered by The Pennsylvania State University; by the National Science Foundation under the grants CT-20013A, CT-0716567, CT-0716323, and CT-0627493; by the Air Force Office of Scientific Research under grants FA9550-07-1-0527, FA9550-09-1-0421, and FA9550-08-1-0157; by the Army Research Office DURIP award W911NF-09-01-0352; by the Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, and by Fonds de recherche sur la nature et les technologies. 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 sponsoring organizations.

References

1. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of CCS 2002 (2002)
2. Balzarotti, D., Monga, M., Sicari, S.: Assessing the risk of using vulnerable components. In: Proceedings of the 1st Workshop on Quality of Protection (2005)
3. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4), 398–461 (2002)
4. Dacier, M.: Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse (1994)
5. Frigault, M., Wang, L., Singhal, A., Jajodia, S.: Measuring network security using dynamic bayesian network. In: Proceedings of ACM Workshop on Quality of protection (2008)
6. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Proceedings of ACSAC 2009, Washington, DC, USA, 2009, pp. 117–126. IEEE Computer Society Press, Los Alamitos (2009)
7. Jaquith, A.: Security Merics: Replacing Fear Uncertainity and Doubt. Addison Wesley, Reading (2007)
8. Lee, W., Xiang, D.: Information-theoretic measures for anomaly detection. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 130. IEEE Computer Society Press, Los Alamitos (2001)
9. Leversage, D.J., Byres, E.J.: Estimating a system's mean time-to-compromise. IEEE Security and Privacy 6(1), 52–60 (2008)
10. McHugh, J.: Quality of protection: Measuring the unmeasurable? In: Proceedings of the 2nd ACM Workshop on Quality of Protection (QoP 2006), pp. 1–2 (2006)
11. McQueen, M., McQueen, T., Boyer, W., Chaffin, M.: Empirical estimates and observations of 0day vulnerabilities. In: Hawaii International Conference on System Sciences, pp. 1–12 (2009)
12. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. IEEE Security & Privacy Magazine 4(6), 85–89 (2006)
13. National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133 (1985)
14. Manadhata, J.W.P.: An attack surface metric. Technical Report CMU-CS-05-155 (2005)
15. Pamula, J., Jajodia, S., Ammann, P., Swarup, V.: A weakest-adversary security metric for network configuration security analysis. In: Proceedings of the 2nd ACM Workshop on Quality of Protection, pp. 31–38. ACM Press, New York (2006)

16. Samarati, P.: Protecting respondents' identities in microdata release. IEEE Transactions on Knowledge and Data Engineering (TKDE), 1010–1027 (2001)
17. Reiter, M., Stubblebine, S.: Authentication metric analysis and design. ACM Transactions on Information and System Security 2(2), 138–158, 5 (1999)
18. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated generation and analysis of attack graphs. In: Proceedings of the IEEE Symposium on Security and Privacy (2002)
19. Swanson, M., Bartol, N., Sabato, J., Hash, J., Graffo, L.: Security metrics guide for information technology systems. NIST Special Publication 800-55 (2003)
20. Wang, L., Islam, T., Long, T., Singhal, A., Jajodia, S.: An attack graph-based probabilistic security metric. In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 283–296. Springer, Heidelberg (2008)
21. Wang, L., Jajodia, S., Singhal, A., Noel, S.: k-zero day safety: Measuring the security risk of networks against unknown attacks. Technical report, Spectrum Research Repository, Concordia University (2010), http://spectrum.library.concordia.ca/6744/1/k0d.pdf
22. Wang, L., Noel, S., Jajodia, S.: Minimum-cost network hardening using attack graphs. Computer Communications 29(18), 3812–3824, 11 (2006)

Are Security Experts Useful? Bayesian Nash Equilibria for Network Security Games with Limited Information

Benjamin Johnson[1], Jens Grossklags[2], Nicolas Christin[1], and John Chuang[3]

[1] CyLab, Carnegie Mellon University
[2] Center for Information Technology Policy, Princeton University
[3] School of Information, University of California, Berkeley

Abstract. A common assumption in security research is that more individual expertise unambiguously leads to a more secure overall network. We present a game-theoretic model in which this common assumption does not hold. Our findings indicate that expert users can be not only invaluable contributors, but also free-riders, defectors, and narcissistic opportunists. A direct application is that user education needs to highlight the cooperative nature of security, and foster the community sense, in particular, of higher skilled computer users.

As a technical contribution, this paper represents, to our knowledge, the first formal study to quantitatively assess the impact of different degrees of information security expertise on the overall security of a network.

Keywords: Security Economics, Game Theory, Bounded Rationality, Limited Information.

1 Introduction

To what extent does information security expertise help make a network more secure?

Common sense seems to dictate that the more security experts participate in the network, the higher the level of overall security should be, since each expert can contribute her own knowledge to improving the security of all parties. However, such a reasoning does not take into account that most modern information networks such as the Internet are distributed over potentially competing entities. In other words, it may not be in everybody's best interest to contribute resources to secure the network, particularly if the benefits of such contributions are unclear.

As an illustration, consider a simple denial-of-service example. The attacker is a customer of Internet Service Provider (ISP) A, while the target is a customer of a different ISP B. A is not directly connected to B, instead traffic going from A to B may have to cross several other administrative boundaries (e.g., ISPs C, D, ...), causing potential congestion at all of these intermediaries. A very desirable solution, from an engineering standpoint, is to filter traffic at ISP A, before it even enters the rest of the network (*ingress filtering*, [9]). Yet, what is the incentive for ISP A to perform ingress filtering? From A's perspective, ingress filtering means they have to refuse some of their customers' traffic and perhaps deploy intrusive techniques such as deep packet inspection, in order to improve their competitors' security, which may be an economically questionable proposition. Worse even, with modern attack configurations

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 588–606, 2010.

(e.g., botnets [20]), where attack traffic is originating from several different ISPs (A_1, A_2, . . ., A_n) it may be difficult for the source ISPs A_i to distinguish attacks from benign traffic. Thus, ingress filtering could result in unwillingly discarding legitimate traffic, which in turn would certainly lead to loss of reputation, and of customer revenue at the filtering ISPs.

Through the denial-of-service example, we observe that negative externalities play a predominant role in security strategies, even (and especially) when network participants are knowledgeable about security dependencies. Furthermore, in large scale networks such as the Internet, the limited information available about other players renders the decision choices even more complex.

In our previous work [17, 16], we explored the impact of limited information on the strategies chosen by a single expert in a network of naïve players all facing a similar security threat. Naïve players took a myopic view of the network, ignoring all externalities, while the expert player had better threat modeling, and understood externalities arising from the threat faced; this naïve approach is substantiated by empirical surveys of organizational security, e.g., [4]. Addressing the problem relied on a decision-theoretic formulation, where the expert was optimizing her strategy in absence of strategic competition from the naïve players.

In the present work, we address the more general case, where a number k of experts ($1 \leq k \leq N$, with N the size of the network) are making security decisions based both on the security threat they face, and on the behavior of other (expert and naïve) players. The main contribution of this paper is to give formal elements of answer to the question posed in the preamble to this paper, that is, to provide a formal characterization of the impact of the number of competing expert players on the overall security of an information network.

We build upon previously proposed models of network security games [35, 14, 15], which abstract security interactions between network participants by a set of stylized games (weakest link, best shot, total effort). The strategy set for each player is defined by two actions: self-protection, and self-insurance. Self-protection reduces the probability an attack is successful in causing losses to the player, while self-insurance deterministically reduces these losses when a successful attack has happened. Protection is a public good, in that the level of protection chosen by each player impacts (positively or negatively) the overall level of protection all players receive, while self-insurance is a private good, which only benefits the players investing in it.

The remainder of this paper is organized as follows. We review related work in Section 2. In Section 3, we describe our security model, including several simplifications to models presented in our prior work [14]. In Section 4, we explain the methodology of our game-theoretic analysis. Section 5 discusses our numerical results and we conclude in Section 6.

2 Related Work

2.1 Limited Information

Strategic aspects of different interdependency scenarios have been studied in limited information environments (without self-insurance [18]). In particular, Xu computes

Bayesian Nash equilibria for the traditional best shot game when the net benefit of the public good is common knowledge, however, each player's contribution cost is her private information and the general distribution of effort costs are known to all players [38]. Burnett utilizes the weaker link model (in which marginal and declining benefits of contributions above the group minimum exist [8]) to study a model of defense against invasive species. She also assumes common knowledge about the distribution of costs, and limited information about individual effort costs [5]. Somewhat complementary is Manzini and Mariotti's model on negotiations between individual actors and small alliances [24]. In this model, they presume that alliance members are uninformed about the other members' negotiation toughness.

In the context of the value of security information, research has been mostly concerned with incentives for sharing and disclosure. Several models investigate under which conditions organizations are willing to contribute to an information pool about security breaches and investments when competitive effects may result from this cooperation [11, 12]. Empirical papers explore the impact of mandated disclosures [6] or publication of software vulnerabilities [33] on the financial market value of corporations. Other contributions to the security field include the computation of Bayesian Nash outcomes for an intrusion detection game [23], security patrol versus robber avoidance scenarios [27], and the preservation of location privacy in mobile networks [10].

2.2 Mixed Player Populations and Bounded Rationality

In the economic literature, the distinction between perfect rational and bounded rational preferences has found recognition in several psychologically-inspired studies. For example, analytical models for present-biased preferences (i.e., where individuals assign stronger relative importance to a moment in time as it gets closer) consider both sophisticated and naïve agents. The former foresee and comprehend their self-control problems, whereas the latter lack the strategic understanding of their personal fallacies [26]. Acquisti studies these types of preferences in the context of information security [1]. In a related paper, Acquisti and Varian study how individuals interact with a price-discriminating merchant. In their work, myopic agents do not react strategically to the seller's actions [3]. An additional form of preferences is the resolute type where agent stick to their initial strategy during each subsequent choice [36].

However, common to these studies is typically that they evaluate the decision-making behavior of the different agent types separately, whereas in practice it is likely that many marketplace decisions are subject to the interaction with mixed populations with diverse degrees of sophistication. An exception are studies that consider the choice of an intermediary given the interaction with a probabilistically unknown agent type. For example, in signalling games an agent is endowed by nature with a particular type (e.g., good or malicious) and the agent can provide evidence of her type to the intermediary via a costly signal [31]. In a similar fashion, Stigler interpreted the concern for privacy as the rightful ownership of knowledge about a person. Individuals want to achieve and maintain a high degree of reputation while keeping undesirable information a secret [32]. Intermediaries such as employers want to learn the individual's type via signalling or other forms of information revelation.

In agent-based economics, mixed populations have been mostly studied in the tournament context with the goal to determine the most successful agent types in a challenging environment. For example, in the 2001 Trading Auction Competition agents lumped together in small groups were tasked to arrange and automatically purchase travel bundles [37]. Similarly, in the Santa Fe Double Auction tournament, researchers were invited to submit automatic agents to compete on an auction market [29]. In the auction tournament, a majority of agent types did not follow explicit optimization principles, but rather applied heuristic principles [22]. The trading approaches varied in complexity, and the capability to predict and adapt [29]. A more recent example is the lemonade stand tournament in which agents were able to optimize direct sales aspects (such as inventory, quality, and price) and generate indirect payoffs with information trading [25].

In the online safety context, we might wonder whether security markets or large-scale interactions will wipe out any unfamiliar, non-optimizing psychological phenomena (e.g., naïveté) [28]? We doubt that any current trends are able to overcome the rationality obstacles furthered by the complexity of online interactions, and existing information barriers [2]. Therefore, in absence of an effective and sustainable learning process for computer security we have to consider distributional and efficiency consequences of bounded rationality, and the impact on overall system security.

3 Model Overview

The security model adopted in this paper builds upon the hybrid public/private goods model that was defined in [14], and extended and refined in [15, 17]. Specifically, we consider a network of N ($N \in \mathbb{N}$) agents that are subject to an exogenous security threat. Each agent i is endowed with an amount of money M_i, and stands to lose L_i if the attack is successful. For now, we hypothesize that attacks occur with a probability p, exogenous to the game.

Agents can reduce the probability an attack is successful, by choosing a self-protection level e_i ($0 \leq e_i \leq 1$). They can also lower the damage in case an attack is in fact successful, with a self-insurance effort s_i ($0 \leq s_i \leq 1$). Self-protection and self-insurance have nominal costs of b_i and c_i, respectively.

With these definitions, the expected payoff to player i is defined as

$$U_i = M_i - pL_i(1 - s_i)(1 - H(e_1, \ldots, e_N)) - b_i e_i - c_i s_i \,, \tag{1}$$

where H is a joint "contribution" function that characterizes the externalities in the security game being played. As in [35], we consider three different security games:

- *Weakest-link:* The level of protection achieved by player i is that contributed by the least protected player in the entire network. That is, $H(e_1, \ldots, e_N) = \min_{1 \leq j \leq N}\{e_j\}$.
- *Best shot:* The level of protection achieved by player i is that contributed by the most-protected player in the network. That is, $H(e_1, \ldots, e_N) = \max_{1 \leq j \leq N}\{e_j\}$.
- *Total effort:* The level of protection achieved by player i is equal to the average protection level for the N players. That is, $H(e_1, \ldots, e_N) = \frac{1}{N}\sum_{j=1}^{N} e_j$.

All three games have practical applications [14]. Best shot, for instance, is useful in modeling the level of security achieved in a censorship-resilient network like a mix-net [7], where the overall security of the network is guaranteed as long as a single node remains uncompromised. Weakest link is a relatively accurate modeling of perimeter security, where a network, shielded from the rest of the Internet by connecting only through a set of protected machines (routers, firewalls) can be reached by intruders as long as these intruders manage to compromise *one* of these protected machines. Finally, total effort is a good model of parallelized file transfers (e.g., in the case of a peer-to-peer network), where the achieved throughput is roughly equal to the aggregate throughput provided by all peers one is downloading from.

3.1 Simplifications and Additional Assumptions

In an effort to alleviate notational burden, we adopt several simplifying modifications to the model described by Eqn. (1). The purpose of the following discussion is to justify that these simplifications do not considerably restrict the generality of our results and conclusions.

First, the total endowment M_i is not relevant to the strategies that a player chooses, as it is an a priori known constant. We will dispense with this parameter, studying instead the net changes in the utility U_i (which we now denote by u_i).

Since we are now considering only changes in utilities, it makes sense to normalize the remaining parameters, so that the maximum total loss to player i is set to 1. Henceforth we assume that $b_i, c_i, L_i \in [0, 1]$.

The next change is to incorporate the attack probability p into the loss parameter L_i, treating L_i an *expected* loss, as opposed to a realized loss. This modification is mathematically without loss of generality since the model was simply treating p as a known constant.

Ultimately, Eqn. (1) becomes:

$$u_i = -L_i(1 - s_i)(1 - H(e_1, \ldots, e_N)) - be_i - cs_i \ , \tag{2}$$

with $0 \leq b_i, c_i, L_i, e_i, s_i \leq 1$, and H as defined above for the three games under consideration.

All remaining assumptions about the model are the same as those adopted in [17]. To simplify the analysis we assume that self-insurance and protection costs are homogeneous.[1] To focus on the interesting case in which protection and self-insurance are rationally interesting prospects we assume that $b_i, c_i \leq L_i$. Due to our intent to focus on utility-maximizing equilibria we assume that e_i, s_i are in fact discrete decision variables taking values in $\{0, 1\}$. Each of these assumptions is discussed more thoroughly in [17].

Finally, to improve readability of the presentation, we will derive our initial results on protection equilibrium while ignoring caveats resulting from the availability of self-insurance. Expanded results incorporating self-insurance are included in Appendix A.

[1] Note that, for the full information, only-expert player case, we explored the case where b_i and c_i are heterogeneous in [15].

4 Analysis

4.1 Methodology

To determine how the composition of experts can affect systemwide network protection in a security context, we analyze three distinct N-player security games in which there are k selfish experts ($0 \leq k \leq N$) and $N - k$ naïve players. We assume that the experts understand the dynamics of each game and choose strategies to maximize their own expected payoffs, while the naïve players choose whether to protect based on a simple cost-benefit analysis – comparing protection costs to their individual expected loss in the event of protection failure.

For expert agents, we distinguish between knowledge conditions about other players' expected losses. In a complete information environment, each expert has full knowledge of the expected losses of all players resulting from network protection failure. In an incomplete information environment, each expert knows her own expected loss in the event of protection failure, but only knows the distribution on these expected losses for all players (the uniform distribution).

Finally, to illustrate the drawbacks of selfish equilibria more generally, we compute the social optimum strategy for each game. To facilitate comparisons with the scenario involving selfish experts, we characterize the social optimum from the individual perspective of N cooperative agents.

4.2 Protection Strategies in the Best Shot Security Game

In the best shot security game, the security of the network is determined by the highest protection level of any player (i.e. $H(e_1, \ldots, e_N) = \max_j e_j$). The upshot of this game is that if a single player pays for full protection, then the whole network is protected. This scenario gives selfish experts an incentive to shirk protection responsibilities and pass the buck to other players. In the paragraphs below, we consider three different types of player configurations: k selfish experts with incomplete information, k selfish experts with complete information, and N cooperative experts.

Best Shot: *k* selfish experts with incomplete information. In this player configuration, the rationale for expert i goes as follows. If she protects, she pays b; and if she does not protect, she pays $L_i \cdot FailE^*_{\neg i} \cdot FailN^*_{\neg i}$, where $FailE^*_{\neg i}$ is the probability (over the distribution on the expected losses L_j) that all other experts fail to protect, and $FailN^*_{\neg i}$ is the probability that all naive players fail to protect. We can easily compute $FailN^*_{\neg i} = b^{N-k}$ (since naïve player j protects if and only if $L_j \geq b$). It follows that expert i should protect if and only if $b \leq L_i \cdot FailE^*_{\neg i} \cdot b^{N-k}$, or equivalently, $L_i \geq \frac{1}{b^{N-k-1} \cdot FailE^*_{\neg i}}$. To solidify the proper strategy, it remains to determine $FailE^*_{\neg i}$. To do this, we assume that the strategy is a symmetric Bayesian Nash equilibrium. In this event, all experts have the same (pure) strategy of the form: "protect if and only if $L_j \geq \alpha$," (where α depends on b, k, and N). Since α represents the lower protection threshold for all experts, we have $FailE^*_{\neg i} = \alpha^{k-1}$, and we can solve for α using $\alpha = \frac{1}{b^{N-k-1} \cdot \alpha^{k-1}}$. The result is $\alpha = b^{\frac{-N+k+1}{k}}$. Thus the equilibrium strategy for expert i is to protect iff $L_i \geq b^{\frac{-N+k+1}{k}}$.

If $k < N$, then this strategy reduces to the simple conclusion that expert i should *never* protect.[2] The explanation is that if there are any naïve players in the network, the likelihood that any such player fails to protect is at most the cost of protection. Thus the expected loss of expert i in the event of a protection failure is strictly less than the cost of protecting.

If there are N players, then the above strategy simplifies to "protect if and only if $L_i \geq b^{\frac{1}{N}}$." Uniform adoption of this strategy among all experts yields a viable symmetric Bayesian Nash equilibrium.

Best Shot: *k* selfish experts with complete information. In this player configuration, the rationale for expert i goes as follows. If any of the naive players draws $L_i > b$ they will protect, so none of the experts needs to (or in their own selfish interest ought to) protect. If none of the naive players make such draws, then the game reduces to one consisting of all expert players. Because the losses are drawn from a uniform distribution, one of the experts will have the highest draw L_j^*. If $L_j^* \geq b$, then this expert should protect. All other experts can refrain from protecting. If all of the L_j are less than b, then the network will remain unprotected. To summarize, the equilibrium strategy is for expert i to protect if and only if $b \leq L_j < L_i$ for all $j \neq i$.

Best Shot: *N* cooperative experts with complete information. We next consider what happens in the best shot game if experts can cooperate to share wealth in an effort to reduce their overall expected costs. First note that if the network is not protected, then the sum of players' losses is $\sum_{j=1}^{N} L_j$. If $\sum_{j=1}^{N} L_j \geq b$, then for the purpose of maximizing expected outcomes, it would be advantageous for each expert to contribute toward ensuring protection. One quite reasonable strategy is for player i to pay $\frac{L_i \cdot b}{\sum_{j=1}^{N} L_j}$, and for the sum paid (which is b) to be used to pay for a single player's protection cost (say the player with the highest L_j). This strategy is fair, symmetric, and results in the lowest possible sum of expected long term costs for all players.

4.3 Protection Strategies in the Weakest Link Security Game

In the weakest link security game, the security of the network is determined by the lowest protection level of any player (i.e. $H(e_1, \ldots, e_N) = \min_j e_j$). The upshot of this game is that for the network to be protected, every player must pay for protection. This scenario gives rational players cause to worry whether other players will defect, ultimately resulting in a notable systemwide protection disincentive. Our analysis of player configurations below is structured analogously to the best shot case.

Weakest Link: *k* selfish experts with incomplete information. In this player configuration, the rationale for expert i can be framed as follows. If she does not protect, then she loses L_i, while if she protects, she pays $b + L_i \cdot (1 - ProtE^*_{\neg i} \cdot ProtN^*_{\neg i})$, where $ProtE^*_{\neg i}$ is the probability that all other experts protect, and $ProtN^*_{\neg i}$ is the probability that all naive players protect. The condition for player i to choose protection can be expressed as $b + L_i \cdot (1 - ProtE^*_{\neg i} \cdot ProtN^*_{\neg i}) \leq L_i$, and this can be simplified (assuming the probabilities in question are non-zero) to the condition: $L_i \geq \frac{b}{ProtE^*_{\neg i} \cdot ProtN^*_{\neg i}}$.

[2] This idea confirms the result from [17] which considered the case of one expert.

We know that $ProtN^*_{\neg i} = (1-b)^{N-k}$, because naive player j protects if and only if $L_j \geq b$; and it remains to determine $ProtE^*_{\neg i}$. As we did in the best shot case, we will assume that the strategy for player i is one component of a symmetric Bayesian Nash equilibrium, and that expert j plays an identical strategy of the form "protect if and only if $L_j \geq \gamma$", for some γ depending on b, k, and N. Under these conditions, we have $ProtE^*_{\neg i} = (1-\gamma)^{k-1}$, and so we may solve for γ using $\gamma = \frac{b}{(1-\gamma)^{k-1}(1-b)^{N-k}}$.

For the purpose of numerical analysis we can rewrite the above equation as

$$\gamma(1-\gamma)^{k-1} = \frac{b}{(1-b)^{N-k}}. \tag{3}$$

Unfortunately, there is no algebraic solution for γ when $k \geq 5$. Figure 1 plots γ as a function of b for various values of k and N.

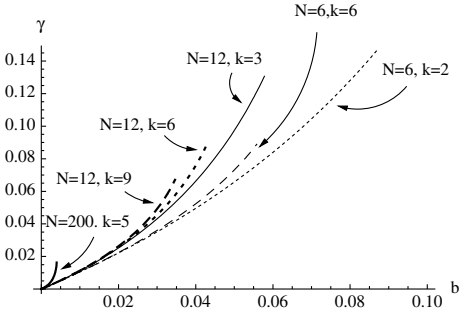


Fig. 1. Evolution of γ as defined by Eqn. (3). We plot the evolution of γ as a function of the protection cost b for various network sizes N and various number of expert users k Recall that γ is an upper bound for expected losses that determines whether an expert in the given configuration will participate in a protection equilibrium. Player i protects if and only if $L_i \geq \gamma$.

Weakest Link: k selfish experts with complete information. In this player configuration, if any naive player j draws $L_j < b$ he will not protect, and so systemwide protection will fail, and so no expert will protect. Similarly if expert j draws $L_j < b$ then (absent cooperation opportunities) she will not protect, and again systemwide protection will fail. The only (noncooperative) scenario in which the network is protected, thus occurs when each and every player draws $L_j \geq b$. In this case, everyone will protect, and the network will be protected. The equilibrium strategy for expert i is thus to protect if and only if every player j draws $L_j \geq b$.

Weakest Link: N cooperative experts with complete information. The cost to protect the entire network in the weakest link game is bN. Thus if $\sum_{j=1}^{N} L_j \geq bN$, it is, over the long term, advantageous for all the players to cooperate and protect the network. A sensible cooperative equilibrium strategy in this instance is for player i to pay $\frac{L_i bN}{\sum_{j=1}^{N} L_j}$ if and only if $\sum_{j=1}^{N} L_j \geq bN$; and for the total amount collected (bN) to be divided equally among all players for the purpose of network protection.

4.4 Protection Strategies in the Total Effort Security Game

In the total effort security game, the security of the network is determined by the average protection level of players, (i.e. $H(e_1, \ldots, e_N) = \sum_j \frac{1}{N} e_j$). The upshot of this game is that every player receives a partial benefit from his or her contribution to the protection effort. It turns out that this benefit does not depend on the number of other players who choose to protect. Thus a player's decision to contribute to the protection effort can be made without considering the choices of other players. We discuss three player configuration below, following the format of the other two games described above.

Total Effort: k selfish experts with incomplete information. In this configuration, player i's strategy can be framed as follows. If she protects, she pays $b + L_i \cdot (1 - \frac{ExpProt^*_{\neg i}+1}{N})$ where $ExpProt^*_{\neg i}$ is the expected number of players in the network other than i that choose protection. If she does not protect she pays $L_i \cdot (1 - \frac{ExpProt^*_{\neg i}}{N})$. Thus player i should protect if and only if $b + L_i \cdot (1 - \frac{ExpProt^*_{\neg i}+1}{N}) \leq L_i \cdot (1 - \frac{ExpProt^*_{\neg i}}{N})$. This inequality simplifies to $L_i \geq bN$. Notably, this condition does not depend on the value of $ExpProt^*_{\neg i}$ or any other variable related to the choices of other players. Expert i should protect if and only if $L_i \geq bN$.

Total Effort: k selfish experts with complete information. Similar to the situation above, if experts have complete information in a total effort security game, they can determine other players' protection incentives. However, the core economic structure remains the same as in the case incomplete information. That is, for protection to be a worthwhile investment strategy for player i, it is necessary and sufficient that $L_i \geq bN$.

Total Effort: N cooperative experts with complete information. If coordination is allowed in this game, the situation is analogous to the cooperative efforts in the weakest link game. The cost to protect the entire network in the total effort game is still bN. Thus if $\sum_{j=1}^{N} L_j \geq bN$, it is, over the long term, advantageous for all the players to cooperate and protect the network. A sensible cooperative equilibrium strategy in this instance is for player i to pay $\frac{L_i bN}{\sum_{j=1}^{N} L_j}$ if and only if $\sum_{j=1}^{N} L_j \geq bN$; and for the total amount collected (bN) to be divided equally among all players for the purpose of network protection. Such a strategy, if agreed upon in advance, in guaranteed to yield a higher expected payoff for every player over the course of time – i.e. over the course of numerous draws of expected losses L_i from the uniform distribution on $[0, 1]$.

5 Numerical Illustrations and Observations

To synthesize the information from our previous analysis, we compare, using tables and graphs, the decision outcomes resulting from various configurations of information conditions and expert player configurations. For each game, we compute the conditions for expert i to protect, the probability that expert i protects (over the distribution on L_i), the expected contribution of expert i, and the expected level of network protection – where 1 denotes complete network protection and 0 denotes no protection. All

probabilities and expected values are computed assuming each L_i is drawn independently from the uniform distribution on $[0, 1]$. Our discussion of these numerical results focuses primarily on the effect of experts on the probability of network protection.

5.1 Best Shot

Systemwide protection results for the best shot game are shown in Table 1.

Table 1. Best Shot Security Game: Bayesian Nash Symmetric Protection Equilibrium with N Players

Composition of Expert Players	Experts' Knowledge of Losses	Conditions under which Expert i Protects	Probability that Player i Protects	Expected Contribution from Player i	Probability of Network Protection
k Selfish $1 \leq k < N$	incomplete	Never	0	0	$1 - b^{N-k}$
N Selfish	incomplete	$L_i \geq b^{\frac{1}{N}}$	$1 - b^{\frac{1}{N}}$	$b \cdot \left(1 - b^{\frac{1}{N}}\right)$	$1 - b$
k Selfish $1 \leq k \leq N$	complete	$\forall$ Expert $j \neq i, L_i > L_j$ and $\forall$ Naïve $j, L_j < b$ and $L_i \geq b$	$\frac{b^{N-k}(1-b)}{k}$	$\frac{b^{N+1-k}(1-b)}{k}$	$1 - b^N$
N Naïve	-	$L_i \geq b$	$1 - b$	$b(1-b)$	$1 - b^N$
N Cooperative	complete	$\sum_i L_i \geq b$	$1 - \frac{b^N}{N!}$	$\frac{b}{N}\left(1 - \frac{b^N}{N!}\right)$	$1 - \frac{b^N}{N!}$

Perhaps the most interesting point to observe about this table in terms of overall network protection is that, in the incomplete information case, increasing the number of experts actually decreases the protection level of the network. This is because experts are incentivized to free-ride when other players are paying for protection. The more experts there are in the network, the more freeriding takes place.This effect can be seen directly in Figure 2 which plots the expected systemwide network protection level as a function of protection costs for various configurations of experts in a 6-player best shot game.

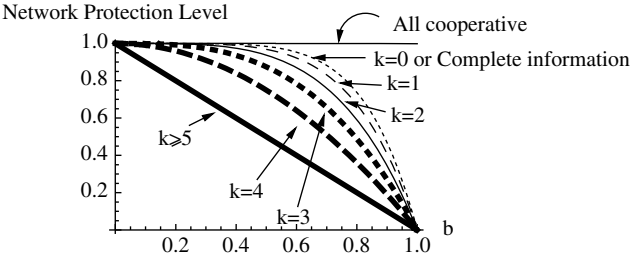


Fig. 2. Best shot. Evolution of the network protection level as a function of the protection cost b. The different plots vary the number of experts k in a network of $N = 6$ players. We observe that the fewer experts participating in the game, the higher the network protection level is, on average.

In the complete information case, the expected protection outcome for the network is the same regardless of the number of experts. The net effect is always that the network will always be protected if any of the players draws $L_i \geq b$.

The best shot game is especially effective at highlighting the advantage of cooperation. In a configuration of cooperative experts, each player bears a substantially smaller expected cost burden, and the network is far more likely to be secure, in comparison to the analogous configuration of selfish experts with complete information.

5.2 Weakest Link

Systemwide protection results for the weakest link game are shown in Table 2. As was the case in the shot game, the limited information scenario has the property that increasing the number of experts in the game decreases the protection level of the network. Experts are influenced by the risk that other players will be the weak link that causes a protection failure. This risk has a cascading effect, so that as more experts are added, the risk of defection increases.

Except for the similarity between 1 and 2 experts, each additional expert reduces the likelihood of systemwide protection. Figure 3 plots the expected systemwide network protection level as a function of protection costs for various configurations of experts in a 6-player weakest link game.

Observe that in configurations of experts with limited information in the weakest link game, there is an abrupt cut-off in the protection levels facilitating protection conditions. As shown in the decision analysis, once the price of protection exceeds a given value, there no longer exist any protection equilibrium, and so all the experts in the network will choose not to protect.

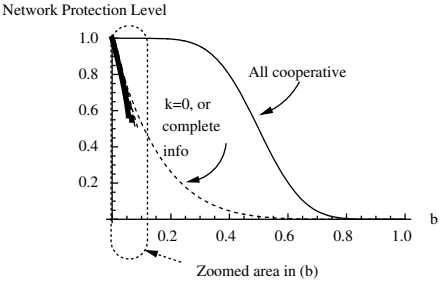
In the case of complete information, the only scenario in which the network is protected is when, for every player, the individual expected loss is more than the protection cost. The same network protection outcome results from a configuration of N naïve

Table 2. Weakest Link Security Game. Bayesian Nash Symmetric Protection Equilibrium with N Players

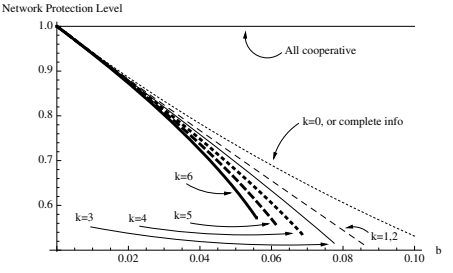
Composition of Expert Players	Experts' Knowledge of Losses	Conditions under which Expert i Protects	Probability that Player i Protects	Expected Contribution from Player i	Probability of Network Protection
k Selfish $1 \leq k \leq N$	incomplete	* $L_i \geq \gamma$	$1-\gamma$	$b(1-\gamma)$	$(1-b)^{N-k}(1-\gamma)^k$
k Selfish $1 \leq k \leq N$	complete	$\forall j, L_j \geq b$	$(1-b)^N$	$b(1-b)^N$	$(1-b)^N$
N Naïve	-	$L_i \geq b$	$(1-b)$	$b(1-b)$	$(1-b)^N$
N Cooperative	complete	$\sum_j L_j \geq bN$	** ρ	$b\rho$	ρ

* γ is the least positive solution (if a solution exists) to the equation $\gamma(1-\gamma)^{k-1} = \frac{b}{(1-b)^{N-k}}$.
** ρ can be computed using well-known formulas for the uniform sum distribution. If $bN \leq 1$, we have $\rho = 1 - \frac{(bN)^N}{N!}$. More generally, $\rho = \int_{bN}^{N} \frac{1}{2(N-1)!} \sum_{k=0}^{N} (-1)^k \binom{n}{k} (bN-k)^{N-1} \cdot \mathrm{sgn}(bN-k)$.

players (although the naïve players pay a much higher expected cost for the same net protection effect).



(a) Evolution of expected network protection as a function of b. The thick lines represent cases on which we zoom in (b).



(b) Zoom on small values of the protection cost b.

Fig. 3. Weakest link. Evolution of the network protection level as a function of the protection cost b. The short lines illustrate the presence of limiting conditions on protection equilibria for this game. Where the lines end, the expected network protection level becomes zero. Also note that the cases $k = 1$ and $k = 2$ produce identical curves.

In the cooperative game, the expected protection cost for each player is a bit higher, but the overall expected cost is less (compared to the analogous game with selfish experts), and systemwide network protection is substantially improved. Computing the expected protection contribution for this game requires determining the likelihood that a sum of independently and uniformly distributed random variables from $[0, 1]$ exceeds an arbitrary threshold (bN). The desired probability is easily computed using well-known formulas for the uniform sum distribution, although it is somewhat cumbersome to express.

5.3 Total Effort

Systemwide protection results for the total effort game are shown in Table 3. This game differs from the best shot and weakest link games in that the decision to protect does not depend on the choices of the other players. It is individually worthwhile for an expert to protect the network only if the cost of protection is a $\frac{1}{N}$ fraction of the player's expected loss. For experts, this results in a high threshold for protection – an unfortunate occurrence since protection by any individual player would increase the utility for every player.

In the configuration consisting only of naïve players, protection is much more likely, even though much of the time (over the distribution on L_i) paying that protection cost is a losing investment. This can be seen by comparing the naïve configuration to the cooperative one as shown in Figure 4. The expected network protection level for naïve users exceeds the social optimum protection level whenever $b \geq \frac{1}{2}$.

The cooperative game affords the same result as in the weakest link game.

Table 3. Total Effort Security Game. Bayesian Nash Symmetric Protection Equilibrium with N Players

Composition of Expert Players	Experts' Knowledge of Losses	Conditions under which Expert i Protects	Probability that Player i Protects	Expected Contribution from Player i	Expected Level of Network Protection
k Selfish $1 \leq k \leq N$	incomplete	$L_i \geq bN$	$1 - bN$	$b(1 - bN)$	$1 - bN$ if $bN < 1$
k Selfish $1 \leq k \leq N$	complete	$L_i \geq bN$	$1 - bN$	$b(1 - bN)$	$1 - bN$ if $bN < 1$
N Naïve	-	$L_i \geq b$	$(1 - b)$	$b(1 - b)$	$(1 - b)$
N Cooperative	complete	$\sum_j L_j \geq bN$	* ρ	$b\rho$	ρ

* ρ can be computed using well-known formulas for the uniform sum distribution. If $bN \leq 1$, we have $\rho = (1 - \frac{(bN)^N}{N!})$. More generally, $\rho = \int_{bN}^{N} \frac{1}{2(N-1)!} \sum_{k=0}^{N} (-1)^k \binom{n}{k} (bN - k)^{N-1} \cdot \text{sgn}(bN - k)$.

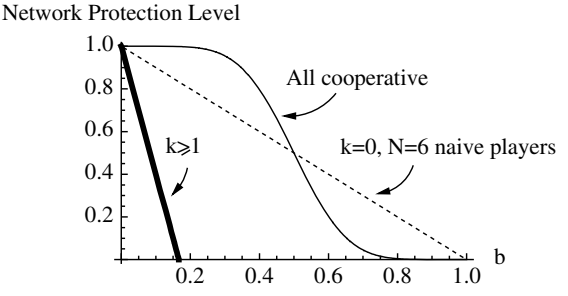


Fig. 4. Total effort. Evolution of the network protection level as a function of the protection cost b. For any number of experts $k \geq 1$, the network protection level is inferior to that obtained with a network solely consisting of naïve players. The cooperative equilibrium, here, provides a less desirable overall system outcome as soon as b exceeds 0.5.

6 Discussion and Conclusion

We carried out a game-theoretic analysis of the impact of the number of security experts in a network of competitive players facing a common security threat, under limited information. Our results are somewhat counter-intuitive, in that in all three scenarios considered (best shot, weakest-link and total effort), the addition of selfish experts actually *never* increases the expected level of systemwide network protection. This outcome is rather unexpected, considering our previous result [17], which showed that a lone expert in a network of naïve players stood to make considerable payoff gains compared to naïve players, regardless of the information condition in which the game was played. We thus could have expected that adding more experts would help the network as a whole, but the force of the negative externalities in play *in all scenarios* actually drives the overall security of the network down.

On the other hand, and much less surprisingly, having N *cooperative* experts improves individual expected payoffs, and dramatically increases the expected level of systemwide network protection (relative to N selfish experts).

In sum, we showed in [17] that user expertise could lead to strong individual benefits even in the case of limited information. The present paper shows that, from a community standpoint, expert users may not only be invaluable contributors, but also free-riders preying on the weaknesses of naïve users. As a result, even networks populated with a large share of sophisticated users will frequently not achieve superior security. One of the direct outcomes of this work is that user education needs to highlight the cooperative nature of security, and heighten the community sense of better educated computer users.

While the model proposed has some limitations, and oversimplifications, we believe that it raises a number of points that warrant further consideration by security practitioners.

6.1 Caveats

Our model of limited information is decidedly simple. Only one parameter is unknown to player i, the expected loss L_j for players $j \neq i$. Even so, we assume that player i knows the probability distribution of L_j.[3] In practice, even that probability distribution may be unknown to most players, and the nominal costs of protecting or self-insuring (b and c respectively) may also be heterogeneous, and unknown. We note that, if all external costs and potential losses are unknown, all players may have to resort to naïve strategies since they have no way of estimating the externalities in play. More generally, our crude, binary, distinction between naïve and expert players is much more nuanced in practice, where we likely would find a near-continuum of expertise levels, ranging from the abysmally clueless to the highly cognizant. Nonetheless, we believe that our model captures the salient distinction between users who do understand externalities and the role they play, and those who do not.

6.2 Applications

There are three immediate applications of the mathematical results we obtained.

Tragedy of the commons in best-shot games. Best shot security environments are in theory extremely resilient to security threats, since only one agent needs to contribute vigorously (or act securely) to save the whole network. However, our results indicate that there may be a "tragedy of the commons" situation in play, where none of the agents actually has an interest in fulfilling that role. The interesting point is that limited information exacerbates this phenomenon, something we first identified in prior work [14].

Fragility of weakest-link games. Likewise, we discovered that weakest-link games have an increased fragility in presence of expert players and limited information. Weakest-link games are known to offer a vexing set of negative externalities [15, 14, 35], and

[3] For the sake of simplicity of the presentation, we assumed the uniform distribution, but similar derivations could be carried out for any known, "well-behaved" probability distribution.

they rarely converge to a satisfying equilibrium [34]. Unfortunately, one outcome of this work is that limited information degrades even more the likelihood of reaching an acceptable state when players understand the externalities at hand.

Developing side-payment mechanisms. The main take-away from the mathematical formalism proposed in this paper is that, while selfish experts lead to extremely undesirable protection equilibria in the presence of limited information, cooperative experts can completely change the outcome of a game, *and* cooperation can likely be enforced by simply considering side-payments between users [21]. Determining how these side-payments could be carried out highly depends on the context in which the game is played, but it is not inconceivable to imagine bilateral contractual relationships between pairs of players. For example, between ISPs, such side-payments could be made at the time transit agreements are established.

6.3 Public Policy Impact

Beyond the applications outlined above, we believe that our work has a clear public policy impact, on at least two levels.

Inter-agency collaboration. In the United States, and in many other countries, national security is not within the purview of a single governmental agency. While, in the U.S., the Department of Homeland Security was created after 9/11 as a vehicle to centralize national security decisions, the reality is that a multitude of government agencies (CIA, NSA, Pentagon, Coast Guard, Bureau of Alcohol, Tobacco and Firearms etc.) are involved, at one degree or another, in such decisions. Such agencies are usually competing for funding budgets, and could be viewed as competing players in national security matters. What we have seen from our model, is that having qualified personnel in all of these agencies actually exacerbates the harmful effects of competition. As such, enforcing collaboration between the different agencies playing a role in containing a threat is not just a desirable goal, it is an absolute necessity to achieve any level of security.

User education The need for technical user education and training has been repeatedly emphasized by security researchers, based on empirical field studies (e.g., [30]) and analytical results (e.g., [17, 16]). Yet, while user education has primarily focused on better understanding of the threats faced, our results indicate that, an equally, if not more important aspect of user education should be in highlighting the cooperative nature of security, and fostering the community sense of high-skilled computer users.

6.4 Future Research Directions

This paper closes an arc of research on game-theoretic modeling of negative externalities in information security, by complementing our previous work on game-theoretic models of complete information [14, 15], and decision-theoretic models of incomplete information [17, 16]. This does not mean that there are no other interesting directions to pursue on these models, but we believe that to go to the next level, we need to inform

our models with field data. In particular, we are interested in removing assumptions on the nature of the various cost functions on which we rely. For instance, surely, the cost incurred by deploying protection measures is not perfectly linear in the degree of protection deployed. Behavioral biases (e.g., risk-aversion [19]) can also create a dichotomy between actual and perceived costs. We have started preliminary investigations through user studies [13], but plan on enriching our models with field data applied to specific contexts. For instance, we are considering whether we can find evidence of correlation between (de)peering decisions and the amount of undesirable traffic (e.g., attack traffic) traversing different ISP networks.

References

1. Acquisti, A.: Privacy in electronic commerce and the economics of immediate gratification. In: Proceedings of the 5th ACM Conference on Electronic Commerce (EC 2004), New York, NY, May 2004, pp. 21–29 (2004)
2. Acquisti, A., Grossklags, J.: Privacy and rationality in individual decision making. IEEE Security & Privacy 3(1), 26–33 (2005)
3. Acquisti, A., Varian, H.: Conditioning prices on purchase history. Marketing Science 24(3), 367–381 (Summer 2005)
4. Bashir, M., Christin, N.: Three case studies in quantitative information risk analysis. In: Proceedings of the CERT/SEI Making the Business Case for Software Assurance Workshop, Pittsburgh, PA, pp. 77–86 (September 2008)
5. Burnett, K.: Introductions of invasive species: Failure of the weaker link. Agricultural and Resource Economics Review 35(1), 21–28 (2006)
6. Campbell, K., Gordon, L., Loeb, M., Zhou, L.: The economic cost of publicly announced information security breaches: Empirical evidence from the stock market. Journal of Computer Security 11(3), 431–448 (2003)
7. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)
8. Cornes, R.: Dyke maintenance and other stories: Some neglected types of public goods. Quarterly Journal of Economics 108(1), 259–271 (1993)
9. Ferguson, P., Senie, D.: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, RFC 2267 (January 1998)
10. Freudiger, J., Manshaei, M., Hubaux, J.-P., Parkes, D.: On non-cooperative location privacy: A game-theoretic analysis. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009), Chicago, IL, p. 324–337 (November 2009)
11. Gal-Or, E., Ghose, A.: The economic incentives for sharing security information. Information Systems Research 16(2), 186–208 (2005)
12. Gordon, L.A., Loeb, M., Lucyshyn, W.: Sharing information on computer systems security: An economic analysis. Journal of Accounting and Public Policy 22(6), 461–485 (2003)
13. Grossklags, J., Christin, N., Chuang, J.: Predicted and observed behavior in the weakest-link security game. In: Proceedings of the 2008 USENIX Workshop on Usability, Privacy and Security (UPSEC 2008), San Francisco, CA (April 2008)
14. Grossklags, J., Christin, N., Chuang, J.: Secure or insure? A game-theoretic analysis of information security games. In: Proceedings of the 2008 World Wide Web Conference (WWW 2008), Beijing, China, pp. 209–218 (April 2008)
15. Grossklags, J., Christin, N., Chuang, J.: Security and insurance management in networks with heterogeneous agents. In: Proceedings of the 9th ACM Conference on Electronic Commerce (EC 2008), Chicago, IL, pp. 160–169 (July 2008)

16. Grossklags, J., Johnson, B., Christin, N.: The price of uncertainty in security games. In: Proceedings (online) of the Eighth Workshop on the Economics of Information Security (WEIS), London, UK (June 2009)
17. Grossklags, J., Johnson, B., Christin, N.: When information improves information security. In: Proceedings of the 2010 Financial Cryptography Conference (FC 2010), Canary Islands, Spain (January 2010)
18. Hirshleifer, J.: From weakest-link to best-shot: The voluntary provision of public goods. Public Choice 41(3), 371–386 (1983)
19. Kahneman, D., Tversky, A.: Prospect theory: An analysis of decision under risk. Econometrica XLVII, 263–291 (1979)
20. Kandula, S., Katabi, D., Jacob, M., Berger, A.: Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In: Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI 2005), Boston, MA, pp. 287–300 (May 2005)
21. Katz, M., Shapiro, C.: Network externalities, competition, and compatibility. American Economic Review 75(3), 424–440 (1985)
22. Lettau, M., Uhlig, H.: Rules of thumb versus dynamic programming. American Economic Review 89(1), 148–174 (1999)
23. Liu, Y., Comaniciu, C., Man, H.: A Bayesian game approach for intrusion detection in wireless ad hoc networks. In: Proceedings of the Workshop on Game Theory for Communications and Networks, page Article No. 4 (2006)
24. Manzini, P., Mariotti, M.: Alliances and negotiations: An incomplete information example. Review of Economic Design 13(3), 195–203 (2009)
25. Noy, A., Raban, D., Ravid, G.: Testing social theories in computer-mediated communication through gaming and simulation. Simulation & Gaming 37(2), 174–194 (2006)
26. O'Donoghue, T., Rabin, M.: Doing it now or later. American Economic Review 89(1), 103–124 (1999)
27. Paruchuri, P., Pearce, J., Marecki, J., Tambe, M., Ordonez, F., Kraus, S.: Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, pp. 895–902 (May 2008)
28. Rabin, M.: A perspective on psychology and economics. European Economic Review 46(4-5), 657–685 (2002)
29. Rust, J., Miller, J., Palmer, R.: Characterizing effective trading strategies: Insights from a computerized double auction tournament. Journal of Economic Dynamics and Control 18(1), 61–96 (1994)
30. Sheng, S., Magnien, B., Kumaraguru, P., Acquisti, A., Cranor, L., Hong, J., Nunge, E.: Anti-Phishing Phil: The design and evaluation of a game that teaches people not to fall for Phish. In: Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS 2007), Pittsburgh, PA, July 2007, pp. 88–99 (2007)
31. Spence, A.: Job market signaling. Quarterly Journal of Economics 3(87), 355–374 (1973)
32. Stigler, G.: An Introduction to Privacy in Economics and Politics. The Journal of Legal Studies 4(9), 623–644 (1980)
33. Telang, R., Wattal, S.: An empirical analysis of the impact of software vulnerability announcements on firm stock price. IEEE Transactions on Software Engineering 33(8), 544–557 (2007)
34. Van Huyck, J., Battallio, R., Beil, R.: Tacit coordination games, strategic uncertainty, and coordination failure. American Economic Review 80(1), 234–248 (1990)
35. Varian, H.R.: System reliability and free riding. In: Camp, L.J., Lewis, S. (eds.) Economics of Information Security. Advances in Information Security, vol. 12, pp. 1–15. Kluwer Academic Publishers, Dordrecht (2004)

36. von Auer, L.: Revealed preferences in intertemporal decision making. Theory and Decision 56(3), 269–290 (2004)
37. Wellman, M., Wurman, P., O'Malley, K., Bangera, R., Lin, S., Reeves, D., Walsh, W.: Designing the market game for a trading agent competition. IEEE Internet Computing 5(2), 43–51 (2001)
38. Xu, X.: Group size and the private supply of a best-shot public good. European Journal of Political Economy 17(4), 897–904 (2001)

A Self-insurance Considerations

In this section, we briefly revisit the prior analysis by considering the ways in which self-insurance further decreases the protection likelihood for expert players. Because some of the derivations required in the complete analysis are especially cumbersome, especially in the weakest link game, we shall resort to a high level overview of the situation.

A.1 Self-insurance in the Best Shot Game

In the best shot game, self-insurance is easy to address. Self-insurance is only a spoiler for a self-protection equilibrium when it is cheaper than protection ($c < b$). In this event, all selfish expert players (and even the naïve players) would defect to the self-insurance strategy to improve their own payoff. On the other hand, cooperative experts could still work together to protect the network as long as $b < cN$ (and $\sum_{j=1}^{N} L_j \geq b$). If $b < c$, then no player will choose the self-insurance strategy because it is cheaper to protect for the same individual result.

A.2 Self-insurance in the Weakest Link Game

How could the existence of self-insurance spoil the weakest-link protection equilibrium? For a short but not entirely simple answer, the expert i will defect to the self-insurance strategy if $L_i \geq \frac{c-b}{1-Prot^*_{\neg i}}$ where $Prot^*_{\neg i}$ is the probability that (under the current value of cost parameters) all the players other than i will protect. Unfortunately, unless the number of other experts is zero, the value of $Prot^*_{\neg i}$ is not amenable toward a closed form formula involving b, c, N and k. Because self-insurance introduces an *upper bound* on expected losses, a symmetric equilibrium strategy in which the network has a chance of protection requires expert i to protect iff $\alpha \leq L_i \leq \beta$, for some parameters α and β which both depend on b, c, k, and N. Even when $N = k$, determining α and β requires solving the following parametrized system of equations for α and β:

$$\frac{b}{L(\beta - \alpha)^{N-1}} = \alpha \tag{4}$$

$$\frac{c - b}{L\left(1 - (\beta - \alpha)^{N-1}\right)} = \beta. \tag{5}$$

We can do this, and derive some relations between parameters, but in the end of this process the resulting inequality conditions do not yield substantial insights beyond what

is already obvious from a high-level view – namely that availability of self-insurance can be a serious distractor for a protection equilibrium outcome in the weakest link game. We already deduced from the restricted game without self-insurance, that for any protection equilibria to exist, the protection costs must be very small, on the order of a constant times $\frac{1}{N}$. The presence of self-insurance exacerbates the problem, ruining the chances of there being a protection equilibrium even for low values of b and moderately high values of c. (For a simple example, even if b is as low as $\frac{1}{eN}$, self-insurance can serve as a deterrent to protection investment with values of c as high as $\frac{2}{3}$).

A.3 Self-insurance in the Total Effort Game

In the total effort game, conditions under which the presence of the self-insurance alternative spoils a protection equilibrium are simple to express (although the derivation of these conditions is nontrivial). If $c < bN$, then there is no protection equilibrium, because an expert for whom it is worthwhile to protect must have a loss L_i exceeding bN. Under such conditions, one can derive that it is more advantageous for the expert to pay the self-insurance premium c. On the other hand, if $c > bN$, then the equilibrium strategy defined previously in which expert i protects if and only if $L_i \geq bN$ continues to be a Pareto-dominant Bayesian Nash equilibrium.

A.4 Overall Effects of Self-insurance

The results in this section indicate that the existence of self-insurance strategies can be a deterrent toward protection investments, especially in the weakest link game. Indeed the existence of self-insurance only contributes further to the overarching theme of our analysis – that the number of experts does not improve the security of the network when those agents are acting in their own best interest.

RatFish: A File Sharing Protocol Provably Secure against Rational Users

Michael Backes[1,2], Oana Ciobotaru[1], and Anton Krohmer[1]

[1] Saarland University
[2] MPI-SWS

Abstract. The proliferation of P2P computing has recently been propelled by popular applications, most notably file sharing protocols such as BitTorrent. These protocols provide remarkable efficiency and scalability, as well as adaptivity to dynamic situations. However, none of them is secure against attacks from rational users, i.e., users that misuse the protocol if doing so increases their benefits (e.g., reduces download time or amount of upload capacity).

We propose a rigorous model of rational file sharing for both seeders and leechers, building upon the concept of rational cryptography. We design a novel file sharing protocol called RatFish, and we formally prove that no rational party has an incentive to deviate from RatFish; i.e., RatFish constitutes a Nash equilibrium. Compared to existing file sharing protocols such as BitTorrent, the tracker in RatFish is assigned additional tasks while the communication overhead of a RatFish client is kept to a minimum. We demonstrate by means of a prototype implementation that RatFish is practical and efficient.

1 Introduction

Recently, the peer-to-peer (P2P) paradigm has emerged as a decentralized way to share data and services among a network of loosely connected nodes. Characteristics such as failure resilience, scalability and adaptivity to dynamic situations have popularized P2P networks in both academia and industry. The proliferation of P2P computing has also been propelled by popular applications, most notably file sharing protocols such as BitTorrent [5].

A crucial assumption underlying the design of such file sharing protocols is that users follow the protocol as specified; i.e., they do not try to bypass the design choices in order to achieve higher download rates, or to avoid uploading to the system at all. However, not all users are necessarily altruistic, and publicly available, modified BitTorrent clients like BitThief [19] or BitTyrant [21] can be used to strategically exploit BitTorrent's design to achieve a higher download while contributing less or nothing at all in return. While several minor protocol adaptations have been suggested to mitigate the attacks underlying these clients [28], the core weaknesses remain: In its current form, BitTorrent – and current file sharing protocols in general – offer better service to cheating clients, thereby creating incentives for users to deviate from the protocol; in turn, it

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 607–625, 2010.

further decreases the performance of honest clients. The task is thus to design a protocol that not only retains the remarkable characteristics of current file sharing protocols, but that is rational in the sense that it offers sufficient incentives for users to stick to the precise protocol specification. In more technical terms, this file sharing protocol should constitute an equilibrium state: Adhering to the protocol should optimize the benefits received by each individual participant, and any deviation from the protocol should result in a lower payoff for the cheating user.

1.1 Our Contribution

We contribute RatFish, a protocol for rational file sharing. RatFish is built upon the concepts and design choices that underlie BitTorrent, but it resolves the weaknesses that BitThief and BitTyrant exploit. We achieve this mostly by ensuring fair exchange of pieces between leechers and by having the tracker participate in the coordination of the downloads.

The distinctive feature of RatFish, however, is not that it discourages the use of several selfish strategies, but that it comes with a formal proof that deviating from RatFish is irrational for both seeders and leechers. In order to do this, we first characterize rational behaviors of leechers and seeders in file sharing protocols, building upon the concept of the recently emerging field of rational cryptography, in which users are defined as rational players in a game-theoretic sense. Intuitively, leechers are primarily interested in minimizing their download time and the amount of uploaded data, whereas seeders value the efficiency of the protocol in using their upload capacity. We cast this intuition into a rigorous mathematical model, and we formally prove that our protocol is secure against deviations of rational parties, by showing that it constitutes a Nash equilibrium. This holds even though RatFish allows dynamically leaving and (re-)joining users. We prove this Nash equilibrium using a new proof technique that is of independent interest for rational cryptography: the step-by-step substitution of a deviating strategy with hybrid, semi-rational strategies.

We have built a prototype implementation of RatFish that demonstrates that RatFish is practical and efficient. We stress though that the purpose of RatFish is not to achieve performance improvements over existing protocols, but to establish a formal proof that under realistic conditions, such as dynamically joining users, no rationally-behaving user has an incentive to deviate from RatFish. The additional computational overhead of RatFish compared to BitTorrent is small: basic cryptographic primitives (symmetric encryptions and digital signatures schemes) are used, and the tracker is assigned additional tasks such as the coordination of downloads and the generation of user incentives. The communication overhead of a RatFish client is kept to a minimum.

1.2 Related Work

The performance of BitTorrent has been thoroughly studied [24,3,11,23,22]. All these works attest to the impressive performance of BitTorrent in the presence

of honest participants; however, [3] has noted that the rate-based Tit-For-Tat policy of BitTorrent does not prevent nodes from uploading less content than they should serve (in all fairness), thereby creating an incentive for abuse of the protocol.

The behavior of BitTorrent in the presence of cheating peers was subsequently investigated [18,21,19,26], revealing that cheating leads to a loss in overall performance for honest peers.

Our rigorous model of rational file sharing is grounded in the recently emerging field of rational cryptography, where users are assumed to only deviate from a protocol if doing so offers them an advantage. Rational cryptography is centered around (adapted) notions of game theory such as computational equilibria [6]. A comprehensive line of work already exists that develops novel protocols for important cryptographic primitives such as rational secret sharing and rational secure multiparty computation [7,10,9,1,14,8].

In this spirit, there has been a variety of research aimed at making BitTorrent more robust against deviations of rationally-behaving users [28,15,22,20,27]. All these works provide stronger user incentives: they choke problematic connections [28], grant additional bandwidth to generously uploading neighbors [15], reward leechers that continue seeding after their download is completed [22], optimally distribute a seeder's bandwidth across swarms [20], and employ fair exchange protocols to stop leechers from aborting the protocol [27] early. These modified protocols, however, are still prone to rational attacks; in particular, none of these works reached a (Nash) equilibrium.[1]

The first work that strived to establish an equilibrium in the context of file sharing is [24]. However, this equilibrium was severely restricted in that it was only guaranteed when rational parties were allowed to only tweak the protocol parameters, but not when they could deviate in larger ways.

More recent research such as BAR-B [2], Equicast [13], and FOX [25] aimed at deploying incentives and punishments such that obeying the protocol is the best strategy for every rational player. The first two protocols were shown to be strict Nash equilibria, i.e., a rational peer obtains no benefit from unilaterally deviating from the assigned strategy. The drawback is that their strict equilibrium solutions limit the design: the BAR-B system only permits a static set of users. Equicast requires the rate at which leechers join to precisely match the rate of which they leave and considers only restricted utility functions that do not take downloading time into account; moreover, these protocols require nodes to waste network bandwidth by sending garbage data to balance bandwidth consumption. [25] establishes a stable Nash equilibrium, but again it only allows a static set of leechers; moreover, its rationality is not grounded on incentives but on fear of retaliation such that a single Byzantine node can cause the entire system to collapse. Somewhat orthogonal to our work are the file streaming applications

[1] [27] contains proofs that certain deviations from selfish leechers or attacks of malicious peers cannot succeed (e.g., no peer can assume another authorized peer's identity), but there is no equilibirum proof, i.e., that deviating from the protocol cannot yield better results.

BAR-Gossip [17] and FlightPath [16]. Both works show a Nash equilibrium (a strict one for BAR-GOSSIP, and an approximate one for Flightpath), but rational players are only interested in minimizing the amount of uploaded data and reducing jitter. While such time-independent utility functions are reasonable for streaming applications, they do not apply to the more sophisticated setting of rational file sharing, where minimizing the time to complete a download is usually the primary goal. Moreover, none of these five protocols considers seeders as part of the rational model. We conclude by saying that like our approach, none of these works offers resistance against Sybil attacks.[2]

1.3 Outline

Section 2 provides a bird's eye view of the core ideas underlying how we create incentives in file sharing. Section 3 summarizes the concepts we use from rational cryptography and defines rational behaviors of seeders and leechers. Section 4 presents the RatFish protocol in detail. Section 5 contains the proof of equilibrium for RatFish; i.e., it shows that users cannot achieve a better payoff by deviating from the protocol. Section 6 discusses our experimental results. Section 7 concludes.

2 A Bird's Eye View on How to Rationalize P2P

For the sake of exposition, we provide a high-level overview of the core ideas underlying how we create incentives in file sharing. We briefly discuss which behaviors of seeders and leechers we consider rational, intuitively explain how to incentivize these behaviors, and finally discuss how an equilibrium is obtained for a small example protocol. In this section, we abstract away many important systems details and impose several assumptions to improve understanding.

In the following, we consider a single seeder S that intends to upload a file f to leechers $L_1, \ldots, L_M$. The file is split into pieces $f_1, \ldots, f_{M^2}$. In this exposition, we describe a simplistic protocol that proceeds in a sequence of $M+1$ monolithic rounds. We assume that the seeder can upload exactly M pieces per round and that every leecher is able to upload and to download at least M pieces of the file in each round.

On the Rationality of Seeding. A seeder is a player that uploads without requesting reciprocation. Intuitively, it thus acts rationally if it uses its upload time and upload speed as efficiently as possible; i.e., for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. It is thus in the interest of the seeder to incentivize leechers to share parts of the file amongst each other as this

[2] A Nash equilibrium ensures that no individual user has an incentive to deviate. However, it conceptually does not take coalitions of users into account, rendering Sybil attacks possible in most works on rationally secure protocols.

increases the throughput of the whole system.[3] In the simplistic protocol, the seeder sends to each leecher L_i in each round j the piece $f_{j \cdot M+i}$.

On the Rationality of Leechers. Leechers aim to download the file as fast as possible while saving upload capacity. The protocol thus has to enforce leecher participation as they will otherwise just download and leave. We need to propose a suitable piece selection algorithm and a piece exchange mechanism that prevents parties from cheating each other. In our example, piece selection is easy: In each round j a leecher L_i holds a piece $f_{j \cdot M+i}$ obtained from the seeder that no one else has. As the leecher can upload M pieces per round, he can exchange with the rest of the leechers their unique pieces. To ensure fair exchanges, leechers first exchange the pieces in encrypted form and subsequently send the corresponding decryption keys.

How an Equilibrium is Achieved. We conclude with some basic intuition on why no rational user has an incentive to deviate from the protocol. If all peers adhere to the protocol, the seeder will upload the file exactly once and stay in the system for M rounds. Each of the leechers will upload $M^2 - M$ pieces and complete its download after $M + 1$ rounds. It is easy to see that the average download time and hence the seeder's utility cannot be improved.

This outcome cannot be further improved for the leechers either: None of the leechers can download the file in less than $M + 1$ rounds since after round M each of them is missing at least $M - 1$ pieces.[4] Moreover, since the seeder only provides M pieces to each of its peers, no leecher can obtain the full file without uploading at least $M^2 - M$ pieces in exchange for the pieces that it is missing from the seeder. This statement holds as no leecher can cheat during the fair piece exchange protocol: A leecher spends his upload capacity to receive an encrypted piece, hence he has no incentive not to send the much smaller decryption key to its partner. Thus, no party can improve its utility by deviating from the protocol.

3 A Game-Theoretic Model for File Sharing

In this section, we propose a game-theoretic model for rationally secure file sharing. We start by reviewing central concepts from game theory and rational cryptography.

[3] As a consequence, the naive approach of uploading the whole file to an arbitrary leecher at once cannot yield a rationally secure protocol: This leecher may just complete the download and leave, causing some pieces of the file to be effectively lost from the system. Moreover, since there is only one seeder in this simplistic protocol and the number of leechers is known and does not change, there is no need for a third party, i.e., a tracker.

[4] This holds because the protocol treats the rounds integrally. Otherwise, we could split a sufficiently big file into M^K pieces for some K and achieve a slightly reduced, optimal download time of $M + \frac{M^2}{M^K}$ using an analog algorithm.

3.1 Review of Game-Theoretic Definitions

A *game* $\Gamma = (\{A_i\}_{i=1}^n, \{u_i\}_{i=1}^n)$ consists of *players* $1, \ldots, n$ where each of them has a set A_i of possible *actions* to play and individual *utility functions* u_i. All actions are played simultaneously; afterwards, every player i receives a *payoff* that is determined by applying its utility function u_i to the vector of actions played in the game.

Recent work has extended the traditional notion of a game to the requirements of cryptographic settings with their probabilistically generated actions and computationally-bounded running times. The resulting definition – called *computational game* [12] – allows each player i to decide on a probabilistic polynomial-time (in the security parameter) interactive Turing machine M_i (short PPITM). The machine M_i is called the *strategy* for player i. The output of M_i in the joint execution of these interactive Turing machines denotes the action of player i.

Definition 1 (Computational Game). *Let k be the security parameter and $\Gamma = (\{A_i\}_{i=1}^n, \{u_i\}_{i=1}^n)$ a game. Then Γ is a* computational game *if the played action A_i of each participant i is computed by a PPITM M_i and if the utility u_i of each player i is polynomial-time computable.*

Because of the probabilistic strategies, the utility functions u_i now correspond to the expected payoffs. Rationally behaving players aim to maximize these payoffs. In particular, if a player knew which strategies the remaining players intend to choose, he would hence pick the strategy that induces the most beneficial outcome of the game. As this simultaneously holds for every player, we are looking for a so-called *Nash equilibrium*, i.e., a strategy vector where each player has no incentive to deviate from, provided that the remaining strategies do not change. Similar to the notion of a game, we consider a computational variant of a Nash equilibrium.

Definition 2 (Computational Nash Equilibrium). *Let $\Gamma = (\{A_i\}_{i=1}^n, \{u_i\}_{i=1}^n)$ be a computational game and k the security parameter. A strategy vector consisting of PPITMs $\boldsymbol{M} = (M_1, \ldots, M_n)$ is a* computational Nash equilibrium *if for all i and any PPITM M_i' there exists a negligible function ϵ such that $u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, \boldsymbol{M}) \leq \epsilon(k)$ holds.*

Here $u_i(k, M_i', \boldsymbol{M}_{-i})$ denotes the function u_i applied to the setting where every player $j \neq i$ sticks to its designated strategy M_j and only player i deviates by choosing the strategy M_i'.

We finally define the *outcome* of a computational game as the transcript of all players' inputs and the actions each has taken. In contrast to strategy vectors, an outcome thus constitutes a finished game where every player can determine its payoff directly. A utility function is thus naturally defined on the outcome of a computational game: When applied to a strategy vector (with its probabilistic choices), it describes the vector's expected payoff; when applied to an outcome of the game, it describes the exact payoff for this outcome.

3.2 A Game-Theoretic Model for File Sharing Protocols

We now define the utility functions for seeders and leechers such that these functions characterize rational behavior in a file sharing protocol. We start by introducing common notation and some preliminaries.

Notation and Preliminaries. Following the BitTorrent convention, we call a player in the file sharing game a *peer*. The peers are divided into two groups: A *seeder* uploads to other peers a *file* f that it owns, whereas a *leecher* downloads f. To mediate the communication among peers, we require a trusted party called the *tracker*. The tracker holds a signing key pair (pk, sk), and we assume that its IP address and public key pk are known to all peers.

The file f consists of pieces $f_1, \ldots, f_N$, each of length B bytes. The participants in the file sharing protocol hold the values $h_1 = h(f_1), \ldots, h_N = h(f_N)$, where h is a publicly known *hash function*. When deployed in practice, this publicly known information is distributed via a *metainfo file*. The tracker is only responsible for coordinating peers that are exchanging the same file. In order to stay close to a realistic setting, we allow different peers to have different upload and download capacities. Every seeder S_i has its individual *upload speed* $up_i^s(t, o)$ that depends on the time t and the outcome o. Note that a seeder does not download anything except for metadata; hence we do not consider the download speed of seeders in this paper. Similarly, every leecher L_i has individual *upload* and *download speeds* $up_i^l(t, o)$ and $down_i^l(t, o)$. We denote by $T_{i,\mathrm{fin}}(o)$ the total time that leecher L_i spends downloading the file. To increase readability, we omit the outcome in all formulas whenever it is clear from the context. We also introduce the sets $L = \{i \mid L_i \text{ is a leecher}\}$ and $S = \{i \mid S_i \text{ is a seeder}\}$.

Rationally-behaving Seeders. A seeder uploads parts of the file to other peers without requesting reciprocation. Intuitively, a seeder is interested in using as efficiently as possible its upload time and upload speed. Thus for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. We express this preference by the following seeder's utility function.

Definition 3 (Seeder's Utility Function). *We say that u_i^s is a* utility function for a seeder S_i *if for any two outcomes o, o' of the game with the same fixed upload speed up_i^s and fixed time T_i^s spent by S_i in the system, it holds that $u_i(o) \geq u_i(o')$ if and only if $\frac{1}{|L|} \sum_{i \in L} T_{i,\mathrm{fin}}(o) \leq \frac{1}{|L|} \sum_{i \in L} T_{i,\mathrm{fin}}(o')$.*

If S_i is the first seeder in the system, we thus require that S_i uploads the whole file at least once. Otherwise, it is not rational to share the file in the first place.

Rationally-behaving Leechers. Leechers aim at downloading the shared file as fast as possible; moreover, they also try to use as little upload capacity as possible. The relative weight of these two (typically contradictory) goals is given by a parameter α_i in the system measuring time units per capacity units, e.g., seconds per kilobytes.

Definition 4 (Leecher's Utility Function). *Let $\alpha_i \geq 0$ be a fixed value. We say that u_i^l is a* utility function for leecher L_i *if the following condition holds: For two outcomes o, o', L_i prefers outcome o to o' if and only if*

$$\alpha_i \cdot \int_0^{T_{i,\mathrm{fin}}(o)} up_i^l(t, o)\, dt + T_{i,\mathrm{fin}}(o) \leq \alpha_i \cdot \int_0^{T_{i,\mathrm{fin}}(o')} up_i^l(t, o')\, dt + T_{i,\mathrm{fin}}(o').$$

The value α_i corresponds to L_i's individual valuation for upload speed and time; e.g., if $\alpha_i = 0.5$, the leecher values time twice as much as the uploaded data.

In particular, this definition implies that a rationally-behaving leecher prioritizes completing the download over everything else: If the leecher does not download the file in outcome o, then $T_{i,\mathrm{fin}}(o)$ equals infinity. If it downloads the file in some outcome o', then $T_{i,\mathrm{fin}}(o')$ is finite and thus increases its payoff.

4 The RatFish Protocol

We now present the RatFish protocol. We start with the description of the tracker and proceed with the seeders and leechers, respectively.

4.1 The Protocol of the Tracker

Similar to BitTorrent, our tracker manages all valid IP addresses in the system and introduces new leechers to a set of neighbors. However, we assign the tracker additional tasks: First, our tracker is responsible for awarding each newcomer with seeding capacity equivalent to γ file pieces (for a tunable parameter γ).[5] Second, our tracker keeps track of which file pieces each peer owns at any given moment. This bookkeeping will be crucial for incentivizing peers to follow the RatFish protocol, for computing the deserved rewards and for answering queries about the leechers' availabilities. Third, the tracker imposes a forced wait for every leecher upon connecting, thereby preventing leechers from gaining advantages by frequently disconnecting and rejoining the protocol. Finally, if a leecher wishes to disconnect, the tracker provides a certificate on the most recent set of pieces the leecher has to offer. This allows leechers to later reconnect to RatFish and use their partially downloaded data, i.e., in order to cope with network disruptions. In the following, we describe the individual subprotocols of the tracker in detail. A rigorous description is given in Fig. 1.

The Connect Protocol. The tracker assigns every new leecher L_i a random subset of size H of all leechers that are currently in the system. This random subset corresponds to L_i's local neighborhood. The tracker sends this neighborhood information to L_i after T seconds. Once the forced wait is over, the leecher may start downloading γ free pieces from seeders. The rationale behind

[5] In practice, γ is a small constant number just big enough for the new leecher to participate in the system. As long as the seeders can provide γ pieces to each newly joining leecher, this value does not influence the existence of the Nash equilibrium.

TrackerConnect($peer$)
If $peer$ is a seeder S_i, receive the seeder's upload speed up_i^s and store it. Else, do:

- If a message "PIECES$(a_1, \ldots, a_N, id_r, sig_{t'})$" is received from L_i, verify that $sig_{t'}$ is a valid signature on $(a_1, \ldots, a_N, id_r)$ for verification key pk and that $p = (a_1, \ldots, a_N, id_r)$ was previously stored. In this case, remove p from storage and set $A_i^m := a_m$ for all $m \in \{1, \ldots, N\}$. Otherwise select a random id_r.
- As soon as the current time T_c is larger than $T + T_p$, where T_p is the connecting time of the leecher, i.e., after the forced wait of T seconds, send L_i a random subset of size H of current leechers' IP addresses, corresponding to L_i's neighborhood. Moreover, compute $\mathsf{S}_{sk}(i, T_p)$, yielding a signature sig_t. Send TIME(T_p, id_r, sig_t) to L_i.

CheckExchange
Do the following steps unless one of their checks fail; abort in this case:

- Upon receiving a message HAS(j, y) from a leecher L_i, send back yes if $A_j^y = 1$, and no otherwise.
- Upon receiving a message EXCHANGED(j, x, y) from a leecher L_i, indicating that the pieces x and y have been exchanged with L_j, check $A_i^x = 1$ and $A_j^y = 1$. Send the message ACKNOWLEDGE(i, x, y) to L_j.
- Upon subsequently receiving the message OK(i, x, y) from L_j, set $X_i := X_i + 1$ and $A_i^y := 1$, and send the message OK(j, x, y) to L_i.

RewardLeechers (called every T seconds, i.e., at the start of a new round)

- Award γ pieces to every leecher that joined in the previous round. Let $prev$ be the number of these leechers.
- Compute for every leecher L_i its deserved percentage of seeders' upload speed:

$$r_i := \min\left\{\frac{X_i}{\sum_{k \in L} X_k}, \frac{1}{2}\right\}$$

- Let $\beta_i := r_i \cdot \left(\frac{\sum_{k \in S} up_k^s \cdot T}{B} - \gamma \cdot prev\right)$. For every i, determine a set of seeders that can jointly offer β_i new pieces to L_i such that the individual upload capacity of the seeders is respected, see Sect. 4.1. Send to every seeder the corresponding set of leechers and the number of pieces that these leechers should download from them.
- Set $A_j^y := 1$ if a seeder already informed the tracker that y was downloaded by S_j. Set $X_i := 0$ for all i.

PeerLeave(i)

- Compute $sig_p := \mathsf{S}_{sk}(A_i^1, \ldots, A_i^N, id_r)$. Store $(A_i^1, \ldots, A_i^N, id_r)$.
- Send the message LEAVE(sig_p) to L_i and disconnect from L_i.

Fig. 1. The protocol of the tracker. The variable A_i used in these protocols represents an array that stores which file pieces L_i has already downloaded. The algorithm ensures that $A_i^m = 1$ if leecher L_i has the m-th piece of f, and $A_i^m = 0$ otherwise.

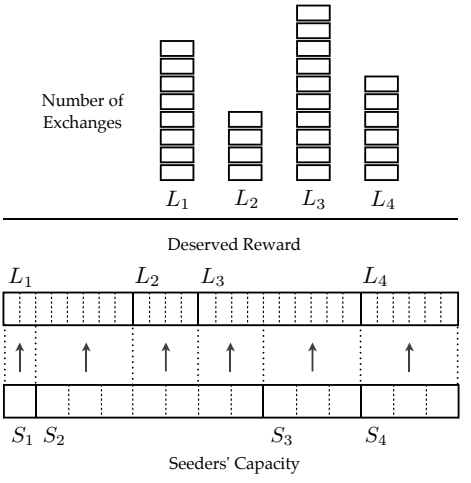


Fig. 2. Schematic distribution of the rewards

this forced wait is that granting newly joined leechers free pieces creates incentives for whitewashing, i.e., frequent disconnecting and rejoining. Intuitively, the forced wait is a simple defense against such a behavior. From a rational perspective, if a leecher joins the system only once, the induced small delay will not be hurtful; however, whitewashing by frequently rejoining will cause an accumulated delay that will result in a smaller payoff. The forced wait is achieved by having the tracker sign the leecher's connecting time and IP address. Such signed timestamps are exchanged between neighboors and are used to determine whether leechers are allowed to start uploading to eachother. Neighbors use the timestamps to determine whether they are allowed to start uploading to each other. Thus as long as a user's IP address does not change, it can idle and become active again without being penalized by a forced wait, since the user's old signature on its IP address and time is still valid.

The RatFish tracker has a mechanism for proving availability of rejoining leechers: it chooses a random rejoin ID id_{r} and signs it together with the departing leecher's availability. The leecher uses id_{r} to prove its availability upon rejoining the system. The rejoining id_{r} is then deleted from the tracker's memory preventing leechers from reconnecting twice using the same reconnect id_{r}.

The Reward Protocol. The reward system constitutes the crucial part of RatFish. The underlying idea is to reward only leechers who are exchanging. We only allow one exception to this rule: The leechers that connected to the tracker in the previous round are also entitled to a reward of γ pieces in the current round. Thus the seeders do not automatically upload to their neighborhood as in BitTorrent; rather they are told by the tracker whom to upload to.

To determine whether an exchange between L_i and L_j has happened, the tracker asks both L_i and L_j to acknowledge the exchange. If the acknowledgements succeed, the tracker internally increases the variables X_i and X_j, which

Seeding$_j$
Upon connecting, the seeder sends its upload speed up_j^s to the tracker. In each round, do:

- Receive from the tracker a set M of leechers and the corresponding number of pieces ω_i that every leecher $L_i \in M$ should receive.
- Inform every leecher $L_i \in M$ how many pieces ω_i they are allowed to download.
- When a leecher $L_i \in M$ request at most ω_i pieces by L_i (potentially incrementally in this round, i.e., it may ask for a few pieces first), send these pieces to L_i and send a message to the tracker that these pieces have been uploaded to L_i. Requests by leechers $L_j \notin M$ are ignored.

Fig. 3. The protocol of the seeder S_j.

corresponds to the number of file piece exchanges of L_i and L_j, respectively. The tracker moreover stores which pieces of the file the leechers now additionally know. Details on the participation of the tracker in the exchange protocol are given in Sect. 4.3, where the exchange of pieces between leechers is explained.

Every round, i.e., after T seconds, the actual rewards are given out. The tracker distributes the seeders' upstream in proportion to the number of exchanges every leecher made. Hence, the more exchanges a leecher completed, the larger the reward it obtains from the tracker, and hence the more download capacity it receives from the seeders. A graphical illustration of the reward protocol is given in Fig. 2.

4.2 The Protocol of the Seeder

Upon connecting, the seeder informs the tracker about the upload speed it is going to offer. The tracker adds the seeder's speed to the amount of free available upload capacity. As the tracker performs all the computations for determining the rewards, the seeder simply proceeds by uploading the number of file pieces to the leechers as instructed by the tracker. To keep the tracker's information about individual leechers up-to-date, the seeder informs the tracker whenever it uploads a piece to a leecher. A rigorous description is given in Fig. 3.

4.3 The Protocol of the Leecher

From a rational perspective, the leecher protocol is the most difficult to get right: while the tracker is honest and seeders partially altruistic, a leecher tries to bypass the incentives for uploading wherever reasonable.

Leechers can use the signed messages from the tracker to verify each other's join times. Also, when two leechers exchange data, the tracker participates in this exchange: Before two leechers start an exchange, they verify with the tracker that the other party holds the desired piece. If this check succeeds, two encryptions of

the pieces are exchanged. Before they also send the key to each other to decrypt these messages, both leechers acknowledge the exchange to each other so that they get a higher reward.

The Connect Protocol. When a leecher connects to the tracker for the first time it requests a local neighborhood. If the leecher rejoins, it additionally proves to the tracker that it already owns some pieces of the file by sending the signature received from the tracker at its last disconnect. When connecting to a

$\mathsf{LeecherConnect}_i(party)$
If *party* is the tracker, distinguish two cases:

1. If L_i rejoins the protocol, send $\mathsf{PIECES}(a_1, \ldots, a_N, id_\mathrm{r}, sig_p)$ to the tracker where $a_m = 1$ if L_i owns the m-th piece of the file, id_r is the rejoin ID and sig_p is the signature received when disconnecting from system last time. If L_i is a new leecher, it sends $\mathsf{PIECES}(0, \ldots, 0, \epsilon, \epsilon)$.
2. Receive $\mathsf{TIME}(T_p, id_\mathrm{r}, sig_t)$ from the tracker – indicating the signed connecting time and ID, as well as a set of neighbors' IP addresses. Connect to them.

If *party* is a leecher L_j, do (abort if a check fails):

- Send the message $\mathsf{MYTIME}(T_p, sig_t)$ to L_j.
- Receive the message $\mathsf{MYTIME}(T'_p, sig'_t)$ from L_j. Verify that sig'_t is a valid signature on (j, T'_p) for pk and that $T_c > T'_p + T$ holds.
- Send $\mathsf{AVAILABILITY}(a_1, \ldots, a_N)$ to L_j where $a_m = 1$ if L_i owns f_m.
- Receive the message $\mathsf{AVAILABILITY}(a'_1, \ldots, a'_N)$ from L_j.

$\mathsf{LeecherAwarded}$
Whenever L_i is informed by a seeder S_j that it can download ω_i pieces, request up to ω_i pieces from S_j (potentially incrementally in this round, i.e., L_i may ask for a few pieces first), and download these pieces.

$\mathsf{Exchange}_i(f_x, j, y)$
If any of the following checks fails, blacklist L_j and abort.

- Send the message $\mathsf{HAS}(j, y)$ to the tracker and wait for a positive answer.
- Choose a random key $k_{j,x}$ and compute $c_{j,x} \leftarrow \mathsf{E}(k_{j,x}, f_x)$.
- Send $c_{j,x}$ to L_j and wait for c_y from L_j.
- Perform the following two steps in parallel and proceed once both steps are completed:
  - Send $\mathsf{EXCHANGED}(j, x, y)$ to the tracker and wait for $\mathsf{OK}(j, x, y)$ as response
  - If receiving $\mathsf{ACKNOWLEDGE}(j, y, x)$ from the tracker, reply with $\mathsf{OK}(j, y, x)$.
- Send the key $k_{j,x}$ to L_j.
- Upon receiving k_y from L_j, retrieve $f'_y \leftarrow \mathsf{D}(k_y, c_y)$ and verify $h_y = h(f'_y)$.
- Broadcast to the local neighborhood that you now own the piece y.

Fig. 4. The protocol of the leecher L_i

seeder, the leecher requests pieces until its seeder's reward is depleted. Upon contacting another leecher, it waits until both forced waits are over. Afterwards, both parties exchange information such that they know which pieces they can request from each other. To keep track of the availability in its neighborhood, the leecher observes the messages that leechers broadcast to their local neighborhood, indicating which pieces of the file they have just downloaded.

The Piece Exchange. The piece exchange protocol run between two leechers uses encryptions to ensure that no leecher can get a piece without completing the exchange phase. From a practical perspective, it is important to note that the key sizes are small compared to a file piece size. Thus the communication and storage overhead induced by the keys and cryptographic operations is kept manageable. Leechers additionally query the tracker to ensure that the corresponding party owns a file piece they need. Moreover, leechers want their exchanges to be counted and rewarded. Thus, after the encryptions are exchanged, each leecher prompts the tracker to ask the other leecher for an acknowledgement. Intuitively, there is no incentive to deviate in this step as they still lack the key from the other party. Once the acknowledgement step is successfully completed, both leechers exchange the keys. If one leecher deviates from this procedure, it is blacklisted by the other leecher. We stress that blacklisting is not required for the security proof; it solely constitutes a common technique in this setting to deal with malicious parties. A rigorous description is given in Fig. 4. Fair exchange protocols have been used in prior work to incentivize peers to fairly exchange information [27]. In contrast to [27], however, RatFish needs to neither periodically renew cryptographic keys nor implement a non-repudiable complaint mechanism to allow parties to prove possible misbehaviors; instead it relies on short acknowledgment messages for each recipient and on collecting these messages to monitor the file completion for the participants. A schematic overview of the core part of the piece exchange protocol is provided in Fig. 5.

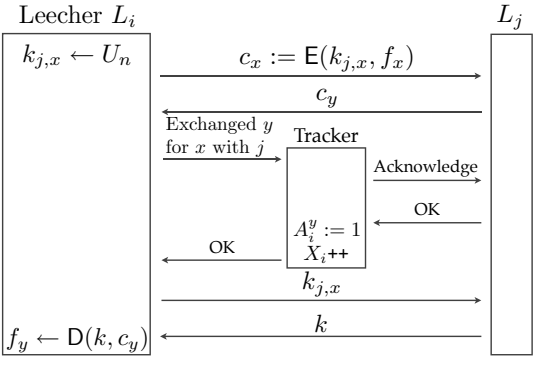


Fig. 5. The core part of the piece exchange protocol between two leechers

5 Equilibrium Proof

In this section we prove that RatFish yields a computational Nash equilibrium; i.e., no leecher or seeder has an incentive to deviate from the protocol.

5.1 Underlying Assumptions

Recall that RatFish proceeds in rounds of T seconds. For simplicity, we assume that peers can join / leave only at the beginning / end of a round[6] and that it is impossible to forge identities on the IP layer (e.g., by using appropriate authentication mechanisms). We assume that at least one seeder is present and that the overall seeding capacity does not exceed twice the overall upload capacity of the leechers; this bound on the seeding capacity prevents the leechers from free riding, which is easy given enough seeding power. We moreover assume that each leecher's dedicated upload speed up_i^l is fully exhausted by other peers. These assumptions seem reasonable as the average seeders/leechers ratio is often close to 1:1 [4], and optimized centralized coordinators are capable of distributing upload capacities among different swarms [20]. Additionally, we assume keys do not contribute to the uploaded amount, since in practice, the size of the short keys is dominated by the size of the encrypted file piece. Moreover, we assume that each peer is only able to maintain one identity in the system. This in particular excludes Sybil attacks, in which multiple distinct identities are created by the same peer to subvert the reputation system of a P2P network. This assumption does not come as a surprise, since the Nash equilibrium conceptually does not defend against coalitions, rendering Sybil attacks possible in most works on rationally secure protocols. (See Section 7 for an outlook on how to tackle this problem.) Regarding the cryptographic primitives, we assume that the signature scheme used by the tracker is secure against existential forgery under chosen-message attack and that the encryption scheme is semantically secure under chosen-plaintext attack.

5.2 Proving the Nash Equilibrium

We finally show that RatFish constitutes a Nash equilibrium. Due to space constraints, we concentrate on illustrating how the proof is conducted and on highlighting the novel proof technique that was used. The technical parts of the proof are given in the full version.

We first show that a leecher deviating from the protocol cannot increase its utility by more than at most a negligible value, provided that no other party deviates. To show this, we determine two sets of possible *cheating actions* for leechers, which we call *independent actions* and *correlated actions*. Intuitively, the independent cheating actions can be *individually* replaced by honest actions

[6] This assumption can be easily enforced by letting the tracker force joining peers to wait until the next round.

without decreasing the utility, independent of the remaining parts of the deviating strategy. Correlated cheating actions are sensitive to the details of the deviating strategy: we can only replace a correlated cheating action by a corresponding honest action without decreasing the utility if all deviating actions that jointly influence the leecher's utility are *simultaneously* replaced in one round. We show that the only correlated cheating action is to not acknowledge an exchange.

Our proof technique starts with an arbitrary deviating strategy M_i' and provides a proof in two steps: In the first step, we replace all independent cheating actions step-by-step; here, a step within a strategy denotes the computation performed within the strategy between two consecutive outputs. Slightly more formally, let M_i be the honest strategy for leecher L_i, M_i' a deviating strategy, and $\{H^*_{ack,j}\}_j$ the set of all strategies that in every step are honest or do not acknowledge an exchange. Then our technique yields a so-called *semi-honest* strategy $M_i^* \in \{H^*_{ack,j}\}_j$ that for every input and random tape outputs in every step the same action as M_i' whenever possible, and plays honest otherwise. We then show that the semi-honest strategy cannot yield a worse payoff than M_i'. The proof is based on the novel concept of hybrid concatenation of strategies.

Lemma 1 (No Independent Cheating Actions of Leechers). *Let γ be the number of uploaded pieces a newly joined leecher is awarded. Let M_i' be a deviating strategy of L_i and let M_i^* be the semi-rational strategy as defined above. Then for $\alpha_i \in [0, \frac{T}{3 \cdot \gamma \cdot B}]$, we have $u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, M_i^*, \boldsymbol{M}_{-i}) \leq \epsilon(k)$ for some negligible function ϵ.*

Thus far, we have transformed a deviating strategy M_i' into a semi-rational strategy M_i^* that uses only correlated cheating actions and does not decrease the payoff. In the second step, we replace all correlated cheating actions round-by-round until we reach the honest strategy M_i. We use a hybrid argument based on the hybrid concatenation of strategies to show that the honest strategy outperforms the semi-rational strategy for leechers.

Lemma 2 (No Correlated Cheating Actions of Leechers). *Let M_i be the honest strategy for L_i, i.e., following the RatFish protocol and let M_i^* be the semi-rational strategy as defined above. Then $u_i(k, M_i^*, \boldsymbol{M}_{-i}) - u_i(k, M_i, \boldsymbol{M}_{-i}) \leq \epsilon(k)$ holds for some negligible function ϵ.*

Showing that seeders have no incentive to deviate from the protocol is considerably easier than the corresponding statement for leechers, since seeders are considered partially altruistic. We show that as long as all leechers follow the protocol, a seeder cannot run a deviating strategy to improve its payoff.

Lemma 3 (No Seeder Deviation). *There is no deviating strategy for any seeder that increases its payoff if all other parties follow the RatFish protocol.*

We finally combine the results that neither leechers nor seeders have an incentive to deviate (the tracker is honest by assumption) to establish our main result.

Theorem 1 (Computational Nash Equilibrium). *The RatFish protocol constitutes a computational Nash equilibrium if $\alpha_i \in [0, \frac{T}{3 \cdot \gamma \cdot B}]$ holds for all $i \in L$.*

6 Implementation and Performance Evaluation

In this section, we describe the implementation of RatFish and experimentally evaluate its performance. We focus on the implementation and performance evaluation of the tracker, since the tracker took on several additional responsibilities and is now involved in every exchange. In contrast to the tracker, seeders and leechers are largely unchanged when compared to BitTorrent: the exchange of encrypted pieces constitutes a small computational overhead, but leaves the network complexity that usually constitutes the performance bottleneck of P2P protocols essentially unchanged.

6.1 Implementation

The RatFish tracker was implemented using about 5000 lines of code in Java, thus ensuring compatibility with common operating systems. The implementation is designed to work with both UDP and TCP.

The messages sent in the protocol start with the protocol version number and message ID (which determines the length of the message), followed by the torrent ID, and additional information that depends on the type of message.

Afterwards, a task is created that processes the received message. This task is given to the threadpool executor – the main part of the RatFish tracker that also ensures parallelization. The threadpool sustains eight parallel threads and assigns new tasks to the next free thread. For instance, when the tracker receives a notification that a leecher joined the protocol, the task extracts the leecher's IP from this message and triggers the forced wait. After T seconds it replies with a digital signature for the leecher using an RSA-based signature scheme that signs SHA-1 hashes.

6.2 Experimental Setup

For the evaluation, we ran the RatFish tracker on a server with a 2-cores Intel Xeon CPU, 2GB of RAM, a 100MBit Internet connection and an Ubuntu SMP operating system with kernel version 2.6.28-18. We simulated a swarm with up to 50,000 peers, divided into neighborhoods of size 4. The simulated leechers send the same messages as a real leecher would, thus yielding an accurate workload measure for the tracker. Every leecher was configured to complete one exchange per second, and we chose the size of a piece to be 256 kB according to BitTorrent standards. Hence every leecher has a virtual upload speed of 256 kB/s. The size of the shared file is 50 MB, and the seeders upload one forth of the file per second in a round-robin fashion to their neighborhoods. The simulated clients are running on a separate machine. This allows us to measure network throughput. In our simulation, we need to pretend to the tracker that clients connect from different IPs. We thus used UDP in our experiments. Deploying RatFish in reality would presumably be based on TCP, which would slightly increase the network complexity.

6.3 Performance Evaluations

Fig. 6 depicts the results for our experiments. The main observation, shown in the left part of Fig. 6, is that even though we engage the tracker in every exchange in the swarm, the protocol scales well (a resource usage of 65% for 50,000 leechers). One can also observe that the computation complexity becomes a limiting factor, but we expect this to change for more cores given our highly parallelized implementation. Memory was not a bottleneck in any experiment.

The right part of Fig. 6 considers the case where many leechers join at once, but no exchanges are happening. This study is important since the tracker's most expensive task is to sign the join time of leechers. In our experiments, the tracker was able to serve about 400 new leechers per second. Since the server has T seconds for signing in practical deployments, the signature computation would be scheduled in free CPU time and hence not delay ongoing exchanges. We also observed that the two measurements depicted in Fig. 6 on CPU usage are additive, e.g., a system with 30,000 leechers and 200 joining leechers per second uses approximately 90% of the CPU.

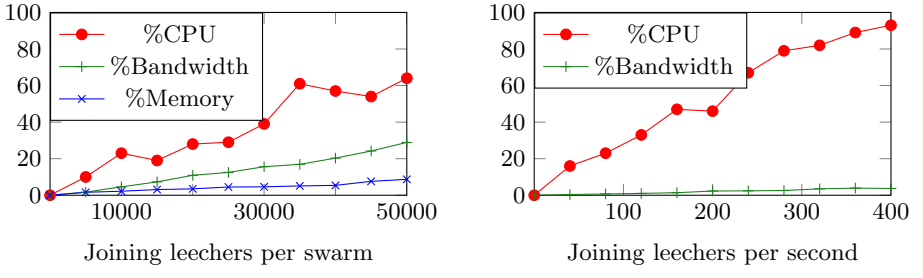


Fig. 6. Left: Resource usage for a static number of leechers engaging in exchanges. **Right:** Resource usage for dynamically joining leechers.

7 Conclusions and Future Work

We have proposed a file sharing protocol called RatFish that we have proven secure against deviations of rational users. We first characterized rational behaviors of leechers and seeders in file sharing protocols. Subsequently, we formally showed that no rational party has an incentive to deviate from RatFish; i.e., RatFish constitutes a Nash equilibrium. While the tracker in RatFish is assigned additional tasks compared to existing file sharing protocols such as BitTorrent, the communication overhead of a RatFish client compared to a BitTorrent client is minimal. We have demonstrated by means of a prototype implementation that RatFish is practical and efficient.

A central question for future work on rational file sharing – and for rational cryptography in general – is whether the Nash equilibrium is a strong enough notion for real-world applications and threat models. Robustness against user

coalitions would be more desirable. (See the discussion in [7] and [1].) RatFish already provides suitable hooks for potential mitigation techniques against coalitions, e.g., by ensuring that players entering small coalitions can only increase their utilities by a negligible amount; hence entering a coalition would be irrational in the first place. Moreover, RatFish currently considers file sharing for independent swarms only, i.e., seeders in one swarm cannot be leechers in another swarm. Extending RatFish to cope with such more a general setting requires to generalize the seeders' utility functions and to adapt the relevant parts of RatFish in order to maintain the Nash equilibrium property.

References

1. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006), pp. 53–62 (2006)
2. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: BAR fault tolerance for cooperative services. Operating Systems Review (OSR) 39(5), 45–58 (2005)
3. Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Analyzing and improving a BitTorrent network's performance mechanisms. In: The 25th IEEE Conference on Computer Communications, INFOCOM 2006 (2005)
4. Bieber, J., Kenney, M., Torre, N., Cox, L.P.: An empirical study of seeders in BitTorrent. Tech. rep., Duke University (2006)
5. Cohen, B.: Incentives build robustness in BitTorrent. Tech. rep., bittorrent.org (2003)
6. Dodis, Y., Halevi, S., Rabin, T.: A cryptographic solution to a game theoretic problem. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 112–130. Springer, Heidelberg (2000)
7. Feigenbaum, J., Shenker, S.: Distributed algorithmic mechanism design: recent results and future directions. In: 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M 2002), pp. 1–13 (2002)
8. Fuchsbauer, G., Katz, J., Naccache, D.: Efficient rational secret sharing in standard communication networks. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 419–436. Springer, Heidelberg (2010)
9. Gordon, D., Katz, J.: Rational secret sharing, revisited. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 229–241. Springer, Heidelberg (2006)
10. Halpern, J., Teague, V.: Rational secret sharing and multiparty computation: extended abstract. In: STOC 2004, pp. 623–632 (2004)
11. Izal, M., Uroy-Keller, G., Biersack, E., Felber, P.A., Hamra, A.A., Garces-Erice, L.: Dissecting BitTorrent: Five months in torrent's lifetime. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 1–11. Springer, Heidelberg (2004)
12. Katz, J.: Bridging game theory and cryptography: Recent results and future directions. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 251–272. Springer, Heidelberg (2008)
13. Keidar, I., Melamed, R., Orda, A.: Equicast: Scalable multicast with selfish users. Computer Networks 53(13), 2373–2386 (2009)

14. Kol, G., Naor, M.: Cryptography and game theory: Designing protocols for exchanging information. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 320–339. Springer, Heidelberg (2008)
15. Levin, D., LaCurts, K., Spring, N., Bhattacharjee, B.: BitTorrent is an auction: analyzing and improving BitTorrent's incentives. Computer Communications Review (CCR) 38(4), 243–254 (2008)
16. Li, H.C., Clement, A., Marchetti, M., Kapritsos, M., Robison, L., Alvisi, L., Dahlin, M.: FlightPath: Obedience vs. choice in cooperative services. In: USENIX OSDI 2008, pp. 355–368 (2008)
17. Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: BAR gossip. In: USENIX OSDI 2006, pp. 191–204 (2006)
18. Liogkas, N., Nelson, R., Kohler, E., Zhang, L.: Exploiting BitTorrent for fun (not for profit) (2006), http://iptps06.cs.ucsb.edu/
19. Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free riding in BitTorrent is cheap. In: 5th Workshop on Hot Topics in Networks (HotNets-V), pp. 85–90 (2006)
20. Peterson, R.S., Sirer, E.G.: Antfarm: efficient content distribution with managed swarms. In: USENIX NSDI 2009, pp. 107–122 (2009)
21. Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A.: Do incentives build robustness in BitTorrent? In: USENIX NSDI 2007, pp. 1–14 (2007)
22. Piatek, M., Isdal, T., Krishnamurthy, A., Anderson, T.: One hop reputations for peer to peer file sharing workloads. In: USENIX NSDI 2008, pp. 1–14 (2008)
23. Pouwelse, J.A., Garbacki, P., Epema, D., Sips, H.J.: The BitTorrent p2p file-sharing system: Measurements and analysis. In: Castro, M., van Renesse, R. (eds.) IPTPS 2005. LNCS, vol. 3640, pp. 205–216. Springer, Heidelberg (2005)
24. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In: SIGCOMM 2004, pp. 367–378 (2004)
25. Rob, D.L., Sherwood, R., Bhattacharjee, B.: Fair file swarming with FOX (2006), http://iptps06.cs.ucsb.edu/
26. Shneidman, J., Parkes, D., Massoulie, L.: Faithfulness in internet algorithms. In: Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS 2004), pp. 220–227 (2004)
27. Sirivianos, M., Yang, X., Jarecki, S.: Robust and efficient incentives for cooperative content distribution. Transactions On Networking (TON) 17(6), 1766–1779 (2009)
28. Thommes, R., Coates, M.: BitTorrent fairness: Analysis and improvements. In: 4th IEEE Workshop on the Internet, Telecommunications and Signal Processing, WITSP 2005 (2005)

A Service Dependency Model for Cost-Sensitive Intrusion Response

Nizar Kheir[1,2], Nora Cuppens-Boulahia[1],
Frédéric Cuppens[1], and Hervé Debar[3]

[1] Télécom Bretagne, 2 rue de la Chataigneraie, 35512 Cesson Sévigné, France
{nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
[2] France Télécom R&D, 42 Rue des Coutures, 14066 Caen, France
nizar.kheir@orange-ftgroup.com
[3] Télécom SudParis, 9 rue Charles Fourier, 91011 Evry, France
herve.debar@telecom-sudparis.eu

Abstract. Recent advances in intrusion detection and prevention have brought promising solutions to enhance IT security. Despite these efforts, the battle with cyber attackers has reached a deadlock. While attackers always try to unveil new vulnerabilities, security experts are bounded to keep their softwares compliant with the latest updates. Intrusion response systems are thus relegated to a second rank because no one trusts them to modify system configuration during runtime.

Current response cost evaluation techniques do not cover all impact aspects, favoring availability over confidentiality and integrity. They do not profit from the findings in intrusion prevention which led to powerful models including vulnerability graphs, exploit graphs, etc. This paper bridges the gap between these models and service dependency models that are used for response evaluation. It proposes a new service dependency representation that enables intrusion and response impact evaluation. The outcome is a service dependency model and a complete methodology to use this model in order to evaluate intrusion and response costs. The latter covers response collateral damages and positive response effects as they reduce intrusion costs.

1 Introduction

The dot-com bubble that occurred in the late nineteen nineties has changed the nature of market places, maybe for ever. These have become very dynamic, pushing IT industries to propose innovative software tools in order to attract new customers. IT industries started suffering increasing challenges, among which are shorter product life cycles, rapid outsourcing and price erosion caused by a fierce worldwide competition [19]. In order to withstand to these challenges, products' security had been relegated to a secondary priority that is handled as an add-on property [18]. Security flaws are more likely to be tolerated in newly released products. IT industries become inclined to reveal new updates and patches on a regular basis in order to strengthen the security of their products. The opposite coin facet for this reality is a vicious race between security experts and cyber

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 626–642, 2010.

attackers. By the time a new security flaw is discovered, thousands of cyber attacks are being reported before a suitable patch is released. For instance, an article from the techworld magazine discusses the effect of one security flaw for the Microsoft Internet Explorer browser. After just a couple of weeks this security flaw is disclosed, and before the next scheduled security updates are revealed, more than thirty thousand daily attacks have been reported [17]. In such a teased environment, security experts cannot only rely on the security of their products when they are constantly updated with new patches. Indeed, they must be equipped with powerful prevention and monitoring tools that help to prevent security breaches when they occur, and detect these before they succeed.

Setting and keeping security equipments require increasing investments that constitute a heavy burden on the shoulders of small and medium companies. Nonetheless, these equipments are often bypassed by skilled attackers, but also script-kiddies, proving to be insufficient when used as a single line of defense. As a consequence, intrusion response systems have started to play a growing role in today's security architectures [24]. Response systems are unable to prevent the threat before it occurs, but they react as it occurs in order to prevent its direct effects. They modify configuration setups in order to contain an attack or prevent its success. Unfortunately, trivial responses that use static mappings between intrusive patterns and available responses do not provide a reliable solution. Attackers rapidly learn about those mappings and adapt their attacks in consequence. This has promoted the need for more advanced response systems that implement more sophisticated strategies, including security policies [8].

Although response systems have reached a high level of sophistication [24], security experts are still reluctant to use those systems due to the potential of damages they may provoke. In fact, current systems are increasingly growing in complexity. They experience growing trends towards providing more interactive services to support every user's need in terms of quality of service. Service providers are thus constrained to use granular and interdependent service architectures which yield a better agility for service configuration. However, the aftermath of one security breach could be drastic because impacts rapidly spread through service dependencies [10]. Besides, cyber attacks are becoming more sophisticated. Internet provides an exceptional facility to conduct collaborative attacks or to use the excessively available botnets [20]. As attacks are getting extremely complicated, they require accurate and severe responses in order to be contained or blocked. The decision to take these responses is more likely to be taken by a security expert in order to avoid boomerang effects as the self-inflicted denial of service. We believe that the lack of a comprehensive approach to represent service dependencies is a major reason for not using those dependencies to support decisions for intrusion response. This is a major limitation because service dependencies provide a well-suited platform to compare intrusion responses and to select cost-sensitive responses [12] [25]. One major contribution of this paper is thus to develop a new service dependency representation that is used to evaluate and compare intrusion and response impacts in order to select optimal responses, i.e. responses that inflict less impact to the system.

When inspecting the literature of the domain, we find that dependencies in their broadest sense have been longly used for intrusion response. We will thus precise what makes this contribution so different. In fact, dependencies are mostly structured into dependency graphs that embed either *logical* or *functional* dependencies. In the former category, we cite the examples of privilege graphs, attack graphs and vulnerability graphs. Privilege graphs [7] [6] trace the attacker privilege escalations in target systems. Dependencies point-out privileges that enable an attacker to achieve a security objective. Attack graphs [23] [3] specify causal relations between potential attacker actions. Finally, vulnerability graphs [2] [11] describe the access that is required for an attacker in order to carry out an attack. These approaches offer to assess intrusion costs by statically assigning quantitative cost metrics to nodes in these graphs. Intrusion costs are evaluated as the aggregation of elementary costs for the already achieved steps in the graph. Meanwhile, and unless they rely on expert knowledge, no formal approach to evaluate elementary costs is yet provided. This is due to the fact that critical services and their dependencies are not represented in these graphs. Moreover, service dependencies are dynamic and may be modified by intrusion responses, which modifies the static elementary costs in these graphs. Another limitation for using these graphs without further extensions is the inability to assess response costs. In fact, only *positive* response costs may be evaluated, i.e. intrusion costs that are prevented by a response. To the best of our knowledge, no formal approach to evaluate response collateral damages can be applied.

On the other hand, functional dependencies are dependencies between system resources [4] [10] [12] [13] [26]. They represent the need for a dependent resource to use an antecedent resource in order to be fully operational. Functional dependency graphs propagate impacts as applied in system dependability management. Although they are more appropriate to evaluate intrusion and response impacts, these graphs suffer multiple limitations. In fact, intrusion impacts are often propagated downwards in these graphs, i.e. from an antecedent resource towards a dependent resource. They are also used to evaluate only response *negative* impacts, i.e. the response collateral damages. However, they are less likely to be used in order to evaluate response positive impacts.

This paper is motivated by the limitations of both existing approaches. It rather bridges the gap between them, by representing both security objectives in logical dependency graphs and resource dynamic dependencies in functional dependency graphs. It provides a new service dependency representation and implements intrusions and responses using the same semantics as for service dependencies. A simulation platform is defined, which simulates intrusion impacts, response impacts and the combined impacts for intrusion and response. Measures are further aggregated in order to select the most cost-effective response. This paper is structured as follows. Section 2 defines a new response index in order to compare and select intrusion responses. Section 3 implements the attributes that constitute this index using service dependencies. Section 4 presents the simulation platform that is used to compute those attributes. Section 5 demonstrates the use of this platform using a real-world example and section 6 concludes.

2 Return-On-Response-Investment index

The fundamental question an intrusion response system must answer is whether the self-inflicted response cost is reasonably tolerated when reacting against an intrusion attempt. In fact, this question challenges every aspect of IT security, that is whether a security investment is justified to avoid a security threat. Finance books provide multiple indicators that answer this question, among which the Return-On-Investment (ROI) index [21]. The ROI index, written as the ratio of net benefit to costs, compares multiple investment alternatives. It is used during risk analysis and to decide about investments in intrusion prevention [1]. The yet Return-On-Security-Investment (ROSI) index is used to promote investments in preventive IT security measures. It is defined in [1] as:

$$\texttt{ROSI} = \frac{(\textit{Expected Losses} - \textit{Residual Losses}) - \textit{Investment Costs}}{\textit{Investment Costs}}$$

Among multiple security investments, the security expert picks-up the one that satisfies a *maximal positive* ROSI index. Based on these facts, this paper proposes a new decision making process for intrusion response that is rather based on a financial comparison of response alternatives. We consider that a system often specifies some security objectives that are directly expressed in terms of monetary losses when they are not satisfied. Intrusions and responses inflict some costs when they affect these security objectives. These costs are classified into three components: response goodness (RG), response collateral damages (CD) and response operational costs (OC) [25]. RG measures the response ability to reduce the costs inflicted by the intrusion attempt. By analogy to the ROSI index, RG is compared to the prevented losses due to security investments. CD is the cost that is added by a newly enacted response, and that is not related to intrusion costs. It is inherent to the response mechanism as the latter affects some security objectives. OC is independent from the security objectives of the system. It includes response setup and deployment costs, such as manpower and over provisioning. By analogy to the ROSI index, investment costs are compared to the aggregation of CD and OC. We adapt the ROSI index to the response process, and thus we obtain the Return-On-Response-Investment (RORI) index, as follows.

$$\texttt{RORI} = \frac{\texttt{RG} - (\texttt{CD} + \texttt{OC})}{\texttt{CD} + \texttt{OC}}$$

To specify the response goodness index, we define the following cost metrics. $\texttt{IC}_b$ represents expected intrusion impacts when no response is enacted. $\texttt{IC}_a$ represents expected intrusion impacts after some response is enacted. $\texttt{IC}_a$ is difficult to evaluate because it is almost impossible to discern intrusion and response costs when they are applied simultaneously. We thus propose the index RC to represent the combined impact for both intrusion and response. Based on these metrics, we develop the expression of the RORI index as follows.

$$\texttt{RORI} = \frac{(\texttt{IC}_b - \texttt{IC}_a) - (\texttt{CD} + \texttt{OC})}{\texttt{CD} + \texttt{OC}} = \frac{\texttt{IC}_b - (\texttt{IC}_a + \texttt{CD}) - \texttt{OC}}{\texttt{CD} + \texttt{OC}} = \frac{[\texttt{IC}_b - \texttt{RC}] - \texttt{OC}}{\texttt{CD} + \texttt{OC}}$$

The OC metric is not related to the system security objectives. It is statically defined as part of a risk analysis plan prior to system runtime. The three remaining metrics, i.e. IC_b, RC and CD, are evaluated online as soon as new intrusions are detected and new candidate responses are proposed. These metrics, for the same intrusion and response combinations, depend on current service configuration. In following sections, we propose a complete methodology to evaluate those metrics using service dependencies. The ultimate goal is to select the candidate response set that provides a maximal positive RORI index, if any.

3 Service Dependency Framework

3.1 Specification of System Security Objectives

We denote security objectives as the set of security guidelines that must be satisfied. These guidelines are specified within security policies that characterize users and their access permissions. Access permissions are sometimes explicitly granted to users; they constitute explicit privileges, e.g. all employees are granted personal laptops. IT systems also implement service architectures where users interact with the system in order to obtain additional privileges. Users are thus less likely to be granted explicit privileges, but only credentials that enable them to interact with the system, e.g. PKI certificates. Users belong to equivalence classes $\mathcal{C}u_i\rfloor_{i=1}^{n}$ where they enjoy the same access permissions. An equivalence class is compared to a role where all users have the same privileges. A user may thus belong to more than one equivalence class. We use equivalence classes and user privileges in order to define security objectives, as follows: The IT system must guarantee the secure user access to the privileges that are relevant within his/her equivalence class. The secure access covers privilege availability, i.e. user ability to acquire this privilege, but also privilege misuse, i.e. to be acquired by an unintended user. Failing to do so implies some security objectives to be unsatisfied. The resulting impacts are manifested as availability impacts, or confidentiality and integrity impacts following a privilege misuse. Before we evaluate those impacts, we first formalize the conditions under which they occur.

We assign a privilege p to an equivalence class $\mathcal{C}u_i$ using the predicate $assign$. Assigning a privilege to an equivalence class means that users of this class require access to this privilege. We express this statement as follows: $assign(p, \mathcal{C}u_i) \Leftrightarrow \forall u \in \mathcal{C}u_i, requires(u, p)$. We note that assigning a privilege to an equivalence class does not mean users are explicitly granted this privilege. It only means that a security objective is unsatisfied when users cannot acquire this privilege. We model the granting of a privilege to a user with the predicate $granted$, which implies the following statement: $assign(p, \mathcal{C}u_i) \not\Rightarrow \forall u \in \mathcal{C}u_i, granted(p, u)$. It follows up that revoking a privilege to a user within an equivalence class provokes the failure of a security objective only when this privilege is assigned to this class. We propose the $revoked$ predicate to represent this statement. It is defined as: $revoked(p, \mathcal{C}u_i) \Leftrightarrow \exists u \in \mathcal{C}u_i, assign(p, \mathcal{C}u_i), \neg granted(p, u)$.

Definition: we define an availability failure every combination of one privilege p and one equivalence class $\mathcal{C}u_i$ that satisfies the predicate $revoked(p, \mathcal{C}u_i)$.

We also define the condition for a privilege misuse, that is a privilege to be granted to an unintended user. We use the predicate $infected$ that is defined as: $infected(p, \mathcal{C}u_i) \Leftrightarrow \exists u \notin \mathcal{C}u_i : assign(p, \mathcal{C}u_i), \neg requires(u, p), granted\ (p, u)$. The outcome of a privilege infection is the granting of inappropriate permissions to a user, which may provoke integrity or confidentiality impacts, according to the permissions that are associated to this privilege.

Definition: we define a confidentiality or integrity failure every combination of one privilege p and one equivalence class $\mathcal{C}u_i$ that satisfies the predicate $infected(p, \mathcal{C}u_i)$.

We propose to use privilege infection and revocation in order to quantify costs for intrusion and response, i.e. to evaluate the metrics $\mathtt{IC}_b$, RC and CD that constitute the RORI index. Meanwhile, intrusions and responses target either users or system services. They infect and/or revoke privileges on either user-side or system-side. Impacts further propagate through service dependencies until they affect end-users. Precisely one needs to know how users interact with system services, and how dependencies influence the impact propagation process.

3.2 Privilege Sharing and Service Dependencies

Trust relationship: As far as users are only granted credentials, they interact with the system services in order to acquire the privileges they are assigned. Privileges are initially held by services and shared with users in counterpart to *trusted* credentials and privileges. We introduce trust relationships as part of an authorization scheme by which we specify the way privileges are shared between users and services. Trust relationships do not only apply to user credentials, but also to privileges. In fact, some services may evolve in a trusted environment, e.g. a shared repository service accessible via an Intranet connection. As a consequence, users who have the privilege of 'being connected to the Intranet' are granted the permission to 'upload data to this service'. We define a trust relationship tr using the predicate $trust(tr)$. tr has two attributes: (1) The *trustee* specifies a privilege or credential $priv_2$ trusted by the service $subj_1$ that implements tr (i.e. $implement(subj_1, tr)$). (2) The *grantee* specifies a privilege $priv_1$ that is granted by the service $subj_1$ when the trusted privilege or credential is used by a subject $subj_2$. We formalize the notion of trust as follows.

$$granted(priv_1, subj_2) \leftarrow trust(tr), implement(subj_1, tr), grantee(tr, priv_1), \\ trustee(tr, priv_2), granted(priv_2, subj_2)$$

Trust relationships are implemented in order to set and configure service dependencies. They enable access control as they restrain access to an antecedent service to the only service that is granted the trusted credentials and privileges.

Service dependencies are made explicit by a request to an antecedent service. The Role-based Trust-management (RT) framework in [16] represents the request concept as a delegation process by which the requester delegates some privileges to its request. The RT framework applies to role management and delegation.

For instance, that some subject Ea requests an authorization which belongs to the role Rb from Eb with its capacity of being empowered in the role Ra is represented as: $Ea \xrightarrow{\texttt{Ea as Ra}} Eb.Rb$. Service dependencies comply to the same request specification. A dependent service uses credentials and privileges to satisfy trust relationships implemented by an antecedent service. The role concept in the RT framework is treated as a collection of permissions [22], which makes it compatible with the privilege concept for service dependencies. We use the dot notation '.' to represent the fact that a subject $subj$ is granted a privilege or credential $priv$. It is defined as: $subj.priv \Leftrightarrow granted(priv, subj)$. We introduce a service dependency with the statement: $dep \xrightarrow{dep.priv_1} ant.priv_2$. It states that the dependent subject dep uses the privilege $priv_1$ in order to support its request through which it requires the privilege $priv_2$ from the antecedent subject ant.

Example: A web server has its root directories hosted by a network file system (NFS) service. Access to NFS service is controlled using the `/etc/exports` file where IP addresses are registered. The request statement is modeled as:

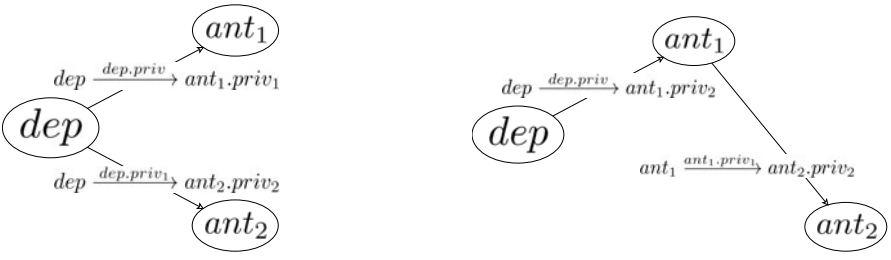
$$\texttt{Web} \xrightarrow{\texttt{Web IP in /etc/exports}} \texttt{NFS.permission(access, Root directories)}$$

Dependency satisfaction constraint: A dependency is satisfied after the dependent service obtains the required privileges. The outcome is expressed as the fact that the dependent service shares some privileges with the antecedent service. We infer, using the definitions of dependency and trust, the condition for a dependency to be satisfied. A dependency is satisfied if, and only if, the dependent service uses the credentials and privileges that are trusted by the antecedent service. The antecedent service trusts a privilege if it implements trust relationships that map between this privilege and the privilege requested by the dependent service. We formalize these concepts as follows.

$$(dep \xrightarrow{dep.priv_1,..,dep.priv_n} ant.priv_o \rightarrow granted(priv_o, dep)) \Leftrightarrow (\forall tr : (implement(ant, tr) \wedge grantee(tr, priv_o)), \exists i(trustee(tr, priv_i)))$$

Dependency compositions occur when multiple dependencies contribute to providing the same privilege. Dependency compositions include two types of elementary patterns, which are logical and functional compositions. We use the same request statement to express these patterns, as in figure 1. In a logical composition (Fig. 1a), the dependent service cannot satisfy its second dependency for the service ant_2 until the former dependency for ant_1 has been satisfied. In a functional composition (Fig. 1b), the dependent service satisfies its unique dependency when the antecedent service ant_1 satisfies its own dependency for service ant_2. Dependency compositions constitute elementary patterns through which intrusion and response impacts propagate as discussed in section 3.3.

Example: In the example of the web and NFS services, the web service provides applications for Intranet users. We have two composition patterns. In fact, only authenticated users access the web service; the latter cannot answer requests unless it accesses the NFS service. We model this example as follows:



(a) Logical composition **(b)** Functional composition

Fig. 1. Dependency composition

```
                valid Login/Password
User ──────────────────────────────► Auth.permission(connect, Intranet)
                Intranet connection
User ──────────────────────────────► Web.permission(access, web applications)
                Web IP in /etc/exports
Web  ──────────────────────────────► NFS.permission(access, Root directories)
```

3.3 Intrusions, Responses and Impact Propagations

This section uses service privileges in order to introduce the impact of intrusions and responses. It relies on existing methods to represent IT attacks, namely attack graphs. It rather bridges the gap between attack graphs and service dependency models using the dependency representation described in section 3.2.

Intrusion costs: It is shown in [5] that the impact of an attack on a target component can be fully characterized using pre/post-condition statements. Attack pre-conditions define the state of the target system prior to an attack success. Post-conditions define the system state after an attack succeeds. From a service dependency perspective, that means an attacker should have enough privileges to access a vulnerability and thus to conduct his attack. The outcome of this attack is that some privileges are either *revoked* to the target component or *infected* by the attacker. The National Vulnerability Database[1] provides a similar classification of vulnerabilities. It associates to a vulnerability (1) an access vector that indicates the requirements that enable exploiting a vulnerability and (2) an impact vector that indicates the post-attack effects on the target service.

We introduce a vulnerability v using the predicate $vulnerability(v, subj)$ where $subj$ is the vulnerable subject. v is assigned three properties: $infects(v, priv)$, $revokes(v, priv)$ and $access(v, priv)$. The first property specifies privileges $priv$ that are infected when v is exploited by an attacker. The second property specifies privileges that are revoked to the vulnerable service when v is exploited. The third property specifies attacker privileges that are required to access the vulnerability v. We use the predicate $vulnerability$ to propose a privilege-based definition of attacks. We use the same request statement as for service dependencies. Meanwhile,

[1] http://nvd.nist.gov/

the attack success criteria are different since the attacker uses an illicit access path. We define an attack and its success criteria as follows:

Listing 1.1. Attack success criteria

$$(att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \rightarrow granted(priv_o, att)) \Leftrightarrow (\exists v : (vulnerability(v, tgt), infects(v, priv_o)), \neg(access(v, q), \forall i(q \neq priv_i)))$$

$$(att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \rightarrow revoked(priv_o, tgt)) \Leftrightarrow (\exists v : (vulnerability(v, tgt), revokes(v, priv_o)), \neg(access(v, q), \forall i(q \neq priv_i)))$$

Privileges that are granted to an attacker are also infected privileges, we may thus infer the statement $infected(priv_o, tgt)$. An intrusion impact further propagates through service dependencies because the attacker abuses of trust relationships implemented to satisfy those dependencies. Impacts either propagate upwards or downwards as illustrated by the following propagation patterns.

Listing 1.2. Impact propagation patterns

$$infected(priv_o, tgt) \wedge \exists(ant, priv_a) : tgt \xrightarrow{tgt.priv_o} ant.priv_a \Rightarrow infected(priv_a, tgt)$$

$$infected(priv_o, tgt) \wedge \exists(dep, priv_a) : dep \xrightarrow{dep.priv_a} tgt.priv_o \Rightarrow infected(priv_o, dep)$$

$$revoked(priv_o, tgt) \wedge \exists(dep, priv_a) : dep \xrightarrow{dep.priv_a} tgt.priv_o \Rightarrow revoked(priv_o, dep)$$

$$revoked(priv_o, tgt) \wedge \exists(ant, priv_a) : tgt \xrightarrow{tgt.priv_o} ant.priv_a \Rightarrow revoked(priv_a, tgt)$$

We add intrusions to the model and we propagate their impacts using the propagation patterns in listing 1.2 in order to evaluate intrusion costs. These costs are a direct consequence to the intrusion impacts in terms of *revoked* and *infected* privileges, as in section 3.1. We may use existing approaches to convert service failures into costs, as in [15] [25]. We evaluate the expected intrusion costs when no response is enacted, i.e. the $\texttt{IC}_b$ metric in the `RORI` expression. We still need to evaluate the `RC` and `CD` metrics, which requires to represent intrusion responses.

Response representation: We model intrusion responses using the same approach that we used to model intrusions. We shall point out two differences between intrusions and responses, and how we handle them in our model. A response is first a decision that is deliberately taken by the system. The latter degrades some security objectives in order to react against an ongoing threat. We thus dispose of the attacker notation (Att) in the response representation. Besides, a response does not infect privileges as intrusions do, at least for conventional responses we consider in this paper (e.g. quarantine host, set firewall rule, block port, stop service, block account). They only render a service more vulnerable to an attack. Privilege infections that may occur if this service is further attacked are not a direct consequence to this response. An attack is still required in order for these infections to take place. We thus dispose of privilege infections (i.e. $infects$ predicate) in the response representation.

A response grants and/or revokes some privileges to a target subject. It is expected to prevent the attack or to contain its impacts. An intrusion is prevented when the response revokes some privileges used by the attacker in order to access the target vulnerability. Intrusion impacts ($\texttt{IC}_a$) are thus reduced to null. Intrusion prevention analysis is not a particularity to our model, it is already handled using attack graphs and anti-correlation techniques [5]. Meanwhile, the containment of attack impacts is difficult to handle using only attack graphs. In fact, we separate between attack containment and attack impact containment. In the former, we shall prevent the attacker from using the privileges he acquired in order to conduct a new attack step. Attack containment is possible using techniques based on attack graphs and does not require excessive knowledge about service dependencies [9]. Meanwhile, attack impact containment requires interleaving with service dependencies in order to prevent the attacker from realizing any benefit when he/she uses the infected privileges. It prevents impact propagations through service dependencies as presented in the previous paragraph.

We introduce the predicate $response(resp, tgt)$ to model the enforcement of a response $resp$ on a target resource tgt. We represent the impact of response on the target resource as: $response(resp, tgt) \Rightarrow \exists priv : granted(priv, tgt) \vee revoked(priv, tgt)$. We note that this definition applies to one elementary response. A comprehensive response scenario against an ongoing attack includes multiple elementary responses that are modeled each using the predicate $response$. These responses interfere with intrusion impact propagations, by either increasing or -*hopefully* - decreasing those impacts. We shall evaluate response impacts in order to infer the RC and CD metrics.

Intrusion prevention is modeled using the following statement:

$$att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \wedge \exists i(revoked(priv_i, att)) \Rightarrow \neg(infected(priv_o, tgt) \vee revoked(priv_o, tgt))$$

This response expression is a direct consequence to the attack success criteria in listing 1.1. These may no longer be satisfied because the attacker is revoked from privileges that he needs in order to access the vulnerable service. A targeted response like this does not impact the system security objectives because it only affects the malicious user. Although it often constitutes an ideal case, it could be impossible because of multiple reasons. In fact, an attacker may be unknown (e.g. IP spoofing), which constrains the system to select target-centric responses. An intrusion scenario may also be detected at mid-point to its ultimate goal. The known attacker would be only a stepping stone to the real remote attacker. Attacker-centric responses would thus apply to some system component and not to the real attacker. Besides, the attacker-centric response may be excluded because the system may not have enough capability to do so. Therefore, and by the time a privilege is revoked to some system component, impacts may further propagate as used for intrusion impacts.

Intrusion impact containment does not deal with the direct causes of an intrusion, but limits its impacts when they occur. Intrusion impacts are actually

prevented from propagating through service dependency paths. Responses interleave with these paths in order to stop impact propagations. This paragraph discusses propagation patterns in listing 1.2 and shows how responses interleave with these patterns. We use the example of the web-NFS dependency in section 3.2 in order to illustrate each of these patterns.

First propagation pattern represents upward infection propagation. Infected privileges for a dependent service are misused in order to infect privileges to an antecedent service. Impact propagation is contained by denying access to the antecedent service ($revoked(priv_o, tgt)$) or by quarantining (i.e. revoking) the threatened privileges for the antecedent service ($revoked(priv_a, ant)$). The counterpart of this response (CD metric) is to initiate new impact propagations that are described by the third and fourth propagation patterns in listing 1.2.

Example: An attacker conducts a buffer overflow against the web server, which enables him to execute arbitrary code using the web server permissions, including its IP address (`infected (web IP in /etc/exports, web)`). Upward propagation affects the NFS server, i.e. `infected(access root directories, web)` (please refer to the first pattern in listing 1.2). Responses revoke access to root directories for the NFS server (`revoked(access root directories, NFS)`) or deny web access to the NFS server (`revoked(web IP in /etc/exports, web)`).

Second propagation pattern represents downward infection propagation. Infected privileges for an antecedent service remain infected when they are shared with its dependent services. In fact, infected privileges are granted to the attacker. He actually uses these privileges wherever the service configuration (i.e. dependency) enables to do so. Downward propagation is prevented, as for upward propagation, by disabling the threatened dependency. It either requires to revoke infected privileges to the target resource, ($revoked(priv_o, tgt)$) or to deny access to those privileges for other dependent services ($revoked(priv_a, dep)$).

Example: An attacker targets the NFS server, and thus directly provoking the infection `infected(access root directories, NFS)`. Root directories remain infected when they are shared through the web-NFS dependency (please refer to the second pattern in listing 1.2). The denial of web access to the NFS service (`revoked(web IP in /etc/exports, web)`) keeps the root directories infected, but hampers the use of infected directories by the web service. Downward propagation is also hampered by denying access to the root directories for the NFS service, i.e. `revoked(access root directories, NFS)`.

Third propagation pattern mimics availability impact propagations that occur in case of functional dependency compositions. A privilege that is revoked to a service is also revoked to all its dependent services. Interleaving with availability propagations includes the ability to implement disjunctive dependencies. Impact propagation may be prevented if the following condition is satisfied. It expresses the ability for a dependent service to use more than only one antecedent service in order to obtain its required privileges.

$$revoked(priv_o, tgt) \wedge \exists(dep, ant, priv_a, priv_b) : (dep \xrightarrow{dep.priv_a} tgt.priv_o) \wedge$$
$$(dep \xrightarrow{dep.priv_b} ant.priv_o) \wedge granted(priv_o, ant) \Rightarrow \neg revoked(priv_o, dep)$$

Example: we discuss the example of a `DoS` attack against the `NFS` service. It revokes access to the root directories for the `NFS` service, i.e. `revoked(access root directories, NFS)`. This privilege may no longer be shared with the web service, i.e. `revoked(access root directories, web)`. Impact propagation may be hampered in case of another web-`NFS` dependency providing load balancing with the failed `NFS` dependency.

Fourth propagation pattern mimics availability impact propagations that occur in case of logical dependency compositions. A privilege that is revoked to a service may no longer be used by this service in order to support dependencies for other services. We also refer to dependency disjunction in order to illustrate the condition for a response to prevent this pattern. It is written as:

$$revoked(priv_o, tgt) \wedge \exists(ant_1, ant_2, priv_a, priv_1) : (tgt \xrightarrow{tgt.priv_o} ant_1.priv_a) \wedge$$
$$(tgt \xrightarrow{tgt.priv_1} ant_2.priv_a) \wedge granted(priv_1, tgt) \Rightarrow \neg revoked(priv_a, tgt)$$

The example for this inference rule is similar to the one of the third rule, but denying the web instead of the `NFS` service. It thus revokes the `IP` connection to the `NFS` service, i.e. `revoked(web IP in /etc/exports, web)`.

The evaluation of response collateral damages, i.e. the `CD` metric, consists of adding only the response to the model and to exclude the intrusion. Response collateral damages are not only restrained to availability impacts, that is privilege revocations. It may also provoke privilege infection when yet some granted privilege enables the propagation of an infection that was previously intercepted. This is automatically depicted by the inference process using the rules in listing 1.2. Response collateral damages are thus closely related to the current system state and may not be statically defined beforehand. On the other hand, the evaluation of the `RC` metric, that is the combined impact of intrusion and response, requires adding both intrusion and response to the model. The resulting cost after all impacts have been propagated corresponds to the metric `RC`. The use of inference rules that are based on first order logic statements guarantees the convergence of the propagation process within a polynomial time.

The response evaluation process presented in this section is used to assist intrusion response systems by comparing candidate responses. It is implemented in a dynamic environment that requires interleaving with the dependency model. An appropriate implementation of this model must be thus provided. We suggest using Colored Petri Nets (`CPN`) for this purpose.

4 Simulation Platform

4.1 Using Colored Petri Nets

Although we may use a datalog inference process to implement our model, we discarded this alternative for the following reasons. The use of our dependency

model as part of a cost-sensitive response mechanism requires interleaving with the inference process. This is, to the best of our knowledge, difficult to integrate in a datalog engine. Furthermore, the size of datalog inference engines may often be unacceptable. The complexity of these engines makes them inappropriate for systems including a large number of resources and dependencies.

On the other hand, CPNs [14] provide appropriate features to implement our model. They are extensions of petri nets where tokens transit between model places. We use CPN tokens to represent privileges in our model. A service dependency is modeled as a CPN transition that is enabled when some conditions are met, i.e. enough privileges for the dependent service to support its dependency. We also use CPN places to represent user equivalence classes. They are initially marked with default user privileges. User places thus interact with system services through well-defined interfaces. Attackers are actually modeled in a different way. They are not assigned explicit places because their behavior is considered as unpredictable. Any infected privilege (i.e. token) would be thus considered as an attacker property. Last but not least, a CPN simulator enables the iterative simulation and interleaving with the simulation process. It may also constrain the simulation to run on a transition basis. System costs are obtained after the CPN reaches a state where no more transitions are activated.

We transform the request statement that represents a service dependency into the CPN transition in figure 2. This transition shares tokens (i.e. requested privileges) between the dependent and the antecedent services represented as CPN places. This transition is constrained by the existence of specific tokens (i.e. dependency requirements) in the destination place (i.e. dependent service). The privileges used by the dependent service to support its request are implemented as a transition activation constraint. This transition satisfies the properties of the request statement for service dependencies. It is not activated unless the destination place contains the privileges that are required to support the dependency (i.e. $priv_i$). It also shares the privilege $priv_o$ between the dependent and antecedent resources. The transition in figure 2 also satisfies the impact propagation patterns. The token $priv_o$ is infected in the destination place when all $priv_i$ tokens are infected or when $priv_o$ was already infected at the source place (we add a boolean attribute to a token definition, it is set to true when this token is infected). We thus implement the first and second statements in listing 1.2. On the other hand, the transition is only activated when the source place includes the $priv_o$ token and the destination place includes the $priv_i$ tokens. We thus implement the third and fourth statements in listing 1.2.

4.2 Simulation Process

We implement the simulation platform using the CPN tools simulator[2]. The overall architecture we use is illustrated in figure 3. The service dependency model, expressed as a CPN skeleton (without initial marking) is a static input to the CPN simulator. It reliably describes the services that constitute the modeled

[2] http://wiki.daimi.au.dk/cpntools/

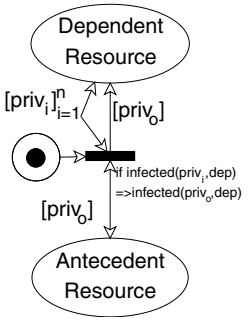


Fig. 2. CPN dependency

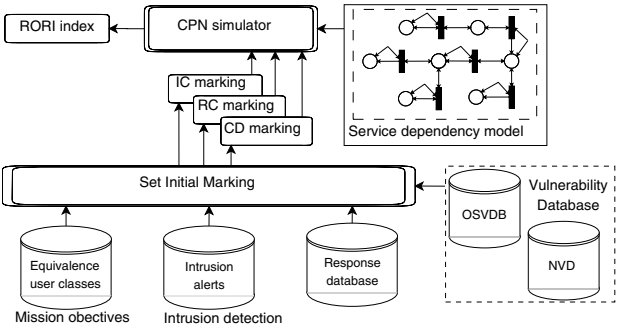


Fig. 3. Dependency simulation framework

environment and its dependencies. It is obtained by transforming dependency statements into a CPN model, as shown in the previous paragraph. The initial marking of the CPN model is dynamic. It characterizes the dynamic system state, including currently applied responses and the existing, but contained, intrusions. Responses either grant or revoke privileges to their target components. This is represented as token assignment or token extraction from appropriate places. A token is either explicitly infected by an intrusion when determining the initial marking of the CPN model (infection boolean attribute set to true), or further infected through CPN simulation (please refer to figure 2). By the end of the simulation, all infected tokens (the boolean attribute set to true) and revoked tokens (by comparing the final marking of the CPN model to the equivalence classes) are used to evaluate the RORI index attributes. As illustrated in figure 3, the CPN model is simulated three times for each candidate response set. The first simulation is executed by introducing only the intrusion attempt in order to obtain the IC_b metric. The second simulation is executed by introducing only the response in order to obtain the CD metric. Finally, the third simulation is executed by introducing both intrusion and response in order to obtain the RC metric. Measures are combined within the RORI index. This operation is iterated for all candidate response sets that are proposed by an external response system. We finally choose the response set that provides a maximal positive RORI index.

5 Case Study

We demonstrate the use of our proposal through the simplified example of an enterprise email service, which is illustrated in figure 4. It uses IMAP and the native exchange mailing protocols, i.e. outlook and outlook web access. Intranet users access the email service using a courier-Imap server or the outlook web access (OWA). The courier-Imap server uses IMAP extension for the exchange server. Extranet users connect to the email service through web access to an apache2 server connected to a DMZ. The latter connects to the IMAP server or to the OWA server using the mod-proxy extension for apache2 server.

In our simple example, we pick-up two user classes, which are `Intranet` and `Extranet` email users. `Extranet` users have the privilege of being able to connect to the web server, as well as one credential, which is a valid email account. `Intranet` users have the privilege of being connected to the `Intranet`, and the same credential as for `Extranet` users. Both user classes require access to mailboxes hosted by the exchange server. The security objectives in this example are to guarantee the mailbox availability and to prevent unintended mailbox access. The email platform also includes four elementary services, which are the web, `IMAP`, `OWA` and Exchange services. Service dependencies are summarized in the resulting `CPN` model in figure 5. The default initial marking (i.e. with no security threats) is illustrated within parenthesis inside `CPN` places.

We examine the following attack scenario. In a first step, an attacker exploits a vulnerability to the web application that enables to execute arbitrary code on the web server. The attacker uses the web server as a stepping stone in order to access the `IMAP` server through the `DMZ-Intranet` firewall. He exploits a flaw for the authentication front-end of the `courier-Imap` server which enables the attacker to connect to the `IMAP` server. The attacker finally conducts a buffer overflow attack to have a root shell on the `IMAP` server. The first attack step infects the `Dm` token (please refer to figure 5), that is to have direct access to the `DMZ`. The infection does not propagate through the dependency between the web and the mail delivery services because the `web-Intranet` transition also requires the `Vm` token to be infected (i.e. a valid user account) in order to propagate the infection. The second attack step infects the `Int` token, that is the connection to the `IMAP` service. The infection does not propagate elsewhere (please check the `CPN` model). The last attack step infects the token `Ex`, i.e. the `IMAP` account to the exchange server. The infection propagates through the `IMAP`-Exchange transition because both `Ex` and `Int` tokens are infected. The mailbox access (`Mb`) token is thus infected (first propagation pattern in listing 1.2). One security objective has failed, that is the misuse of user mailboxes ($infected(Mb, Exchange)$).

Two responses are possible: The first blocks access to the web application. It revokes the `Cw` token for `Extranet` users since the attacker is unknown. The second denies access to the Exchange server for the vulnerable `IMAP` service, i.e. to revoke the token `Ex` to the `IMAP` place. By revoking the `Cw` token, extranet users cannot access their emails because the appropriate transitions are disabled

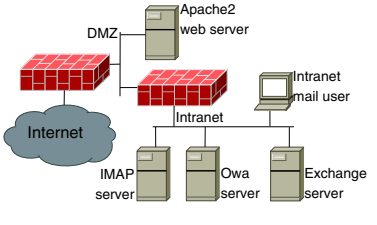


Fig. 4. Email case study

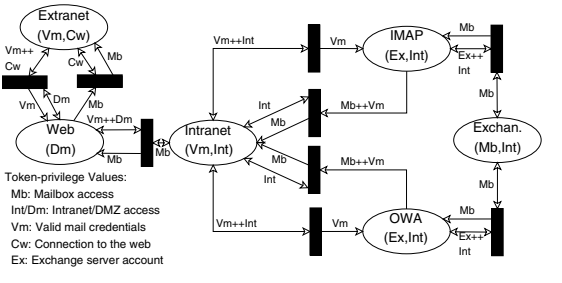


Fig. 5. `CPN` dependency representation

(fourth propagation pattern in listing 1.2). Meanwhile, `Intranet` users are not affected (please check the `CPN` model). The second response prevents email access only through the `IMAP` protocol while not affecting other protocols. The second response has less collateral damages since it does not prevent email access for external users (disjunctive dependency that prevents the third propagation pattern in listing 1.2). Both responses prevent the infection to propagate to user mailboxes. Based on the `RORI` index values provided by the `CPN` simulations discussed above, the second response is selected when detecting the third attack.

6 Conclusion

We implemented in this paper the Return-On-Response-Investment as an adaptation to the `ROI` index. The `RORI` index, in contrast to most existing response evaluation functions, does not use informal cost metrics that rely on expert knowledge. Rather, it is accompanied with a complete methodology to evaluate the metrics that contribute to this index. To the best of our knowledge, the `RORI` index is the first to consider not only response collateral damages, but also response effects on intrusion. The `RORI` index is implemented along with a comprehensive service dependency model that enables to track intrusion and response impacts in the target system. By doing so, we introduce intrusions and responses in the model and analyze the interference between their impacts. The `RORI` index supports privilege infections, that express confidentiality and integrity impacts, but also privilege revocations that express availability impacts. It outruns most existing response evaluation mechanisms that rely on dependability management techniques and therefore only apply to availability failures.

Future work will investigate how time may be added to the `RORI` index. The cost of a privilege infection or revocation may not be static. A response may have a higher `RORI` index when an attack is detected, but later for this index to be degraded in favor of other responses. We may also extend the `RORI` index to tune impacts as the attacker gets closer to critical mission objectives. An intrusion impact will not be restrained to the direct effects in terms of privilege infection and revocation. It will also consider the impact of new attack steps that are made possible by the current intrusion if no response is enacted.

References

1. Aceituno, V.: Return on security investment. ISSA Jounal 1, 16–19 (2006)
2. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proc. 9th ACM Conf. on Computer and Communications Security, pp. 217–224 (2002)
3. Artz, M.L.: A Network Security Planning Architecture. Ph.D. thesis, Cambrdige: Massachusetts Institute of Technology (May 2002)
4. Balepin, I., Maltsev, S., Rowe, J., Levitt, K.: Using specification-based intrusion detection for automated response. In: Vigna, G., Krügel, C., Jonsson, E. (eds.) RAID 2003. LNCS, vol. 2820, pp. 136–154. Springer, Heidelberg (2003)
5. Cuppens, F., Autrel, F., Yacine Bouzida, J.G., Gombault, S., Sans, T.: Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework. Annals of Telecommunications 61, 197–217 (2006)

6. Dacier, M., Deswarte, Y., Kaaniche, M.: Quantitative assessment of operational security: models and tools. Tech. Rep. 96493, LAAS (May 1996)
7. Dacier, M., Deswartes, Y.: Privilege graph: An extension to the typed access matrix model. In: European Symp. on Research in Computer Security, pp. 319–334 (1994)
8. Debar, H., Thomas, Y., Cuppens, F., Cuppens-Boulahia, N.: Enabling automated threat response through the use of a dynamic security policy. Journal in Computer Virology 3, 195–210 (2007)
9. Foo, B., Wu, Y.S., Mao, Y.C., Bagchi, S., Spafford, E.: Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment. In: Proc. Intr'l Conf. DSN, pp. 508–517 (2005)
10. Jahnke, M., Thul, C., Martini, P.: Graph based metrics for intrusion response measures in computer networks. In: 32nd IEEE Conf. Local Computer Networks (2007)
11. Jajodia, S., Noel, S.: Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response. Algorithms, Architectures and Information Systems Security 1, 285–305 (2007)
12. Kheir, N., Debar, H., Cuppens, F., Cuppens-Boulahia, N., Viinikka, J.: A service dependency modeling framework for policy-based response enforcement. In: Flegel, U., Bruschi, D. (eds.) DIMVA 2009. LNCS, vol. 5587, pp. 174–193. Springer, Heidelberg (2009)
13. Kheir, N., Debar, H., Cuppens-Boulahia, N., Cuppens, F., Viinikka, J.: Cost assessment for intrusion response using dependency graphs. In: Proc. IFIP Intrn'l Conf. N2S (2009)
14. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. Intr'l Journal Software Tools for Technology Transfer, 98–132 (1998)
15. Lee, W., Fan, W., Miller, M., Stolfo, S.J., Zadok, E.: Toward cost-sensitive modeling for intrusion detection and response. Journal of Computer Security 10, 5–22 (2002)
16. Li, N., Mitchell, J., Winsborough, W.: Design of a role-based trust-management framework. In: Proc. IEEE Symp. on Security and Privacy, p. 114 (2002)
17. McMillan, R.: Internet explorer vulnerable to hackers, warn experts. microsoft and avg warn of danger. TechWorld magazine (March 2010)
18. Mead, N.R., McGraw, G.: A portal for software security. In: IEEE Security & Privacy, pp. 75–79 (2005)
19. Microsoft: Why microsoft dynamics for high-tech and electronics manufacturers? Microsoft Dynamics CRM
20. Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: Proc. 6th ACM Conf. Internet measurement, pp. 41–52 (2006)
21. Ross, S., Westerfield, R., Jordan, B.: Fundamentals of Corporate Finance Standard Edition. McGraw-Hill/Irwin (2005)
22. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
23. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE Symp. Security & Privacy (2002)
24. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. Intr'l Journal of Information and Computer Security 1, 169–184 (2007)
25. Strasburg, C., Stakhanova, N., Basu, S., Wong, J.S.: Intrusion response cost assessment methodology. In: Proc. ACM Symp. ASIACCS, pp. 388–391 (2009)
26. Toth, T., Kruegel, C.: Evaluating the impact of automated intrusion response mechanisms. In: Proc. 18th Annual Conf. ACSAC (2002)

Secure Code Update for Embedded Devices via Proofs of Secure Erasure

Daniele Perito[1] and Gene Tsudik[2]

[1] INRIA Rhône-Alpes, France
[2] University of California, Irvine, USA

Abstract. Remote attestation is the process of verifying internal state of a remote embedded device. It is an important component of many security protocols and applications. Although previously proposed remote attestation techniques assisted by specialized secure hardware are effective, they not yet viable for low-cost embedded devices. One notable alternative is software-based attestation, that is both less costly and more efficient. However, recent results identified weaknesses in some proposed software-based methods, thus showing that security of remote software attestation remains a challenge.

Inspired by these developments, this paper explores an approach that relies neither on secure hardware nor on tight timing constraints typical of software-based technqiques. By taking advantage of the bounded memory/storage model of low-cost embedded devices and assuming a small amount of read-only memory (ROM), our approach involves a new primitive – Proofs of Secure Erasure (PoSE-s). We also show that, even though it is effective and provably secure, PoSE-based attestation is not cheap. However, it is particularly well-suited and practical for two other related tasks: secure code update and secure memory/storage erasure. We consider several flavors of PoSE-based protocols and demonstrate their feasibility in the context of existing commodity embedded devices.

1 Introduction

Embedded systems are encountered in many settings, ranging from mundane to critical. In particular, wireless sensor and actuator networks are used to control industrial systems as well as various utility distribution networks, such as electric power, water and fuel [24,13]. They are also widely utilized in automotive, railroad and other transportation systems. In such environments, it is often imperative to verify the internal state of an embedded device to assure lack of spurious, malicious or simply residual code and/or data.

Attacks on individual devices can be perpetrated either physically [1] or remotely [30,14,15]. It is clearly desirable to detect and isolate (or at least restore) compromised nodes. One way to accomplish this is via *device attestation*, a process whereby a trusted entity (e.g., a base station or a sink) verifies that an embedded device is indeed running the expected application code and, hence,

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 643–662, 2010.

has not been compromised. In recent years, several software-based attestation protocols have been proposed [27,29,31]. The goal of these protocols is to verify the trustworthiness of resource-constrained systems, without requiring dedicated tamper-resistant hardware or physical access to the device. Attestation based on tamper-resistant hardware [12], though effective [17], is not yet viable on low-cost commodity embedded devices. Furthermore, hardware attestation techniques, while having stronger security properties, ultimately rely on a per-device TPM and the availability of a trusted BIOS that begins attestation at boot time.

In contrast, remote software attestation typically involves a challenge-response interaction, whereby a trusted entity, the *verifier*, challenges a remote system, called the *prover*, to compute a cryptographic checksum of its internal state, i.e., code memory, registers and program counter. Depending on the specific scheme, the prover either computes this checksum using a fixed checksum routine and a nonce [29], or downloads a new routine from the verifier as part of the protocol [31]. The checksum routine sequentially updates the checksum value by loading and processing device memory blocks. Since the verifier is assumed to know the exact memory contents and hardware configuration of the prover, it can compute the expected checksum value and match it with the prover's response. If there is a match, the prover is assumed to be *clean*; otherwise, either it has been compromised or a fault has occurred. In either case, appropriate actions can be taken by the verifier.

Recently, several proposed software-based attestation schemes were shown to be vulnerable [9] to certain attacks, summarized in Section 2. These negative results show that software-based attestation remains to be an interesting and important research challenge.

To summarize, hardware-based attestation techniques are not quite practical for current and legacy low-cost embedded systems. Whereas, state-of-the-art in software-based attestation offers unclear (or, at best, *ad hoc*) security guarantees. These factors motivate us to look for alternative approaches. Specifically, in this paper we zoom out of just attestation and consider a broader issue of *secure remote code update*. To obtain it, we introduce a new cryptographic primitive called a *Proof of Secure Erasure* (PoSE). We suggest some simple PoSE constructs based on equally simple cryptographic building blocks. This allows us, in contrast to prior software-based attestation techniques, to obtain provable security guarantees, under reasonable assumptions.

Our approach can be used to obtain several related security properties for remote embedded devices. The most natural application is *secure memory erasure*. Embedded devices might collect sensitive or valuable data that – after being uploaded to a sink or a base station – must be securely erased. Also, if code resident on an embedded device is sensitive or proprietary, it might eventually need to be securely erased by a remote controller. We note that secure erasure may be used as a prelude to secure code update or attestation. This is because, after

secure erasure of all prior state, new (or old) code can be downloaded onto an embedded device with the assurance that no other code or data is being stored.

The intended **contribution** of this paper is three-fold:

1. We suggest a simple, novel and practical approach to secure erasure, code update and attestation that falls between (secure, but costly) hardware-based and (efficient, but uncertain in terms of security) software-based techniques.
2. We show that the problem of secure remote code update can be addressed using Proofs of Secure Erasure (PoSE-s).
3. We propose several PoSE variants and analyze their security as well as efficiency features. We also assess their viability on a commodity sensor platform.

Organization: Section 2 reviews related work. Next, Section 3 describes the envisaged network environment and states our assumptions. Section 4 presents our design rationale, followed by proposed protocols in Section 5. Implementation, experiments and performance issues are discussed in Section 6. Limitations and directions for future work are addressed in Section 7. An extension to support multi-device attestation is deferred to Appendix A.

2 Related Work

We now summarize related work, which generally falls into either software- or hardware-based attestation methods. We also summarize some relevant cryptographic constructs.

2.1 Hardware Attestation

Static Integrity Measures: Secure boot [2] was proposed to ensure a chain of trusted integrity checks, beginning at power-on with the BIOS and continuing until the kernel is loaded. These integrity checks compare the computation of a cryptographic hash function with a signed value associated with the checked component. If one of the checks fails, the system is rebooted and brought back to a known saved state.

Trusted Platform Module (TPM) is a secure coprocessor that stores an integrity measure of a system according to the specifications of the Trusted Computing Group (TCG) [34]. Upon boot, the control is passed to an immutable code base that computes a cryptographic hash of the BIOS, hash that is then securely stored in the TPM. Later, control is passed to the BIOS and the same procedure is applied recursively until the kernel is loaded. In contrast to secure boot, this approach does not detect integrity violations, instead the task is left to a remote verifier to check for integrity.

[25] proposed to extend the functionality of the TPM to maintain a chain of trust up to the application layer and system configuration. In order to do so,

they extend the Linux kernel to include a new system call that measures files and adds the checksum in a list stored by the kernel. The integrity of this list is then sealed in the TPM. A similar goal is pursued in NGSCB [12], that takes a more radical approach by partitioning a system in a trusted and an untrusted part, each running a separate operating system, where only the trusted part is checked.

Dynamic Integrity Measures: In [21], the use of TPM is extended to provide system integrity checks of run-time properties with ReDAS (**Re**mote **D**ynamic **A**ttestation **S**ystem). At every system call, a kernel module checks the integrity of constant properties of dynamic objects, e.g., invariant relations between the saved frame pointer and the caller's stack frame. Upon detection of an integrity violation, the kernel driver seals the information about the violation in the TPM. A remote verifier can ask the prover to send the sealed integrity measures and thus verify that no integrity violations occurred. However ReDAS only checks for violations of a subset of the invariant system properties and nothing prevents an adversary to succeed in subverting a system without modifying the properties checked by ReDAS. Extending the set of attested properties is difficult due to the increased number of false positives generated by this approach, for example in case of dynamic properties classified as invariants by mistake.

2.2 Software Attestation

Most software-based techniques rely on challenge-response protocols that verify the integrity of code memory of a remote device: an attestation routine on the prover computes a checksum of its memory along with a challenge supplied by the verifier. In practice, memory words are read sequentially and fed into the attestation function. However, this simple approach does not guarantee that the attestation routine is computed faithfully by the prover. In other words, a prover can deviate (via some malicious code) from its expected behavior and still compute a correct checksum, even in the presence of some malicious memory content.

Time-based attestation: SWATT [29] is a technique that relies on response timing to identify compromised code: memory is traversed using a pseudo-random sequence of indexes generated from a unique seed sent by the verifier. If a compromised prover wants to pass attestation, it has to redirect some memory accesses to compute a correct checksum. These redirections are assumed to induce a remotely measurable delay in the attestation that can be used by the verifier to decide whether to trust the prover's response. The same concept is used in [27] where, the checksum calculation is extended to include also dynamic properties, e.g., the program counter or the status register. Furthermore the computation is optimised by having the checksum computed only on the attestation function itself.

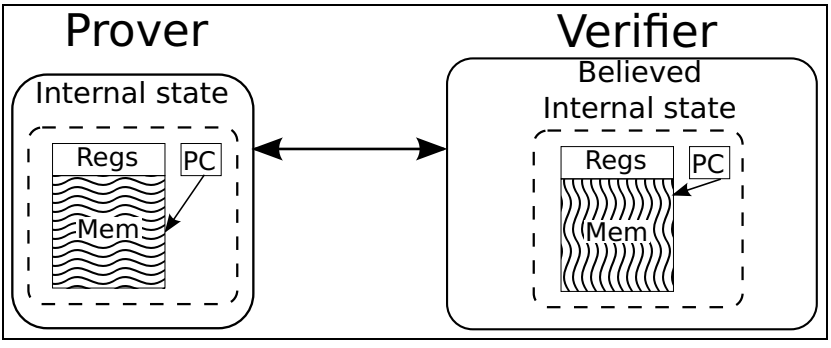


Fig. 1. Generic remote attestation

Jakobsson, et al. [18] proposed an attestation scheme to detect malware on mobile phones. This attestation scheme relies on both careful response timing and memory filling. Timing is used to measure attestation computation as well as external memory access and wireless links. Security of this approach depends on a number of hardware-specific details (e.g., flash memory access time). Hence, formal guarantees and portability to different platforms appear difficult to achieve.

Memory-based attestation: In [35] sensors collaborate to attest the integrity of their peers. At deployment time, each empty node's memory is filled with randomness, that is supposed to prevent malicious software from being stored, without deleting some parts of the original memory. A similar approach is taken in [10], but, instead of relying on pre-deployed randomness, random values are generated using a PRF seeded by a challenge sent by the verifier and are used to fill the prover's memory. However, this does not assure compliance to the protocol of a malicious node that could trade computation for memory and still produce a valid checksum.

Gatzer et al. [16] suggest a method where random values are sent to a low-end embedded device (e.g., a SIM card) and then read back by the verifier, together with the attestation routine itself (called *Quine* in the paper). This construction, while quite valid, was only shown to be effective on an 8-bit Motorola MCU with an extremely simple instruction set. Also, this scheme applies only to RAM, whereas, we our approach aims to verify all memory/storage of an embedded device.

Attestation based on self-modifying code: [31] proposed to use a distinct attestation routine for each attestation instance. The routine is transferred right before the protocol is run and uses self-modifying code and obfuscation techniques to prevent static analysis. Combined with timing of responses, this makes it difficult for the adversary to reverse-engineer the attestation routine fast enough to cheat the protocol and produce a valid (but forged) result. However, this approach

relies on obfuscation techniques that are difficult to prove secure. Furthermore, some such techniques are difficult to implement on embedded systems, where code is stored in flash memory programmable only by pages.

Attacks: Recently, [9] demonstrated several flaws and attacks against some software attestation protocols. Attacks can be summarized as: failure to verify other memories apart from code memory (exploited through ROP attacks [30]); insufficient non-linearity in time-based attestation routines, which could be exploited to generate correct results over forged memory; failure to recognize that legitimate code memory can be compressed and thus save space for malicious code, while still remaining accessible for attestation. Also, [32] points out that side-effects, such as cache misses, are not sufficient to check software integrity using time-based approaches such as [20].

2.3 Provable Data Possession and Proofs of Retrievability

The problem at hand bears some resemblance to Provable Data Possession (PDP) [3,4] and Proof of Retrievability (POR) schemes [19]. However, this resemblance is superficial. In settings envisaged by POR and PDP, a resource-poor client outsources a large amount of data to a server with an unlimited storage capacity. The main challenge is for a client to efficiently and repeatedly verify that the server indeed stores all of the client's data. This is markedly different from attestation where the prover (embedded device) must not only prove that it has the correct code, but also that it stores *nothing else*. Another major distinction is that, in POR and PDP, the verifier (client) is assumed not to keep a copy of its outsourced data. Whereas, in our setting, the verifier (base station) obviously keeps a copy of any code (and/or data) that embedded devices must store.

2.4 Memory-Bounded Adversary

Cryptographic literature contains a number of results on security in the presence of a memory-bounded adversary [8]. Our setting also features an adversary in the guise of a memory-limited prover. However, the memory-bounded adversary model involves two or more honest parties that aim to perform some secure computation. Whereas, in our case, the only honest party is the verifier and no secrets are being computed as part of the attestation process.

3 Assumptions and Adversary Model

Secure code update involves a *verifier* $\mathcal{V}$ and a *prover* $\mathcal{P}$. Internal state of $\mathcal{P}$ is represented by a tuple $S = (M, RG, pc)$ where M denotes $\mathcal{P}$'s memory of size n (in bits), $RG = rg_1, ..., rg_m$ is the set of registers and pc is the program counter. We refer to S_P as the real internal state of the prover and S_V the internal state

Table 1. Notation Summary

$X \longleftarrow Y \ : \ Z$	Y sends message Z to X
$X_1, ..., X_t \Longleftarrow Y \ : \ Z$	Y multicasts message Z to $X_1, ..., X_t$
$\mathcal{V}$	Verifier
$\mathcal{P}$	Prover
ADV	Adversary
M	Prover's contiguous memory
$M[i]$	i-th bit in M $(0 \leq \ i < n)$
n	Bit-size of M
RG	Prover's registers $rg_1, ..., rg_m$
pc	Prover's program counter
$S_P = (M, R, pc)$	Prover's internal state
S_V	Verifier' view of Prover's internal state
$R_1...R_n$	Verifier's n-bit random challenge
$C_1...C_n$	n-bit program code (see below)
k	Security parameter
K	MAC key

of the prover, as viewed by the verifier. Secure code update can be viewed as a means to ensure that $S_V = S_P$. Our notation is reflected in Table 1.

$\mathcal{P}$ is assumed to be a generic embedded device – e.g., a sensor, an actuator or a computer peripheral – with limited memory and other forms of storage. For the ease of exposition, we assume that all $\mathcal{P}$'s storage is homogeneous and contiguous. (This assumption can be easily relaxed, as discussed in section 6.2) From here on, the term *"memory"* is used to denote all writable storage on the device. The verifier is a comparatively powerful computing device, e.g., a laptop-class machine.

Our protocol aims to ascertain the internal state of $\mathcal{P}$. The adversary is a program running in the prover's memory, e.g., a malware or a virus. Since the adversary executes on $\mathcal{P}$, it is bounded by the computational capabilities of the latter, i.e., memory size n.

We assume that the adversary cannot modify hardware configuration of $\mathcal{P}$[1], i.e., all anticipated attacks are software-based. The adversary has complete read-/write access to $\mathcal{P}$'s memory, including all cryptographic keys and code. However, in order to achieve provable security, our protocol relies on the availability of a small amount of Read-Only Memory (ROM) that the adversary can read, but not modify. Finally, the adversary can perform both passive (such as eavesdropping) and active (such as replay) attacks. An attack succeeds if the compromised $\mathcal{P}$ device passes the attestation protocol despite presence of malicious code or data.

[1] In fact, one could easily prove that software attestation is in general impossible to achieve against hardware modifications.

We note that ROM is not unusual in commodity embedded systems. For example, the Atmel ATMEGA128 micro-controller allows a small portion of its flash memory to be designated as read-only. Writing to this memory portion via software becomes impossible and can only be enabled by physically accessing the micro-controller with an external debugger.

As in prior attestation literature, [10,23,26,27,28,29,31,35], we assume that the compromised prover device does not have any *real time* help. In other words, during attestation, it does not communicate with any other party, except the verifier. Put another way, the adversary maintains complete radio silence during attestation. In all other respects, the adversary's power is assumed to be unlimited.

4 Design Rationale

Our design rationale is simple and based on three premises:

- **First**, we broaden our scope beyond attestation, to include both secure memory erasure and secure code update. In the event that the updated code is the same as the prior code, secure code update yields secure code attestation. We thus consider secure code update to be a more general primitive than attestation.
- **Second**, we consider two ways of obtaining secure code update: (1) download new code to the device and then perform code attestation, or, (2) securely erase everything on the device and then download new code. The former brings us right back to the problematic software-based attestation, while the latter translates into a simpler problem of secure memory erasure, followed by the download of the new code. We naturally choose the latter.

  Correctness of this approach is intuitive: since the prover's memory is strictly limited, its secure erasure implies that no prior data or code is resident; except for a small amount of code in ROM, which is immutable. Because the adversary is assumed to be passive during code update, download of new code always succeeds, barring any communication errors.
- **Third**, based on the above, we do not aim to *detect* the presence of any malicious code or extraneous data on the prover. Instead, our goal is to make sure that, after erasure or secure code update, no malicious code or extraneous data remains.

Because our approach entails secure erasure of *all* memory, followed by the code download, it might appear to be very inefficient. However, as discussed in subsequent sections, we use the aforementioned approach as a base case that offers unconditional security. Thereafter, we consider ways of improving and optimizing the base case to obtain appreciably more practical solutions.

5 Secure Code Update

The base case for our secure code update approach is depicted in Figure 3. It is essentially a four-round protocol, where:

- Rounds one and two comprise secure erasure of all writable memory contents.
- Rounds three and four represent code update.

Note that there is absolutely no interleaving between any adjacent rounds. The "evolution" of prover's memory during the protocol is shown in Figure 2.

As mentioned earlier, we assume a small ROM unit on the prover. In the base case, ROM houses two functions: read-and-send and receive-and-write. During round one, receive-and-write is used to receive a random bit R_i and write it in location $M[i]$, for $0 \leq \ i < n$. At round two, read-and-send reads a bit from location $M[i]$ and sends it to the prover, for $0 \leq \ i < n$. (In practice, read and write operations involve *words* and not individual bits. However, this makes no difference in our description.)

If we assume that the $\mathcal{V} \leftrightarrow \mathcal{P}$ communication channel is lossless and error-free, it suffices for round four to be a simple acknowledgement. Otherwise, round four must be a checksum of the code downloaded in round three. In this case, the checksum routine must reside in ROM; denoted by $H()$ in round four of Figure 3. In the event of an error, the entire procedure is repeated.

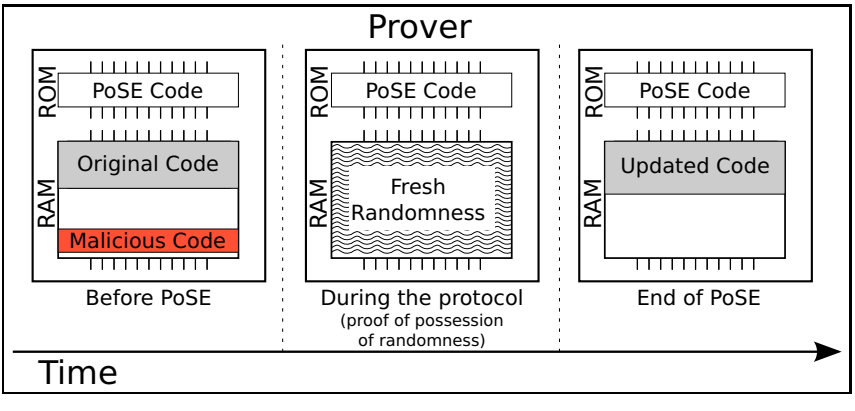


Fig. 2. Prover's Memory during Protocol Execution

[1]	$\mathcal{P}$	$\longleftarrow$	$\mathcal{V}$	: $R_1, ..., R_n$
[2]	$\mathcal{P}$	$\longrightarrow$	$\mathcal{V}$	: $R_1, ..., R_n$
[3]	$\mathcal{P}$	$\longleftarrow$	$\mathcal{V}$	: $C_1, ..., C_n$
[4]	$\mathcal{P}$	$\longrightarrow$	$\mathcal{V}$	: ACK or $H(C_1, ..., C_n)$

Fig. 3. Base Case Protocol

5.1 Efficient Proof of Secure Erasure

As shown in Figure 3, secure erasure is achieved by filling prover's memory with verifier-selected randomness, followed by the prover returning the very same randomness to the verifier. On the prover, these two tasks are executed by the ROM-resident read-and-send and receive-and-write functions, respectively.

It is easy to see that, given our assumptions of: i) adversary's software only attacks, ii) prover's fixed-size memory M, iii) no hardware modification of compromised provers, and iv) source of true randomness on the verifier, the proof of secure erasure holds. In fact, the security of erasure is *unconditional*, due to lack of any computational assumptions.

Unfortunately, this simple approach is woefully inefficient as it requires a resource-challenged $\mathcal{P}$ to send and receive n bits. This prompts us to consider whether secure erasure can be achieved by either (1) sending fewer than n bits to $\mathcal{P}$ in round one, or (2) having $\mathcal{P}$ respond with fewer than n bits in round two. We defer (1) to future work. However, if we sacrifice unconditional security, bandwidth in round two can be reduced significantly.

One way to reduce bandwidth is by having $\mathcal{P}$ return a fixed-sized function of entire randomness received in round one. However, choosing this function is not entirely obvious: for example, simply using a cryptographically suitable hash function yields an insecure protocol. Suppose we replace round two with $CHK = H(R_1, ..., R_n)$ where $H()$ is a hash function, e.g., SHA. Then, a malicious $\mathcal{P}$ can start computing CHK in real time, while receiving $R_1, ..., R_n$ during round one, without storing these random values.

An alternative is for $\mathcal{P}$ to compute a MAC (Message Authentication Code) using the last k bits of randomness – received from $\mathcal{V}$ in round one – as the key. (Where k is sufficiently large, i.e., at least 128 bits.) A MAC function can be instantiated using constructs, such as AES CBC-based MAC [7], AES CMAC or HMAC [6]. However, minimum code size varies, as discussed in Section 6. In this version of the protocol, the MAC function must be stored in ROM. Clearly, a function with the lowest memory utilization is preferable in order to minimize the amount of working memory that $\mathcal{P}$ needs to reserve for computing MAC-s.

Claim: Assuming a cryptographically strong source of randomness on $\mathcal{V}$ and a cryptographically strong MAC function, the following 2-round protocol achieves secure erasure of all writable memory M on $\mathcal{P}$:

[1]	$\mathcal{P}$	$\longleftarrow$	$\mathcal{V}$	: $R_1, ..., R_n$ where $K = R_{n-k+1}...R_n$
[2]	$\mathcal{P}$	$\longrightarrow$	$\mathcal{V}$	: $MAC_K(R_1, ..., R_{n-k})$

where k is the security parameter (bit-size of the MAC key) and K is the k-bit string $R_{n-k+1}, ..., R_n$.

Proof (Sketch): Suppose that malicious code MC occupies $b > 0$ bits and persists in M after completion of the secure code update protocol. Then, during round one, either: (1) some MAC pre-computation was performed and certain

bits (at least b) of $R_1, ..., R_{n-k}$ were not stored in M, or (2) the bit-string $R_1, ..., R_{n-k}$ was compressed into a smaller x-bit string ($x < n - k - b$). However, (1) is infeasible since the key K is only communicated to $\mathcal{P}$ at the very end of round one, which precludes any MAC pre-computation. Also, (2) is infeasible since $R_1, ..., R_{n-k}$ is originates from a cryptographically strong source of randomness and its entropy rules out any compression. □

Despite its security and greatly reduced bandwidth overhead, this approach is still computationally costly considering that it requires a MAC to be computed over entire n-bit memory M. One way to alleviate its computational cost is by borrowing a technique from [4] that is designed to obtain a probabilistic proof in a Provable Data Possession (PDP) setting discussed in Section 2.3. The PDP scheme in [4] assumes that data outsourced by $\mathcal{V}$ (client) to $\mathcal{P}$ (server) is partitioned into fixed-size m-bit blocks. $\mathcal{V}$ generates a sequence of t block indices and a one-time key K which are sent to $\mathcal{P}$. The latter is then asked to compute and return a MAC (using K) of the t index blocks. In fact, these t indices are not explicitly transferred to $\mathcal{P}$; instead, $\mathcal{V}$ supplies a random seed from which $\mathcal{P}$ (e.g., using a hash function or a PRF) generates a sequence of indices.

As shown in [4], this technique achieves detection probability of: $\mathcal{P} = 1 - (1 - \frac{m}{d})^t$ where m is the number of blocks that $\mathcal{V}$ **did not** store (i.e., blocks where malicious code resides), d is the total number of blocks and t is the number of blocks being checked.

Consider a concrete example of a Mica Mote with 128 Kbytes of processor RAM and further 512 Kbytes of data memory, totaling 640 Kbytes. Suppose that block size is 128 bytes and there are thus $5,120$ blocks. If $\frac{m}{d} = 1\%$, i.e., $m = 51$ blocks, with $t = 512$, detection probability amounts to about 99.94%. This represents an acceptable trade-off for applications where the advantage of MAC-ing $\frac{1}{10}$-th of verifier memory outweighs the 0.06% chance of residual malicious code and/or data. Figure 4 plots the probability t for different values of m.

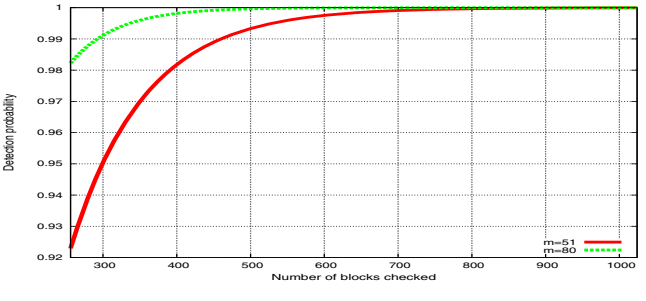


Fig. 4. Probability of detecting memory modifications for # of checked blocks varying between 256 (5%) and 1024 (20%)

5.2 Optimizing Code Update

Recall that, in the base case of Figure 3, round three corresponds to code update. Although, in practice, code size is likely to be less than n, receiving and storing entire code is a costly step. This motivates the need for shortcuts. Fortunately, there is one effective and obvious shortcut. The main idea is to replace a random $(n-k)$-bit string with the same-length encryption of new code under some key K'. This way, after round two (whether as in the base case or optimized as in the previous section), $\mathcal{V}$ sends K' to $\mathcal{P}$ which uses K' to decrypt the code. The resulting protocol is shown in Figure 5.

[1]	$\mathcal{P}$	$\longleftarrow$ $\mathcal{V}$	:	$R_1, ..., R_n$
[2]	$\mathcal{P}$	$\longrightarrow$ $\mathcal{V}$	:	$MAC_K(R_1, ..., R_{n-k})$
[3.1]	$\mathcal{P}$	$\longleftarrow$ $\mathcal{V}$	:	K'
[3.2]	$\mathcal{P}$:	$C_1, ..., C_{n-k} = D_{K'}(R_1, ..., R_{n-k})$, where $D()$ is decryption and $C_1, ..., C_{n-k}$ is new code		
[4]	$\mathcal{P}$	$\longrightarrow$ $\mathcal{V}$	:	ACK

Fig. 5. Optimized Protocol

Note again that, since we assume no communication interference and no packet loss or communication errors, the last round is just an acknowledgement, i.e., not a function of decrypted code or K'. This optimization does not affect the security of our scheme if a secure block cipher is used, since encryption of code $[C_1, ..., C_{n-k}]$ with key K' is random and unpredictable to the prover before key K' is disclosed. Hence, the proof in Section 5.1 also holds for this optimized version of the protocol.

6 Implementation and Performance Considerations

In order to estimate its performance and power requirements, we implemented PoSE on the ATMEGA128 micro-controller mounted on a MicaZ sensor. Characteristics of this sensor [11] platform relevant to our scheme are: 648KB total programmable memory; 250kbps data rate for the wireless communication channel. The total memory is divided into: 128KB of internal flash; 4KB of internal SRAM; 4KB of configuration EEPROM; 512KB of external flash memory. The application was implemented on TinyOS.

6.1 Performance Evaluation

Three main metrics affect the performance of our scheme and for this reason will be evaluated separately: communication speed; read/write memory access time; computation speed of the message authentication code.

Communication channel throughput. The maximum claimed throughput of TI-CC2420 radio chip, as reported in the specifications, is 250kbps, which translates to 31, 250 bytes/sec. This upper-bound is unfortunately quite unattainable and our tests show that, in a realistic scenario, throughput hovers around 11, 000 bytes/sec. The total memory available on a MicaZ is 644KB, including external and internal flash and EEPROM. Our efficient proof of erasure only requires randomness to be sent once, from the verifier to the prover. Then a realistic estimate for the transmission time of the randomness amounts to approximately 59 seconds, as was indeed witnessed in our experimental setup.

Memory Access. Another important factor in the performance of PoSE is memory access and write time. Write speed on the internal and external flash memory is $60KB/sec$ according to specifications. This estimate has also been confirmed by our experiments. Therefore, memory access accounts for only a small fraction of the total run-time.

MAC Computation. We evaluated the performance of three different MAC constructs: HMAC-MD5, HMAC-SHA1 and SkipJack in CBC-MAC. Note that, even though there are well-known attacks on MD5 that find chosen-prefix collisions [33], the short-lived nature of the integrity check needed in our protocol rules out attacks that require 2^{50} calls to the underlying compression function. Table 2(a) shows the results: in each case we timed MAC computation over 644KB of memory on MicaZ.

The fact that MD5 is the fastest is not surprising, given that, in our implementation, the code is heavily in-lined, which reduces the number of context switches for function calls while also resulting in increased code size.

6.2 Memory Usage

We now attempt to estimate the amounts of code and volatile memory needed to run PoSe. An estimate of code memory needed to run it is necessary to understand ROM size requirements. Furthermore, estimating required volatile memory is critical for the security of the protocol. In fact, in order to correctly follow the protocol, the prover needs a minimal amount of working memory. This memory can not be filled with randomness and hence $\mathcal{P}$ could use it to store arbitrary values. However, by keeping the amount of volatile memory to a *minimum* we can guarantee that $\mathcal{P}$ can not store both arbitrary values and carry on the necessary computation to complete the protocol.

Since assuring that the amount of volatile memory used in a specific implementation is difficult, one way to minimize effects of volatile memory is to include it in the computation of the keyed MAC (or send it back to $\mathcal{V}$ in the base case). Even though the contents of volatile memory are dynamic, they are entirely depended on the inputs from $\mathcal{V}$. Therefore, they are essentially deterministic. In this case, the verifier would have to either simulate or re-run the attestation routine to compute the correct (expected) volatile memory contents.

Table 2. MAC constructions on MicaZ

(a) Energy consumption and time

MAC	Time (sec)	Energy ($\mu J/byte$)
HMAC-MD5	28.3	1
HMAC-SHA1	95	3.5
Skipjack CBC-MAC	88	3.1

(b) Code and working memory required

MAC	ROM (bytes)	RAM (bytes)
HMAC-MD5	9,728	110
HMAC-SHA1	4,646	124
Skipjack CBC-MAC	2,590	106

Code Size. To estimate code size, we implemented the base case PoSE protocol in TinyOS. It transmits and receives over the wireless channel using Active Messages. The entire application takes 11,314 bytes of code memory and 200 bytes of RAM. RAM is needed to hold the necessary data structures along with the stack. Our implementation used regular TinyOS libraries and compiler. Careful optimization would most likely reduce memory consumption.

In the optimized version of PoSE, we also need a MAC housed in ROM. Table 2(b) shows the amount of additional memory necessary to store code and data for various MAC constructions. Finally, Table 3 shows the size of both code and working memory for all presented above.

The reason for MD5 having a larger memory footprint is because, as discussed above, the implementation we used is highly inlined. While this leads to better performance (faster code) it also results in a bigger code size.

Table 3. Code and volatile memory size

Protocol	ROM (bytes)	RAM (bytes)
PoSE(Base Case)	11,314	200
PoSE-MD5	21,042	264
PoSE-SHA1	15,960	274
PoSE-SkipJack	13,904	260

Memory Mapping. In the previous discussion, we have abstracted away from specific architectures by considering a system with uniformly addressable memory space M. However, in formulating this generalization extra care must be taken: in real systems, memory is not uniform, since there can be regions assigned to specific functions, such as memory-mapped registers or I/O buffers. In the former case, changing these memory locations can result in modified registers which, in turn, might cause unintended side effects. In the latter, memory content of I/O buffers might change due to asynchronous and non-deterministic

events, such as reception of a packet from a wireless link. When we refer to prover memory M, we always exclude these special regions of memory. Hence both the verifier and the prover have to know a mapping from the virtual memory M to the real memory. However, this mapping can be very simple, thus not requiring a memory management unit. For example on the Atmel ATMEGA128, as used in the MicaZ, the first 96 bytes of internal SRAM are reserved for memory-mapped register and I/O memory.

6.3 Read-Only Memory

PoSE needs a sufficient amount of read-only memory (ROM) to store the routines (read-and-send, receive-and-write and, in its optimized version, MAC) needed to run the protocol. While the use of mask ROM has always been prominent in embedded devices, recently, due to easier configuration, flash memory has supplanted cheaper mask ROM.

However, there are other means to obtain read-only memory using different and widely available technologies. For example, ATMEGA128 [5] allows a portion of its flash memory to be *locked* in order to prevent overwriting. Even though the size of this lockable portion of memory is limited to 4KB, this feature shows the feasibility of such an approach on current embedded devices. Note that, once locked, the memory portion cannot be unlocked unless an external JTAG debugger is attached to unset the lock bit.

Moreover, ATMEGA128 has so-called *fuse bits* that, once set, cannot be restored without unpacking the MCU and restoring the fuse. This clearly illustrates that the functionalities needed to have secure read-only memory are already present in commodity hardware.

Another way to achieve the same goal would be to use one-time programmable (OTP) memory. Although this memory is less expensive than flash, it still offers some flexibility over conventional ROM.

7 Limitations and Challenges

In this paper, our design was guided mainly by the need to obtain clear security guarantees and not to maximize efficiency and performance. Specifically, we aimed to explore whether remote attestation without secure hardware is possible at all. Hence, PoSE-based protocols (even the optimized ones) have certain performance drawbacks. In particular, the first protocol round is the most resource-consuming part of all proposed protocols. The need to transmit, receive and write n bits is quite expensive. It remains to be investigated whether it is possible to achieve same security guarantees with a more efficient design.

In terms of provable security, our discussion of Proofs-of-Secure-Erasure (PoSE-s) has been rather light-weight. A more formal treatment of the PoSE primitive needs to be undertaken. (The same holds for the multi-prover extension described in Appendix A).

Furthermore, we have side-stepped the issue of verifier authentication. However, in practice, $\mathcal{V}$ must certainly authenticate itself to $\mathcal{P}$ before engaging in any PoSE-like protocol. This would entail additional requirements (e.g., storage of $\mathcal{V}$'s public key in $\mathcal{P}$'s ROM) and raise new issues, such as exactly how (possibly compromised) $\mathcal{P}$ can authenticate $\mathcal{V}$?

Another future direction for improving our present work is by giving the adversary the capability of attacking our protocol with another device (not just the actual prover). This device would try to aid the prover in computing the correct responses in the protocol and pass the PoSE. Assuming wireless communication, one way for verifier to prevent the prover from communicating with another malicious device is is by actively jamming the prover.

Jamming can be used to selectively allow the prover to complete the protocol, while preventing it from communicating with any other party. Any attempt to circumvent jamming by increasing transmission power can be limited by using readily available hardware. For example, the CC2420 radio, present on the MicaZ, supports transmission power control. Thresholds can be set for the Received Signal Strength (RSS), RSS_{min} and RSS_{max}, such that only frames with $RSS \in [RSS_{min}, RSS_{max}]$ are accepted and processed. This is enforced in hardware by the radio chip. Hence, if the verifier wants to make sure that the prover does not communicate, it can simply emit a signal with $RSS > RSS_{max}$. This approach is similar to the one employed in [22], albeit, in a different setting.

8 Conclusions

This paper considered secure erasure, secure code update and remote attestation in the context of embedded devices. Having examined prior attestation approaches (both hardware- and software-based), we concluded that the former is too expensive, while the latter – too uncertain. We then explored an alternative approach that generalized the attestation problem to remote code update and secure erasure. Our approach, based on Proofs-of-Secure-Erasure relies neither on secure hardware nor on tight timing constraints. Moreover, although not particularly efficient, it is viable, secure and offers some promise for the future. We also assess the feasibility of the proposed method in the context of commodity sensors.

Acknowledgments

We thank ESORICS'10 anonymous reviewers for their comments. We are also grateful to Ivan Martinovic, Claude Castelluccia, Aurelien Francillon and Brian Parno for their comments on early drafts of this paper. Research presented in this paper was supported, in part, by the European Commission-funded STREP WSAN4CIP project, under grant agreement ICT-225186. All views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsement of the WSAN4CIP project or the European Commission.

References

1. Anderson, R., Kuhn, M.: Tamper resistance - a cautionary note. In: Proceedings of the Second USENIX Workshop on Electronic Commerce (1996)
2. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A secure and reliable bootstrap architecture. In: SP 1997: Proceedings of the 1997 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 65. IEEE Computer Society, Los Alamitos (1997)
3. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 598–609. ACM, New York (2007)
4. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: SecureComm 2008: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, pp. 1–10. ACM, New York (2008)
5. Atmel Corporation. Atmega128 datasheet, http://www.atmel.com/atmel/acrobat/doc2467.pdf
6. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
7. Bellare, M., Kilian, J., Rogaway, P.: The security of cipher block chaining. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 341–358. Springer, Heidelberg (1994)
8. Cachin, C., Maurer, U.: Unconditional security against memory-bounded adversaries. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 292–306. Springer, Heidelberg (1997)
9. Castelluccia, C., Francillon, A., Perito, D., Soriente, C.: On the difficulty of software-based attestation of embedded devices. In: CCS 2009: Proceedings of 16th ACM Conference on Computer and Communications Security (November 2009)
10. Choi, Y.-G., Kang, J., Nyang, D.: Proactive code verification protocol in wireless sensor network. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part II. LNCS, vol. 4706, pp. 1085–1096. Springer, Heidelberg (2007)
11. Crossbow Technology Inc. Micaz datasheet, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf
12. England, P., Lampson, B., Manferdelli, J., Peinado, M., Willman, B.: A trusted open platform. IEEE Computer 36(7) (2003)
13. Flammini, F., Gaglione, A., Mazzocca, N., Moscato, V., Pragliola, C.: Wireless sensor data fusion for critical infrastructure security. In: CISIS 2008: Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems (October 2008)
14. Francillon, A., Castelluccia, C.: Code injection attacks on Harvard-architecture devices. In: Ning, P., Syverson, P.F., Jha, S. (eds.) CCS 2008: Proceedings of the 15th ACM Conference on Computer and Communications Security. ACM, New York (2008)
15. Goodspeed, T.: Exploiting wireless sensor networks over 802.15.4. In: Texas Instruments Developper Conference (2008)

16. Gratzer, V., Naccache, D.: Alien vs. quine. IEEE Security and Privacy 5, 26–31 (2007)
17. Hu, W., Corke, P., Shih, W.C., Overs, L.: secfleck: A public key technology platform for wireless sensor networks. In: Roedig, U., Sreenan, C.J. (eds.) EWSN 2009. LNCS, vol. 5432. Springer, Heidelberg (2009)
18. Jakobsson, M., Johansson, K.-A.: Assured detection of malware with applications to mobile platforms. Tech. rep., DIMACS (February 2010), http://dimacs.rutgers.edu/TechnicalReports/TechReports/2010/2010-03.pdf
19. Juels, A., Kaliski Jr., B.S.: Pors: proofs of retrievability for large files. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 584–597. ACM Press, New York (2007)
20. Kennell, R., Jamieson, L.H.: Establishing the genuinity of remote computer systems. In: SSYM 2003: Proceedings of the 12th conference on USENIX Security Symposium, pp. 21–21. USENIX Association, Berkeley (2003)
21. Kil, C., Sezer, E.C., Azab, A.M., Ning, P., Zhang, X.: Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In: DSN 2009: Proceedings of the 39th IEEE/IFIP Conference on Dependable Systems and Networks (June 2009)
22. Martinovic, I., Pichota, P., Schmitt, J.B.: Jamming for good: a fresh approach to authentic communication in wsns. In: WiSec 2009: Proceedings of the Second ACM Conference on Wireless Network Security, pp. 161–168. ACM, New York (2009)
23. Park, T., Shin, K.G.: Soft tamper-proofing via program integrity verification in wireless sensor networks. IEEE Trans. Mob. Comput. 4(3) (2005)
24. Roman, R., Alcaraz, C., Lopez, J.: The role of wireless sensor networks in the area of critical information infrastructure protection. Inf. Secur. Tech. Rep. 12(1), 24–31 (2007)
25. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: SSYM 2004: Proceedings of the 13th Conference on USENIX Security Symposium, pp. 16–16. USENIX Association, Berkeley (2004)
26. Seshadri, A., Luk, M., Perrig, A.: SAKE: Software attestation for key establishment in sensor networks. In: Nikoletseas, S.E., Chlebus, B.S., Johnson, D.B., Krishnamachari, B. (eds.) DCOSS 2008. LNCS, vol. 5067, pp. 372–385. Springer, Heidelberg (2008)
27. Seshadri, A., Luk, M., Perrig, A., van Doorn, L., Khosla, P.: SCUBA: Secure code update by attestation in sensor networks. In: WiSe 2006: Proceedings of the 5th ACM Workshop on Wireless Security, ACM Press, New York (2006)
28. Seshadri, A., Luk, M., Shi, E., Perrig, A., van Doorn, L., Khosla, P.: Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In: SOSP '05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. ACM, New York (2005)
29. Seshadri, A., Perrig, A., van Doorn, L., Khosla, P.K.: SWATT: SoftWare-based ATTestation for embedded devices. In: IEEE Symposium on Security and Privacy. IEEE Computer Society, Los Alamitos (2004)
30. Shacham, H.: The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, New York (2007)

31. Shaneck, M., Mahadevan, K., Kher, V., Kim, Y.: Remote software-based attestation for wireless sensors. In: Molva, R., Tsudik, G., Westhoff, D. (eds.) ESAS 2005. LNCS, vol. 3813, pp. 27–41. Springer, Heidelberg (2005)
32. Shankar, U., Chew, M., Tygar, J.D.: Side effects are not sufficient to authenticate software. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
33. Stevens, M., Lenstra, A., Weger, B.: Chosen-prefix collisions for md5 and colliding x.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
34. Trusted Computing Group. Specifications
35. Yang, Y., Wang, X., Zhu, S., Cao, G.: Distributed software-based attestation for node compromise detection in sensor networks. In: SRDS. IEEE Computer Society, Los Alamitos (2007)

A: Dealing with Multiple Devices

Thus far, in this paper we assumed one-on-one operation: one prover and one verifier. However, in practice, embedded devices are often deployed in groups and in relative proximity (and density) among them, e.g., Wireless Sensor Networks (WSNs). If the task at hand is to perform code attestation or update of multiple proximate devices, our approach can be easily extended to support this setting and, at the same time, obtain a significant efficiency gain. The main observation is that, if the verifier can communicate with $t > 1$ devices at the same time (i.e., via broadcast), download of randomness in the first round of our protocol – which represents the most time-consuming part of the protocol – can be done in parallel for all devices within the verifier's communication range. Of course, in order to receive replies the verifier has to be within communication range of all t provers.

At the same time, parallel code update of multiple devices prompts us to re-examine the adversarial model. In the one-on-one setting, it makes sense to assume radio silence, i.e., the fact that, during the protocol, the prover device is not communicating with any party other than the verifier, and no other (third) device is transmitting any information that can be received by either the prover or the verifier. Note that the term *adversary* refers collectively to any compromised devices running malicious code as well as any extraneous devices physically controlled by the adversary. However, the one-on-one setting does not preclude the adversary from *over-hearing* communication between the prover and the verifier, i.e., eavesdropping on protocol messages. We claim that this has no bearing on security, since each protocol involves a distinct stream of randomness.

In contrast, when multiple parallel (simultaneous) provers are involved, the situation changes. In particular, we need to take into account that possibility that one or more of the t provers is running malicious code. Suppose that a malicious code-running prover $\mathcal{P}_x$. Then, if we naïvely modify our protocol from Figure 5 as shown in Figure 6, the resulting protocol is insecure. The reason for the lack of security is simple: suppose that $\mathcal{P}_x$ ignores the message in round 1.0 and does not store verifier-supplied randomness. Then, in round 2.0, $\mathcal{P}_x$ over-hears and

records a reply – $MAC_K(R_1, ..., R_{n-k})$ – from an honest prover $\mathcal{P}_1$. Clearly, $\mathcal{P}_x$ can just replay this MAC and thus convince the verifier of having received and stored the randomness from message 1.0.

Assume reachable provers $\mathcal{P}_1, ..., \mathcal{P}_t$ and $1 < j \leq t$

[1.0] $\mathcal{P}_j \Longleftarrow \mathcal{V}$: $R_1, ..., R_n$ where $K = R_{n-k}, ..., R_n$
[2.0] $\mathcal{P}_j \longrightarrow \mathcal{V}$: $MAC_K(R_1, ..., R_{n-k})$
[3.1] $\mathcal{P}_j \Longleftarrow \mathcal{V}$: k'
[3.2] $\mathcal{P}_j$: $C_1, ..., C_{n-k} = D_{K'}(R_1, ..., R_{n-k})$
[4.0] $\mathcal{P}_j \longrightarrow \mathcal{V}$: ACK

Fig. 6. Insecure Multi-Prover Protocol

The above discussion leads us to amend the adversarial model as follows: the adversary is allowed to record any portion of the protocol. However, for fear of being detected, it is not allowed to transmit anything that is not part of the protocol. In particular, during the protocol, none of the (potentially compromised) t provers can transmit anything that is not part of the protocol. And, no extraneous entity can transmit anything to any of the t provers.

Assume reachable provers $\mathcal{P}_1, ..., \mathcal{P}_t$ and $1 < j \leq t$

[1.1] $\mathcal{P}_j \Longleftarrow \mathcal{V}$: $R_1, ..., R_n - k$
[1.2] $\mathcal{P}_j \longleftarrow \mathcal{V}$: K_j where $k = |K_j|$
[2.0] $\mathcal{P}_j \longrightarrow \mathcal{V}$: $MAC_{K_j}(R_1, ..., R_{n-k})$
[3.1] $\mathcal{P}_j \Longleftarrow \mathcal{V}$: K'
[3.2] $\mathcal{P}_j$: $C_1, ..., C_{n-k} = D_{K'}(R_1, ..., R_{n-k})$
[4.0] $\mathcal{P}_j \longrightarrow \mathcal{V}$: ACK

Fig. 7. Multi-Prover Protocol

The modified (and secure) protocol that supports $t > 1$ provers is shown in Figure 7. The main difference from the insecure version in Figure 6 is the fact that random and distinct keys K_j are generated and sent to each prover $\mathcal{P}_j$.

This protocol guarantees that, in the context of the modified adversarial model, each prover has to independently store the randomness sent by the verifier. Since, the key sent by the verifier is unique to each prover and so is the MAC computation. This assertion clearly needs to be substantiated via a proof of security. This issue will be addressed as part of our future work.

Caveat: We acknowledge that, while the multi-prover protocol achieves better performance through parallelization, it does not improve energy consumption on each prover. We plan to explore this issue as part of our future work.

D(e|i)aling with VoIP: Robust Prevention of DIAL Attacks

Alexandros Kapravelos, Iasonas Polakis, Elias Athanasopoulos,
Sotiris Ioannidis, and Evangelos P. Markatos

Institute of Computer Science,
Foundation for Research and Technology Hellas, Greece
{kapravel,polakis,elathan,sotiris,markatos}@ics.forth.gr

Abstract. We carry out attacks using Internet services that aim to keep telephone devices busy, hindering legitimate callers from gaining access. We use the term *DIAL (Digitally Initiated Abuse of teLephones)*, or, in the simple form, *Dial attack*, to refer to this behavior. We develop a simulation environment for modeling a Dial attack in order to quantify its full potential and measure the effect of attack parameters. Based on the simulation's results we perform the attack in the real-world. By using a Voice over IP (VoIP) provider as the attack medium, we manage to hold an existing landline device busy for 85% of the attack duration by issuing only 3 calls per second and, thus, render the device unusable. The attack has zero financial cost, requires negligible computational resources and cannot be traced back to the attacker. Furthermore, the nature of the attack is such that anyone can launch a Dial attack towards any telephone device.

Our investigation of existing countermeasures in VoIP providers shows that they follow an *all-or-nothing* approach, but most importantly, that their anomaly detection systems react slowly against our attacks, as we managed to issue tens of thousands of calls before getting spotted. To cope with this, we propose a flexible anomaly detection system for VoIP calls, which promotes fairness for callers. With our system in place it is hard for an adversary to keep the device busy for more than 5% of the duration of the attack.

1 Introduction

The Internet is a complicated distributed system that interconnects different kinds of devices and interfaces with other types of networks. Traditionally, computer security deals with attacks that are launched from Internet hosts and target other Internet hosts. However, the penetration of Internet services in everyday life enables threats originating from Internet hosts and targeting non Internet infrastructures. Such a non Internet infrastructure is the telephony network, a vital commodity.

The ever increasing number of households that adopt Voice over IP technology as their primary telephony system, demonstrates our shifting towards a digitally

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 663–678, 2010.

interconnected community. According to estimations, IP communication subscribers will reach more than 1.8 billion worldwide by 2013 [2]. While this new technology coexists with the old technology, new methods for their interaction emerge. Today, an Internet user can place calls to anywhere in the world reaching anyone that has a telephone device and take advantage of all characteristics inherent in such digital technologies, thus introducing new threats against traditional telephony systems.

In this paper, we explore the feasibility of an attack using Internet services and targeting regular landline or cellular phones. We seek to characterize the parameter values that will make the attack effective and also the means to mitigate it. Our key contributions are the following:

Dial Attack. We develop an empirical simulation in order to explore the potential effectiveness of Dial attacks. Through the simulated environment we identify and quantify all of the attack's fundamental properties. Using experimental evaluation with existing telephone lines, we demonstrate that an *attacker manages to render an ordinary landline device unusable, holding it busy for 85% of the attack period by issuing only 3 calls per second* . The attack requires no financial resources, negligible computational resources and cannot be traced back to the attacker.

Defenses. We seek to reveal existing countermeasures through reverse engineering of real-world VoIP providers. Our findings suggest that current schemes are not efficient since they follow an *all-or-nothing* approach. We develop and analyze a server-side anomaly detection system for VoIP traffic, which significantly reduces the attack impact. With our defense system deployed, the attacker *can no longer hold the line busy for more than 5% of the attack period.*

This paper is organized as follows. We analyze our motivations in Section 2. In Section 3 we present a threat model and a potential attack in a simulated environment. We carry out the attack against a real landline device in Section 4. In Section 5 we present existing countermeasures and introduce our anomaly detection system, together with experiments that show how effectively our system mitigates the attack. Finally, we review prior work in Section 6 and conclude in Section 7.

2 Motivation

In this section we present the basic motivations that drove us to explore this area and led to the creation of this paper. We explore our motivation in terms of *goal* and *attack platform.*

Goal. The traditional communication through the telephone network has become an important commodity. Take into account, that over 300 billion domestic calls to landlines were served inside the US alone in 2005 according to the FCC[5]. Our argument is that *access to a telephone device is vital for humans.*

Considering the importance of the service, an adversary may target a telephone device in order to harm a user. Prohibiting users from accessing certain services has been done in the recent past. For example, a significant part of computer viruses disrupt Internet connectivity. The impact of such an attack can be enormous, either life threatening (targeting a fire-fighting station during a physical disaster), or financial (targeting a business, like pizza delivery, or hindering a bank to authenticate a transfer request [6]) or simple disturb someone.

Our research is composed by two complementary goals. The first goal is to find out if it is possible to render a telephone device unusable. We want to achieve this goal with no financial resources, negligible computational resources and without being traceable back to the attacker. The second goal is to design and build technologies for protecting users from attacks that target telephones. We want to achieve this goal with minimal deployment effort, minimal user interference and by using existing well-known technologies.

Attack platform. In order to achieve our goals we use VoIP providers as attack platforms against telephone devices. Our choice was driven by various reasons. First, we wanted an attack platform, which is affordable and easy to access. There are hundreds of free VoIP providers, which permit users to access any landline device with no cost at all or mobile devices with a minimal cost. Second, we wanted to be able to completely automate the attack and have enough flexibility in fine tuning the call placement. Most VoIP providers support the SIP protocol [19] which met our expectations. Third, we wanted to perform the attack anonymously. The very nature of VoIP technology allows a caller to hide his true identity. And finally, we wanted to launch the attack from a PC. The fact that our attack platform is already provided by the industry and anyone can use it to launch the attack motivates us highly to explore the area as a precaution from future exploitation of VoIP services.

One can argue, that parts of the attack described in this paper are well-known or can be carried out, manually, by performing an excessive amount of dialing. As far as the novelty is concerned, to the best of our knowledge, this paper is the first one to perform and evaluate a real automated attack *directly* targeting a telephone device. As far as manual dialing is concerned, we already enlisted the four reasons, which drove us to select VoIP as the attack platform. These four reasons, reveal characteristics of a platform far superior to humans performing manual dialing.

3 Attack Overview

In this section we present the fundamental properties of the attack we developed. We start by describing our attack in detail; we specify the threat model, the adversary's overall goal and list all the assumptions we have made. We develop a simulated environment in order to carry out the attack virtually. Based on our findings in this section, we proceed and develop the actual attack prototype in the next section.

3.1 Attack Description

The goal of the attacker is to render a telephone device unusable with zero financial cost. This can be achieved by injecting a significant number of *missed calls* towards a victim telephone device. A call is considered missed, if it is hanged up prior to the other end answering it. By placing the calls correctly in the network, the attacker can keep the target continuously busy and, thus, prevent other users from accessing the telephone device. Even though many VoIP providers allow calls to landlines free of charge, we designed our attack in a way to be able to attack cell phones even if such calls are not free. By hanging-up the placed calls on time, the adversary manages to launch the attack cost free. Even if the target telephone device is answered the attack does not degrade, but rather augments, taking into account that the resource is still in busy state.

The attack in principle consists of a resource R, an attack medium M and calling modules. The resource represents a telephone device and has two states; it can either be available or busy. The attack medium simulates the behavior of a VoIP provider; it receives requests and queries the resource in order to acquire it. A calling module places such requests to the attack medium at a configurable rate.

The proposed attack is based on some important assumptions. First, we assume that M is unreliable, meaning that communication messages may be lost, dropped or delayed. However, we assume that all faults in M are stabilized in the long run. Thus, we do not implement message faults for M in the simulated environment. Second, we assume that R does not support direct querying, or at least it supports it partially. There is no way to directly retrieve all states of R. However, it is possible to implement detection by analyzing parts of the communication messages. Third, we assume that, when no attack is taking place, the calling rate follows a Poisson distribution ($\lambda = 10$) [10]. When an attack is taking place, the calling rate significantly varies from the Poisson distribution. Finally, we make no assumptions about the routing latency for call placement, i.e. the time it takes for a request to reach R through M, or the release time of R after call termination. Instead, we perform real experiments to collect representative approximations of these quantities (see next section).

3.2 Simulation

Before starting experimenting with real calls in the wild, we performed a series of controlled experiments in a simulated environment. Based on the attack description we just presented, we developed a multi-threaded simulator where we instantiated a virtual calling module with an *aggressive behavior* to represent the attacker and one with a *non-aggressive behavior* to represent a legitimate caller. Then, we modeled the resource using a data structure that allows the module to obtain exclusive access with a certain probability. Each calling module behaves similar to a VoIP caller, *i.e.* it attempts to connect to the resource R through an attack medium M. In principle, M simulates a VoIP provider and R a telephone device.

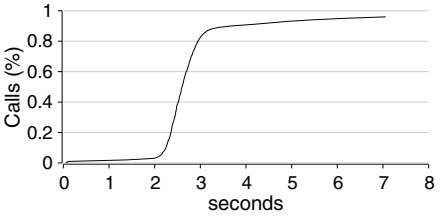


Fig. 1. Cumulative distribution of routing latency times for call placement

In order to simulate the virtual calls in a realistic fashion we collected empirical values of durations from real call placement and hang-up operations. We issued 7,300 calls through a real VoIP provider over the time period of one week (see Figure 1). In this way, we collected representative routing latencies of call placements at various times and days of a typical week. The simulator maintains a pool with the 7,300 routing latencies and uses one, randomly, each time a virtual call attempt takes place. Unfortunately, we could not follow a similar approach for the hang-up operation, since it is hard to detect representative values for hang-up times of a real VoIP provider. However, we used the following approach, which we consider quite realistic. We injected pairs of call placements and hang-up operations in a real VoIP provider. We initially started injecting the pairs back-to-back. The result was that one of the two calls always reached the telephone device when it was in the busy state. In other words, the VoIP provider could not complete the hang-up operation of the first arrived call, before the second arrived. We started increasing the gap between the call pair, until we could measure that both calls had reached the telephone device in available state. We managed to successfully issue over 1,000 such call pairs with this property. The gap times ranged from 1 to 2 seconds. We consider this time window a realistic window for a hang-up operation. Thus, we modeled the virtual hang-up operation accordingly. Each virtual call hang-up operation takes from 1 to 2 seconds to restore R's state back to available.

Based on the above configuration we issued four 1-hour simulation runs, each one having an aggressive calling module placing virtual calls, with different intervals. We used intervals ranging from 0.01 to 5 seconds. Concurrently a legitimate module tried to acquire R following a Poisson distribution with $\lambda = 10$.

We examine the results of our experiments in terms of the aggressive calling module's success in acquiring resource R, the virtual call status distribution of the aggressive calling module and how many times the legitimate module succeeded in acquiring resource R. The rate of successful R acquisitions an aggressive thread managed to issue is presented in Figure 2. The best we could achieve was more than 39 acquisitions per minute. In the real-world, this result translates into more than 39 ringing calls per minute; a severe attack rate that would render the telephone device unusable. In Figure 3 we examine the call

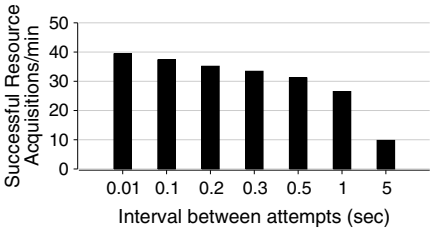


Fig. 2. Rate of successful resource acquisitions managed by an aggressive calling module in simulation environment

status distribution of all virtual call placements the aggressive calling module managed to issue. Observe that as the interval reduces, the amount of failures in acquiring R increases rapidly. Practically, there is no benefit in reducing the interval below 0.3 seconds.

The performance of the legitimate module is depicted in Figure 4. We also plot the results for the first 10 minutes of the experiment's duration in this case. First, observe that the legitimate module fails to acquire R for almost 85% of the simulation duration at the interval of 0.3 seconds, while the aggressive module issued less than 20,000 attempts. By reducing the intervals down to 0.01 seconds, the legitimate module is completely prevented from acquiring R, with the downside of requiring almost 200,000 more issued attempts to achieve just 15% more failed resource acquisitions compared to the interval of 0.3 seconds. Second, we can see that the first 10 minutes approximate the result of the full duration (1 hour) of the simulation, with tolerable error (from below 1% to 1.5%) in most cases. An exception occurs only in the case of the 1 second interval, where the difference is about 8%.

Our simulation experiments confirm our intuition for a potential threat against telephone devices. In addition to this, they highlight that only 3 calls per second are needed to render a device unusable and that the error between 1 hour and 10 minutes long experiments is tolerable.

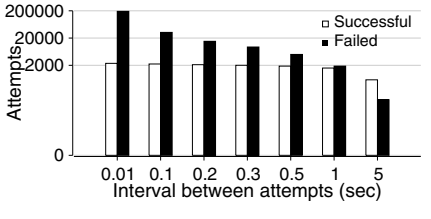


Fig. 3. Distribution of all acquire attempts issued by an aggressive calling module in simulation environment

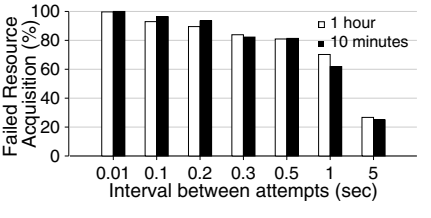


Fig. 4. Percentage of failed resource acquisitions for the legitimate module which models a legitimate caller in simulation environment

4 Attack Evaluation

Based on the simulated studies we carried out in the previous section, we present an attack prototype. Our aim is to reach the performance we achieved in the simulated environment, using an existing system which tries to acquire an actual telephone device.

4.1 Attack Prototype

Our attack prototype implementation uses VoipDiscount [7] as an attack medium, which uses the Session Initiation Protocol (SIP) [19] for remote communication. As SIP is the most common used protocol among VoIP service providers, our prototype implementation is not limited to VoipDiscount but could be applied using any different provider.

We implemented caller modules, which communicate with M, in our case the VoIP provider, using the SIP protocol and exchange invite and termination messages. We used the Python programming language and the `pjsip`[1] library which provides an implementation of the SIP protocol. We developed two types of callers: (a) an attacker caller and (b) a legitimate caller. The attacker caller places calls one after the other, trying to keep the telephone device busy continuously. The legitimate caller places calls following the Poisson distribution ($\lambda = 10$).

Recall, from section 3, that we assumed that the resource does not support querying, or it supports partial querying. Indeed, the telephone device does not support querying and thus there is no easy way to track down the status of the device, i.e. if it is in ringing or busy state. Although, SIP supports querying the status of a placed call, many providers do not implement this feature. The one we used is among them. Specifically, we can retrieve that the line is busy, using a SIP operation, but we can not retrieve a ringing status. To overcome this issue we implemented a detector module, based on a Fast Fourier Transformation of the incoming audio signal. This way we analyze the frequency of the audio signal and detect a ringing tone when we observe signals at 420 Hz. Notice that this approach successfully recognizes the ringing tone, since the tone has a constant frequency. Having immediate access to the ringing status is vital for the attack, since we want to achieve the attack with zero financial resources. We want to keep the telephone device busy by injecting short time lived calls (i.e. missed calls). For the generation of a missed call, the call has to be terminated immediately after the first ringing tone.

4.2 Real World Experiments

We conducted several real world experiments over the period of eight months using a landline device located in our lab as a victim. For the presentation of this section we issued a series of runs over the duration of 1 week. This subset of runs is consistent with our overall experimental results. For each configuration

[1] PJSIP, `http://www.pjsip.org/`.

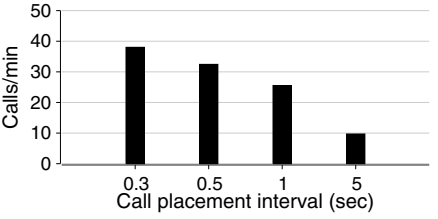


Fig. 5. Rate of ringing calls managed by an adversary

we issued 6 runs each with a 10 minute duration with intervals ranging from 0.3 to 5 seconds. Recall from section 3 that the first ten minutes of each simulation run approximate the result of the full duration (1 hour) of the simulation, with tolerable error (from below 1% to 1.5%) in most cases. Thus we face the following trade-off: conducting more short-lived or less long-lived real world experiments. We chose the first approach, so as to be flexible enough to conduct a larger experimental base.

As was the case with the simulation, we are interested in three measurements: (i) the call rate of the attacker, (ii) the call status distribution of the attacker, and (iii) the probability for a legitimate user to acquire the resource, while an attack is taking place.

We present the call rate achieved by the attacker in Figure 5. Observe, that the results are highly consistent with the simulated ones (see Fig. 2). The adversary has managed to issue almost 40 ringing calls/minute for a calling placement interval of 0.3 seconds.

We present the distribution of all call attempts by the attacker in Figure 6. Again, the results are highly consistent with the simulated ones (see Fig. 3). Note, that as the call placement interval reduces, the fraction of busy calls increases, having a negative impact on the attack. Another side-effect of shorter call placement intervals is failed calls. A call is considered failed, when no ringing or busy status is identified after 10 seconds from call placement.

Finally, in Figure 7 we present the percentage of busy calls received by a legitimate caller, while the target telephone device was under attack. Observe,

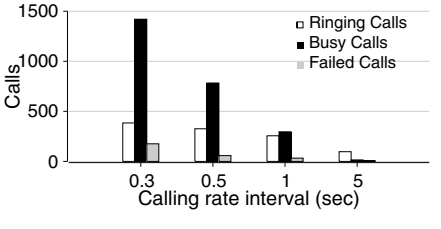


Fig. 6. Distribution of all calls issued by an adversary

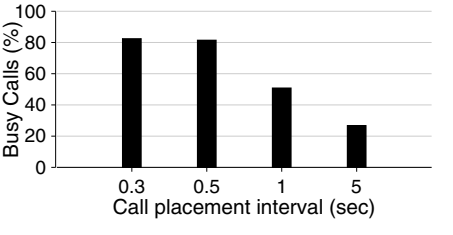


Fig. 7. Percentage of busy calls received by a legitimate caller, while the target telephone device was under attack

that the adversary managed to hold the target landline device busy for 85% of the attack duration, preventing access, for most of the time, to the legitimate caller.

4.3 Attack Impact

In our real world experiments, we managed to hold an existing landline busy for 85% of the attack duration, by issuing only 3 calls per second. By aggressively issuing calls, an attacker targeting the telephone centers of critical infrastructures such as police or fire-fighting stations and hospitals can completely disrupt their operation and create life threatening situations. By issuing dozens or hundreds of calls an attacker can hinder legitimate users from accessing the call centers of critical services. Taking into account that these services are vital to our society, *any* threat against them must be seriously considered, and mechanisms for protection should be designed and employed.

4.4 Attacker's Anonymity

An important aspect of *Dial attacks* is that they cannot be traced back to the attacker. This is achieved by having two layers of anonymity. The first is provided by the part of traditional telephony network, where only the VoIP provider's Caller ID is revealed. In our experiments, the Caller ID was *Unknown*, requiring the use of law enforcement in order to find the source of the calls. The second layer is the communication with the VoIP provider. The only way that the VoIP provider can track the attacker is by her IP address. In order to remain anonymous, the call requests must be placed from a safe IP address, *e.g.* through an anonymization proxy or with the use of a botnet.

5 Countermeasures

In this section we investigate existing countermeasures currently employed by VoIP providers. We present a study about Skype, a leading provider of VoIP services, VoipUser[8] and VoipDiscount[7], two representative VoIP providers. Based on our analysis, we propose and implement an anomaly detection system that promotes fairness to callers and is able to successfully mitigate the attack outlined in this paper.

5.1 Existing Countermeasures

Skype. Skype is a popular VoIP provider with more than 400 million user accounts and capable of serving 300,000 simultaneous calls without any service degradation [3,1]. Skype internally uses an anomaly detection system, whose technical details are not publicly available. In order to reverse engineer part of its logic, we used four different user accounts and three different landline devices. We performed experiments with very aggressive call initialization rates against

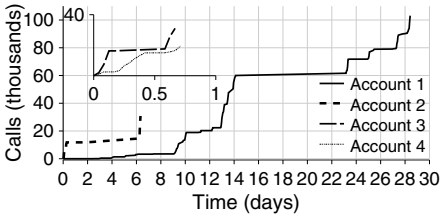


Fig. 8. Call history of each Skype account, until it is blocked

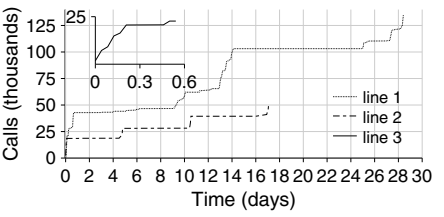


Fig. 9. Call history of each telephone line targeted through Skype, until it is blocked

our landlines. Eventually, all four accounts were blocked permanently and all three victim landline devices were permanently banned from the system. This means, that the victim landlines were further inaccessible by *any* Skype user. We refer to this policy as *all-or-nothing*, meaning that the anomaly system either permits full access or no access at all to the service.

In Figure 8 we present the cumulative time of the call history of each blocked Skype account. Our initial intuition was that Skype blocks our account when we pass a specific call-rate threshold. However, each Skype account got blocked when it exceeded a totally different threshold, indicating non-deterministic detection based on heuristics or human inspection of call logs. With the first account we placed more than *one hundred thousand* calls before the anomaly detection system spotted us. The other accounts were blocked by making a large number of calls in a very short time period. This is shown in Figure 8 by an almost vertical increase of at least fifteen thousand calls.

In addition, Skype also blocked the landline telephone numbers which we used as victims. In Figure 9 we can see the call history of these numbers. The graphs terminate at the time the blocking actually happened. The Skype service permitted us to place more than 130,000 calls to the first line we used, before blocking it. The rest of the telephone lines we used were blocked as a result of more aggressive experiments.

We consider, that the *all-or-nothing* policy of Skype's anomaly detection is highly inefficient and, most importantly, enables further abuse. We proved that the slow reaction of the anomaly detection system allowed us to issue tens of thousands of calls. This would be catastrophic for any service that is based on telephone communication. We believe that the slow reaction is a fundamental result of the *all-or-nothing* approach. The penalty is so high (i.e. permanent block), that the anomaly detection system is triggered only during occasions where there is severe abuse. An adversary, could still carry out the attack in a more stealthy fashion. We also showed that an adversary can intentionally block certain devices from Skype. All she needs is to issue a vast amount of missed calls towards the victim device for it to be completely banned from the system.

Voipdiscount. During our experiments with the Voipdiscount provider we have not observed any countermeasures. We have used their infrastructure for multiple experiments, issuing hundreds of thousands calls for over 8 months without being warned or banned.

Voipuser. We speculate that Voipuser relies on manual inspection which is not effective and cannot provide adequate defense against such attacks. After a series of initial experiments we conducted, they blocked the accounts used, as well as all other accounts we had created; note that these accounts had not been used in the attack experiments. Account bans based on the correlation of the domain of the email addresses we used for the account registrations suggest a manual process of log inspection. However, our accounts were banned after the experiments had ended, proving the inability of manual countermeasures for the early detection of such attacks.

5.2 Server Side Countermeasures

Our system is based on a detection module and a policy enforcement module. We decided to implement the detection module entirely in software, using the well-known Intrusion Detection System (IDS), Snort[18]. As far as the policy enforcement is concerned, we have two options. We can either implement it in software or in hardware. For the first case, we can use the built-in firewall functionality of Linux operating systems, `iptables`. However, this gives us poor flexibility in complex policies. On the contrary, the hardware solution gives as a range of functionalities employed by modern router devices. In order to easily perform an evaluation of various policies, we chose to use the Click router[16], which is a rich framework for testing router configurations. The Click router incorporates a wide range of elements for traffic shaping, dropping decisions and active queue management, which can also be found in most modern routers.

Detection Module. Snort is responsible for the detection. It handles user requests by monitoring all incoming traffic and flags flows that belong to hosts that initiate a large number of calls in a short amount of time. We further refer to this threshold as *abt (abuse behavior threshold)*, which is expressed in *invite*[2] requests per second per host. We have implemented, a Snort-rule similar to those for *port-scans* for detecting hosts that exceed *abt*. Whenever we have a Snort alert, the policy enforcement module is invoked, in order to mitigate the suspicious behavior.

Policy Enforcement Module. Policies are enforced over specific time windows. We refer to this quantity as *pew (policy enforcement window)*. Each policy applies an action to a host, that has been flagged suspicious by the detection module. We have implemented two different types of actions: mute and shape. The *mute action* drops all invitation messages and the *shape action* imposes a fixed rate of message delivery in a fashion that approximates a legitimate behavior.

[2] An *invite* request in SIP is associated with a call placement.

Table 1. Policies supported by our anomaly detection system

Policy	INVITE behavior	Implementation
soft-mute	Drop	`iptables`
hard-mute	Drop	Click Router
hard-shape	Fixed rate	Click Router

We implemented the *mute* action using `iptables` in Linux, by using Snort's plugin SnortSam [4]. We provide a hypothetical hardware implementation of both *mute* and *shape* actions using the emulation environment provided by Click. In Table 1 we summarize the policies we support, along with their notation.

Evaluation. In order to evaluate our anomaly detection system we performed our attack once again, but this time, both the attacker and legitimate caller were forced to pass their requests through our system. This was done at the network level, by rerouting all communication messages through a gateway that acts as an anomaly detection system.

In order to eliminate false positives we decided to use a more tolerating *abt* value, equal to 10 invitation messages per 30 seconds ($abt = 10msg/30secs$). Notice, that although this decision leaves us with no false positives (indeed, we have measured zero false positives in all experiments), relaxing *abt* is negative for the mitigation result. The attacker can become more aggressive and still remain under *abt*.

In Figure 10 we depict the effects on the attack's firepower when our policies are enabled. Each policy is applied for a time duration equal to *pew*. Notice, we do not provide results for hard-shape values for any *pew*, since the hard-shape policy is enforced for the whole attack duration. This is not explicitly forced by our detector, but it stems from the fact that the attacker does not adapt to the policy, and the *pew* is always extended.

In Figure 11 we provide a comparison of all policies along with the original attack. For each policy we state the *pew* used inside parenthesis. Observe that the attack's firepower can be reduced to 5% using *soft-mute* or up to 30% using a more relaxed policy, *hard-shape*. We consider the *shape* policy more relaxed than the *mute*, since the suspicious host is not muted and thus the policy is more tolerable in enforcing restraints on false positives.

5.3 Client Side Countermeasures

Telephone devices currently have no means of defense against a Dial attack. Our solution is based on *phone CAPTCHAs*. A CAPTCHA [22] challenge aims to distinguish between human initiated and automated actions. The core component of our platform is the Asterisk PBX, an open-source software implementation of a private branch exchange (PBX). It supports Interactive Voice Response (IVR) technology, can detect touch tones and respond with pre-recorded

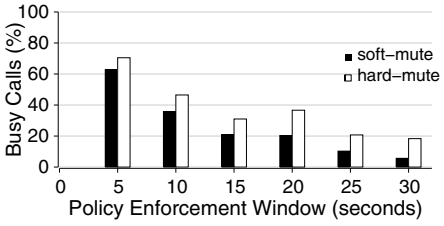


Fig. 10. Attack mitigation for soft-mute and hard-mute policies for various *pew*

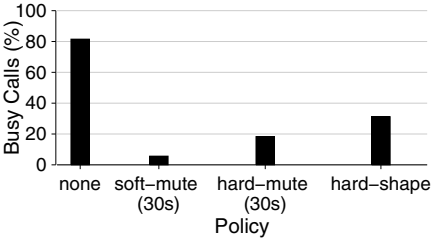


Fig. 11. A comparison of all policies along with the original attack

messages or dynamically created sound files. To handle landlines, the host machine is equipped with specialized hardware that connects it to the telephony circuit.

When an incoming call is received, Asterisk places the call in a call queue. The caller then receives a phone CAPTCHA and has a limited time to respond by using the phone's dial pad. The phone CAPTCHA test requires the caller to spell a word randomly selected from a large pool[3]. If the caller provides the correct answer, Asterisk forwards the call to its destination. Otherwise the call is dropped. The use of words greatly increases the difficulty of phone CAPTCHAs being broken by speech recognition software, compared to traditional audio CAPTCHAs that only contain digits. We intend to further explore the use of CAPTCHAs as a client side countermeasure in future work.

6 Related Work

In this work we use VoIP technology as an attack medium. Given its low access cost and its wide deployment, VoIP services have attracted a lot of attention [15]. For example, extensive research has been recently conducted on VoIP security. Wang *et al.* exploit the anonymity of VoIP calls by uniquely watermarking the encrypted VoIP flow [23]. Wright *et al.* investigate whether it is possible to determine the language of an encrypted VoIP conversation by observing the length of encrypted VoIP packets [24]. Zhang *et al.* in [25] exploit the reliability and trustworthiness of the billing of VoIP systems that use SIP [19]. Spam over Internet Telephony has also gained significant attention [11,17]. In this paper we explore new ways for abusing VoIP services as well as identifying possible defenses to this abuse.

Research for attacks to the telephony network has been carried out in the past, mostly targeting cellular networks. For example, it has been shown that a rate of only 165 SMS messages per second is capable of clogging both text and voice traffic across GSM networks in all of Manhattan [20,21]. Countermeasures to

[3] Words have been widely used in the USA to help people memorize telephone numbers by "translating" numbers into letters.

alleviate this problem are based on using weighted queues before traffic reaches the air interface, and/or more strict provisioning and partitioning resources after traffic leaves this bottleneck [12,13]. Enck *et al.* demonstrate the ability to deny voice service by just using a cable modem [12]. They claim that with the use of a medium-sized zombie network one could target the entire United States. Their work also included suggestions on how to counter SMS-based attacks. Specifically, they call for the separation of voice and data channels, increased resource provisioning, and rate limits of the on-air interfaces.

Last but not least, there are concerns in the research community about attacks that threaten the operation of emergency services. Aschenbruck *et al.* report that it is possible to peer VoIP calls to public service answering points (PSAP) [9]. This peering can have grave implications because it makes it possible to carry out DoS attacks against emergency call centers. In their work they monitored calls from a real PSAP of a fire department which serves about one million people. During emergencies the PSAP received approximately 1100 calls per 15 minutes. These calls overloaded the PSAP and the authors suggested that the high call-rate was the result of citizens constantly redialing until they got service. In their follow-up publication Fuchs *et al.* show that under heavy load at the same PSAP, up to half of the incoming calls were dropped [14].

7 Conclusion

In this paper we perform an extensive exploration of Dial attacks. Initiated by our theoretical findings, we implement a prototype and carry out the attack in the wild proving that an adversary can keep a telephone device in busy state for 85% of the attack duration by issuing only 3 calls per second. Our attack requires zero financial resources, negligible computational resources and cannot be traced back to the attacker. Considering the severity of such a threat, we explore already employed countermeasures and conclude that current VoIP infrastructures employ countermeasures based on an *all-or-nothing* approach, react slowly to possible abuse or offer no protection at all. As these defenses appear inefficient we propose an anomaly detection system for VoIP calls and demonstrate that it can mitigate Dial attacks and prevent an adversary from holding a telephone device busy for more than 5%.

Acknowledgments. We would like to thank the anonymous reviewers for their feedback and Professor Giovanni Vigna for his insightful comments. Alexandros Kapravelos, Iasonas Polakis, Elias Athanasopoulos, Sotiris Ioannidis and Evangelos P. Markatos are also with the University of Crete. Elias Athanasopoulos is funded by the Microsoft Research PhD Scholarship project, which is provided by Microsoft Research Cambridge. This work was supported in part from the FP7 project SysSec, funded by the European Commission under Grant agreement no: 257007 and by the Marie Curie Actions - Reintegration Grants project PASS.

References

1. Ebay Inc. FQ, results (2008), http://investor.ebay.com/results.cfm
2. IDC Predicts more than 1.8 billion Worldwide Personal IP Communications Subscribers by 2013, http://www.idc.com/getdoc.jsp?containerId=219742
3. Skype Fast Facts, Q4 2008, http://ebayinkblog.com/wp-content/uploads/2009/01/skype-fast-facts-q4-08.pdf
4. Snortsam, http://www.snortsam.net
5. Statistics of Communications Common Carriers 2005/2006 Edition, http://hraunfoss.fcc.gov/edocs_public/attachmatch/DOC-282813A1.pdf
6. Thieves Flood Victims Phone With Calls to Loot Bank Accounts, http://www.wired.com/threatlevel/2010/05/telephony-dos/
7. Voipdiscount, http://www.voipdiscount.com
8. Voipuser.org, http://www.voipuser.org
9. Aschenbruck, N., Frank, M., Martini, P., Tolle, J., Legat, R., Richmann, H.: Present and Future Challenges Concerning DoS-attacks against PSAPs in VoIP Networks. In: Proceedings of International Workshop on Information Assurance (2006)
10. Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., Zhao, L.: Statistical analysis of a telephone call center. Journal of the American Statistical Association 100(469), 36–50 (2005)
11. Dritsas, S., Soupionis, Y., Theoharidou, M., Mallios, Y., Gritzalis, D.: SPIT Identification Criteria Implementation: Effectiveness and Lessons Learned. In: Proceedings of The IFIP International Information Security Conference, Springer, Heidelberg (2008)
12. Enck, W., Traynor, P., McDaniel, P., Porta, T.L.: Exploiting Open Functionality in SMS Capable Cellular Networks. In: Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS 2005), Alexandria, Virginia, USA (2005)
13. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking (1993)
14. Fuchs, C., Aschenbruck, N., Leder, F., Martini, P.: Detecting VoIP based DoS attacks at the public safety answering point. In: ASIACCS (2008)
15. Keromytis, A.D.: A Look at VoIP Vulnerabilities. USENIX; login: Magazine 35(1) (February 2010)
16. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The click modular router. ACM Transactions on Computer Systems (2000)
17. Mathieu, B., Gourhant, Y., Loudier, Q.: SPIT mitigation by a network level Anti-SPIT entity. In: Proc. of the 3rd Annual VoIP Security Workshop (2006)
18. Roesch, M.: Snort - lightweight intrusion detection for networks. In: LISA 1999: Proceedings of the 13th USENIX Conference on System Administration. USENIX Association (1999)
19. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), Updated by RFCs 3265, 3853, 4320, 4916 (2002)
20. Traynor, P., Enck, W., McDaniel, P., Porta, T.L.: Mitigating Attacks on Open Functionality in SMS-Capable Cellular Networks. In: 12th Annual International Conference on Mobile Computing and Networking (2006)

21. Traynor, P., Mcdaniel, P., Porta, T.L.: On attack causality in internet-connected cellular networks. In: USENIX Security Symposium (2007)
22. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using Hard AI Problems for Security. LNCS. Springer, Heidelberg (2003)
23. Wang, X., Chen, S., Jajodia, S.: Tracking anonymous peer-to-peer VoIP calls on the internet. In: CCS 2005: Proceedings of the 12th ACM conference on Computer and Communications Security (2005)
24. Wright, C.V., Ballard, L., Monrose, F., Masson, G.M.: Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In: SS 2007: Proceedings of the 16th USENIX Security Symposium. USENIX Association, Berkeley (2007)
25. Zhang, R., Wang, X., Yang, X., Jiang, X.: Billing attacks on SIP-based VoIP systems. In: WOOT 2007: Proceedings of the First USENIX Workshop On Offensive Technologies

Low-Cost Client Puzzles Based on Modular Exponentiation

Ghassan O. Karame and Srdjan Čapkun

ETH Zurich, Switzerland
karameg@inf.ethz.ch, capkuns@inf.ethz.ch

Abstract. Client puzzles have been proposed as a useful mechanism for mitigating Denial of Service attacks on network protocols. While several puzzles have been proposed in recent years, most existing non-parallelizable puzzles are based on modular exponentiations. The main drawback of these puzzles is in the high cost that they incur on the puzzle generator (the verifier). In this paper, we propose cryptographic puzzles based on modular exponentiation that reduce this overhead. Our constructions are based on a reasonable intractability assumption in RSA: essentially the difficulty of computing a small private exponent when the public key is larger by several orders of magnitude than the semi-prime modulus. We also discuss puzzle constructions based on CRT-RSA [11]. Given a semi-prime modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. We further show how our puzzle can be integrated in a number of protocols, including those used for the remote verification of computing performance of devices and for the protection against Denial of Service attacks. We validate the performance of our puzzle on PlanetLab nodes.

Keywords: Client Puzzles, Outsourcing of Modular Exponentiation, DoS Attacks, Secure Verification of Computing Performance.

1 Introduction

Client Puzzles are tightly coupled with Proof of Work systems in which a client (prover) needs to demonstrate to a puzzle generator (verifier) that it has expended a certain level of computational effort in a specified interval of time. Client puzzles found their application in a number of domains, but their main applications concerned their use in the protection against Denial of Service (DoS) attacks [43,45,49] and in the verification of computing performance [13,44].

To be useful in practice, client puzzles have to satisfy several criteria: namely, they need to be inexpensive to construct and verify, and in many applications should be non-parallelizable. Non-parallelizability of puzzles is an especially important property since clients can involve other processors at their disposal e.g., to inflate their problem-solving performance claim.

A number of puzzles have been proposed [45], but these proposals are either efficient and parallelizable [24,49] or non-parallelizable and inefficient (typically

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 679–697, 2010.

in result verification) [13,43,44]. Non-parallelizable puzzles are mainly based on modular exponentiation (e.g., [43]); in these puzzles, the verifier has to perform $O(\log(N))$ modular multiplications to construct a puzzle instance and verify its solution. This high cost hindered the large-scale deployment of puzzles based on modular exponentiation in today's online applications [45].

In this paper, we propose puzzles based on modular exponentiation that reduce the cost incurred on the puzzle generator in existing modular exponentiation puzzles. Our constructions are based on a reasonable intractability assumption in RSA: informally, this assumption states that it is computationally intractable to compute a small private exponent d when the public exponent e is larger by several orders of magnitude than the modulus N. It is well known that RSA is insecure when the private exponent is small and the public key $e < N^{1.875}$ [10,50]. However, when $e \geq N^2$, RSA is considered to be secure [10,11,50]. Defeating this assumption would essentially imply a further restriction in the RSA problem, that has not been reported to date. Note that when e is large, the cost of encryption and/or signature verification in RSA is prohibitively high, which explains why this class of RSA keys is not widely used. To the best of our knowledge, this is the first work that leverages on this class of RSA keys to construct low-cost modular exponentiation puzzles. Where appropriate, we also discuss puzzle constructions based on CRT-RSA [11].

Based on this intractability assumption, we show that the costs incurred on the generator of modular exponentiation puzzles can be considerably reduced for any exponent of choice (i.e., for any puzzle difficulty). More specifically, we provide constructions for (variable-base) fixed-exponent and variable-exponent modular exponentiation puzzles and we show that the verifier only needs to perform a modest number of modular multiplications to construct and verify these puzzles. Given a modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. For example, for a 1024-bit modulus N, $k = 80$, the verifier's cost is reduced by a factor of 12.

As a by-product, our puzzle can be used to efficiently verify the integrity of outsourced modular exponentiations (modulo a semi-prime). We further show how our puzzle can be integrated in protocols used for remote verification of computing performance and for DoS protection. We validate the performance of our puzzle through experiments on a large number of PlanetLab nodes [1].

The rest of the paper is organized as follows. In Section 2, we define client-puzzles and we introduce our assumptions based on RSA. In Section 3, we introduce our puzzles and we provide a security proof for their constructions. Section 4 outlines some applications that can benefit from our proposed scheme. In Section 5, we overview the related work and we conclude the paper in Section 6.

2 Preliminaries

2.1 Client Puzzle Properties

Here, we state the security notions of client puzzles (adapted from [14]).

Definition 1. *A client puzzle Puz is given by the following algorithms:*

- Setup *is a probabilistic polynomial time setup algorithm that is run by the puzzle generator. Given a security parameter k, it selects the key space $\mathcal{S}$, the hardness space $\mathcal{T}$, the string space $\mathcal{X}$, the puzzle instance space $\mathcal{I}$ and puzzle solution space $\mathcal{P}$. It then selects the puzzle parameters $params \leftarrow (\mathcal{S}, \mathcal{T}, \mathcal{X}, \mathcal{I}, \mathcal{P})$. The secret $s \in \mathcal{S}$ is kept private by the puzzle generator.*
- GenPuz *is a probabilistic polynomial time puzzle generation algorithm that is run by the puzzle generator. On input $s \in \mathcal{S}$, $Q \in \mathcal{T}$ and $a \in \mathcal{X}$, it outputs a puzzle instance $puz \in \mathcal{I}$.*
- FindSoln *is a probabilistic solution finding algorithm. On inputs $puz \in \mathcal{I}$ and a run time $\tau \in \mathbb{N}$, it outputs a potential solution $soln \in \mathcal{P}$ after at most τ clock cycles of execution.*
- VerAuth *is a puzzle authenticity verification algorithm. On inputs $s \in \mathcal{S}$ and $puz \in \mathcal{I}$, it outputs* true *or* false.
- VerSoln *is a deterministic solution verification algorithm. On inputs $s \in \mathcal{S}$, $puz \in \mathcal{I}$ and a solution $soln \in \mathcal{P}$, it outputs* true *or* false.

It is required that if $params \leftarrow \text{Setup}(k)$ and $puz \leftarrow \text{GenPuz}(s, Q, a)$ where $s \in \mathcal{S}$, $Q \in \mathcal{T}$ and $a \in \mathcal{X}$, then (1) $\text{VerAuth}(s, puz) = \text{true}$, (2) $\exists \tau \in \mathbb{N}$ *such that $soln \leftarrow \text{FindSoln}(puz, \tau)$ and $\text{VerSoln}(s, puz, soln) = \text{true}$.*

Definition 2. ***(Puzzle-unforgeability.)*** *A client puzzle Puz is* UF *(unforgeable) if the probability that any probabilistic polynomial-time adversary $\mathcal{M}$ succeeds in producing $\bar{Puz}$, such that $\bar{Puz}$ was not previously created by the puzzle generator and $VerAuth(\bar{Puz}) = \text{true}$, is a negligible function of k.*

Definition 3. ***(Puzzle-difficulty.)*** *Let $\epsilon_{k,Q}(.)$ be a monotonically increasing function, where k is a security parameter and Q is a hardness parameter. A client-puzzle Puz is* $\text{DIFF}_{\text{k,Q}}$ *if for all $\tau \in \mathbb{N}$, for all security parameters $k \in \mathbb{N}$, for all $Q \in \mathbb{N}$, the success of any adversary $\mathcal{M}$, that is restricted to τ clock cycles of execution, is bounded by $\epsilon_{k,Q}(\tau)$ in solving Puz.*

2.2 Rivest's Repeated-Squaring Puzzle

In [43], Rivest *et al.* proposed a non-parallelizable time-lock puzzle based on repeated-squaring to enable time-release cryptography.

In this puzzle, the puzzle generator encrypts a message M into a ciphertext C as follows: $C = M + X^{a^t} \mod N$ given an integer X, an exponent a, a large integer t and an appropriate semi-prime modulus N. This computation can be performed efficiently using the trapdoor offered by Euler's function: $X^{a^t} \mod N \equiv X^{a^t \mod \phi(N)} \mod N$. On the other hand, to acquire M from C, the client needs to compute $X^{a^t} \mod N$ in $\log(a^t) \approx t$ modular multiplications.

When used as a client-puzzle (e.g., [44]), this puzzle is used such that the prover is required to compute $X^{a^t} \mod N$ given X, a, t and N. This computation is then verified by the puzzle generator through the trapdoor offered by Euler's function in $O(\log(N))$ modular multiplications.

2.3 RSA with a Small Private Exponent

The RSA cryptosystem [42] is the most widely used public-key cryptosystem. Let $N = pq$ be the product of two large and distinct primes and let e and d be inverses modulo $\phi(N) = (p-1)(q-1)$. Throughout the rest of the paper, we assume that p and q are balanced primes; that is, $|p| = |q|$. For $k \in \mathbb{N}^+$ ($\mathbb{N}^+ = \mathbb{N} - \{0\}$), the public RSA key e and the private RSA key d satisfy: $e \cdot d - 1 = k \cdot \phi(N)$.

It is known that RSA is insecure when $e \leq N^{1.875}$ and d is small [8,10,15–17,20,23,25,28,33,38,50]. Existing attacks on this class of "weak" RSA keys are mostly based on Wiener's attack [50] and/or on Boneh and Durfee's attack [10]. Wiener's continued fraction attack can be used to efficiently factor N when $e \leq N$ and $d < N^{\frac{1}{4}-\epsilon}$ and Boneh and Durfee's lattice-based attack [10] shows that private exponents up to $N^{0.2929}$ are unsafe when $e < N^{1.875}$. Blömer *et al.* [8] further generalized Wiener's attack to factor N in polynomial time for every $e \leq N$ satisfying $ex + y \equiv 0 \mod \phi(N)$, where x and y are short. Gao[1] and Howgrave-Graham and Seifert [30] extended these attacks to factor N given several common modulus instances of RSA with $d < N^{0.4}$ and $e \leq N$.

2.4 Low-Cost Decryption in RSA

In this work, we consider RSA keys that do not belong to the weak class of RSA keys, yet enable low-cost decryption in RSA. More specifically, we consider the following class of RSA keys:

Class $\mathcal{A}$: *Class $\mathcal{A}$ is defined as the set of all RSA keys (N, e, d) where: $N = pq$, p and q are two large balanced primes, $e \geq N^2$ such that* $\gcd(e, \phi(N)) = 1$ *and d is small such that* $ed - 1 \equiv 0 \mod \phi(N)$.

When $(N, e, d) \in \mathcal{A}$, the fastest known algorithm that computes d from (N, e) runs exponentially in time with $|d|$. This hardness assumption on class $\mathcal{A}$ is based on the observations of Wiener [50] and Boneh *et al.* [10]. When $e \geq N^2$, all known attacks against small private RSA exponent are defeated. More specifically, the continued fraction algorithm [50], the lattice-based attack [10] and Coppersmith's attack [15,16] fail even when d is small (for the reason why, refer to Appendix A). For example, when $e \geq N^2$, $|d| \geq$ 80-bits, no known feasible algorithm can compute d from $(N, e) \in \mathcal{A}$, and therefore factor N. RSA keys that belong to $\mathcal{A}$ clearly do not optimize the cost of RSA encryption and signature schemes; when e is large, the cost of encryption and/or signature verification is prohibitively high, which explains why this class is not widely used in RSA.

Remark 1. Given the work of Blömer *et al.* [8], we can safely extend class $\mathcal{A}$ to the set of RSA keys that satisfy a generalized RSA key equation of the form $ex + y \equiv 0 \mod \phi(N)$, where $e \geq N^2$ and x, y are small (for the reason why, see Appendix B). Note that a special instance of this equation is the standard RSA equation, where $x = d$ and $y = -1$.

[1] Gao's unpublished attack is described by Howgrave-Graham and Seifert in [30].

Remark 2. One simple way to generate large public keys whose modular inverses are small is to pick d such that $|d|$ is small, and compute $e' = d^{-1} \mod \phi(N)$. Then, a large public key e is computed from e' as follows: $e = t\phi(N) + e'$, where $t \in \mathbb{N}^+$ and $t \approx N^2$. The verifier then deletes e' and publishes (N, e) [9].

Where appropriate, we also consider in this work the following class of RSA keys:

Class $\mathcal{B}$: *Class $\mathcal{B}$ is defined as the set of all RSA keys (N, e, d) where: $N = pq$, p and q are balanced large primes, $e \in \mathbb{N}^+$, $\gcd(e, \phi(N)) = 1$, $d_p \equiv d \mod p$ and $d_q \equiv d \mod q$ such that $d_p \neq d_q$, $d > N^{0.5}$ and $ed - 1 \equiv 0 \mod \phi(N)$.*

When $(N, e, d) \in \mathcal{B}$, the fastest known algorithm that computes d from (N, e) runs in $\min(\sqrt{d_p}, \sqrt{d_q})$. The use of RSA keys in class $\mathcal{B}$ is suggested by Wiener [50] and Boneh [10] to speed up RSA decryption[2]. Since decryptions are often generated modulo p and q separately and then combined using the Chinese Remainder Theorem (CRT) [11], Wiener proposes the use of a private key d such that both $d_q \equiv d \mod q$ and $d_p \equiv d \mod p$ are small ($d_p \neq d_q$). The best known attack against this scheme runs in $\min(\sqrt{d_p}, \sqrt{d_q})$ [10, 26].[3] When $|\min(\sqrt{d_q}, \sqrt{d_p})| \geq 80$ bits, $|N| = 1024$-bits, there exists no feasible algorithm that can compute d from $(N, e) \in \mathcal{B}$.

Remark 3. Throughout this paper, we consider RSA keys in the class $\mathcal{A} \cup \mathcal{B}$ as a building block to construct low-cost puzzles based on modular exponentiation. To simplify the description and analysis of our puzzles, we consider RSA keys in $\mathcal{A} \cup \mathcal{B}$ where the public exponent $e \geq N^2$. We point out, however, that our analysis also applies for all RSA keys in $\mathcal{A} \cup \mathcal{B}$.

3 Low-Cost Puzzles Based on Modular Exponentiation

3.1 System and Attacker Model

We consider the following model. A verifier (puzzle generator) wants to verify that a prover performed a certain number of modular exponentiations (modulo a semi-prime) in a specified interval of time. For that purpose, the verifier requires that the prover runs a software on its machine (i.e., a modular exponentiation puzzle) for a specific amount of time. In some application scenarios, we will need to assume that the verifier and the prover can exchange authenticated messages over the communication channel. We assume, however, that the verifier does not have access to the prover's machine and thus cannot check the prover's environment; this includes the number of processors at the disposal of the prover, the connections established from the prover's machine, etc..

[2] This RSA variant is widely used in smart cards.

[3] Recently, Jochemsz *et al.* propose in [31] a polynomial attack on small private CRT-RSA exponents. This attack only works when $\min(d_p, d_q) \leq N^{0.073}$. However, in this case, brute-force search attacks would also be feasible on CRT-RSA.

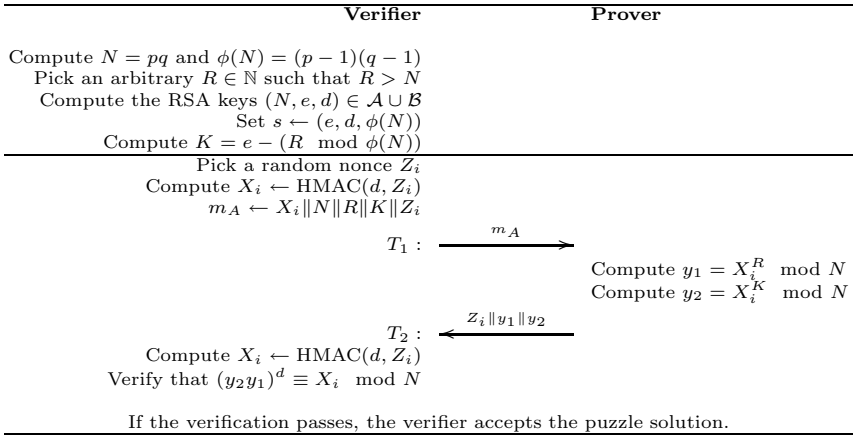


Fig. 1. Fixed-Exponent Puzzle based on Modular Exponentiation

An untrusted prover constitutes the core of our attacker model. We assume that a prover possesses considerable technical skills by which it can efficiently analyze, decompile and/or modify executable code as necessary. More specifically, an untrusted prover has knowledge of the algorithm used for the computation and of the algorithm that is run by the verifier. We assume that untrusted provers are motivated to inflate their puzzle solving performance (i.e., untrusted provers have incentives to solve the puzzle in a faster time than what they can genuinely perform). However, we assume that provers are computationally bounded.

3.2 Low-Cost Fixed-Exponent Modular Exponentiation Puzzle

Here, we present our puzzle based on (variable-base) fixed-exponent modular exponentiation. In Section 3.3, we propose a variant puzzle based on variable-exponent modular exponentiation. Our puzzle is shown in Figure 1.

In the setup phase of our puzzle, the verifier picks two large balanced primes p and q (of sufficient size to prevent factoring of $N = pq$, e.g., $|p| = |q| \geq 512$-bits), computes $N = pq$ and $\phi(N) = (p-1)(q-1)$. Given N, the verifier also generates RSA keys (N, e, d) such that $(N, e, d) \in \mathcal{A}$, $|d| = k$, where k is a security parameter or $(N, e, d) \in \mathcal{B}$, where $|\min(\sqrt{d_p}, \sqrt{d_q})| = k$. The verifier also picks a puzzle difficulty $R \in \mathbb{N}$ and computes $K = e - (R \mod \phi(N))$. We show later that K will enable low-cost verification of the puzzle solution. (N, R, K) are public parameters that set the puzzle hardness and $s \leftarrow (e, d, \phi(N))$ is kept secret. Note that R needs to be larger than $\phi(N)$ to ensure the security of our scheme[4]. Typically, R is chosen such that $R \gg \phi(N)$ ($|R| \geq 100,000$ bits) to achieve a moderate runtime of the puzzle (in the order of tens of milliseconds, see Section 3.4). However, even in the case where the verifier would like to e.g., simply outsource the computation of an arbitrary $R' \leq \phi(N)$, this can be remedied by setting $R \leftarrow R' + t\phi(N)$, where $t \in \mathbb{N}^+$.

[4] This can be achieved by setting $R > N$.

- *Puzzle Generation:* In round i, the verifier generates $puz \leftarrow (X_i, Z_i, R, K, N)$,[5] where $X_i \leftarrow \text{HMAC}(d, Z_i)$. Here, Z_i is a nonce and $|X_i| \geq k$. In the sequel, we assume that $\text{HMAC}(X, Y)$ is a keyed collision-resistant pseudo-random function, where X is used as an input key.
- *Puzzle Solution:* Given puz, the prover computes $soln \leftarrow (y_1 = X_i^R \mod N, y_2 = X_i^K \mod N, Z_i)$.
- *Solution Verification:* Given $soln$, the verifier checks if $(y_2 y_1)^d \equiv X_i^{d(R+K)} \mod N \equiv X_i^{ed} \mod N \equiv X_i \mod N$.

Remark 4. Note that our puzzle is stateless; only a single value of the secret $s \leftarrow (e, d, \phi(N))$ is stored by the verifier regardless of the number of puzzles (instances) that the verifier generates. All the required data to solve a given puzzle is contained in puz, whereas the knowledge of s and $soln$ are sufficient to verify the puzzle solution $soln$. The uniqueness of each puzzle instance can be ensured by having GenPuz select Z_i a counter and increment Z_i in each puzzle instance.

Remark 5. When $R = 0$, the prover simply computes $y_2 = X_i^e \mod N$, and the verifier verifies the puzzle solution by computing y_2^d. Such a puzzle is then based on "standard" RSA. The major limitation of this "standard" RSA-based puzzle is that the choice of the puzzle difficulty (i.e., the exponent) is dependent on the choice of d and $\phi(N)$. This particularly hinders the construction of repeated-squaring puzzles (e.g., [43]) or the secure outsourcing of modular exponentiations for a given exponent.

Puzzle Construction and Verification Costs: In our puzzle, the verifier only needs to perform 1 HMAC operation (2 hashes) to construct the puzzle and a small number of modular multiplications (computing $(y_2 y_1)^d$) to verify the puzzle solution:

- $(N, e, d) \in \mathcal{A}$: In this case, the puzzle verification is performed in $O(\log d)$ modular multiplications. When $|d| = k$, the verifier's cost is reduced by a factor of $\frac{\log N}{\log d} = \frac{|N|}{k}$, when compared to the original repeated-squaring puzzle [43]. When $|N| = 1024, k = 80$, the puzzle verification cost could be as low as $\frac{3}{2}80 = 120$ modular multiplications[6] and the average improvement gain in the puzzle solution verification is almost 12 (i.e., $\frac{1.5 \times 1024}{1.5 \times 80}$). Similarly, when $|N| = 2048$, $k = 112$, the average improvement gain increases to 18.
- $(N, e, d) \in \mathcal{B}$: In this case, the puzzle verification is performed in $O(\log(d_p) + \log(d_q))$ modular multiplications using the CRT. When $|\min(\sqrt{d_p}, \sqrt{d_q})| = k$, the verifier's cost is reduced by a factor of $\frac{\log N}{2 \log d} = \frac{|N|}{4k}$, when compared to the original repeated-squaring puzzle [43].

[5] When R is very large, the verifier can reduce the communication costs by sending $r \ll R$, such that $R = F(r)$, where $F(r)$ is an expansion function of r.

[6] On average, the computation of $X^d \mod N$ requires $1.5 \log d$ modular multiplications [35].

Prime number generation (i.e., computing N) and the pre-computation of e and d are generally expensive operations for the verifier; however, this computation is performed only once at the setup phase[7] and (N, e, d) are subsequently used for all the puzzles generated by the verifier.

Security Analysis: To analyze the security of our scheme, we first show that it is computationally infeasible for an adversary to acquire the secret s held by the verifier in our puzzle. Based on this, we show that an adversary needs to perform at least $O(\log R)$ modular multiplications to compute the solution $soln$ to a puzzle instance puz such that $\text{VerSoln}(s, puz, soln) = \text{true}$.

We use the following game $\text{Exec}_{\mathcal{M}}(k)$ between a challenger and a probabilistic polynomial time (p.p.t.) adversary $\mathcal{M}$:

- The challenger runs Setup on input k to obtain $N = pq$ chosen uniformly at random from $\mathcal{N}$, d chosen uniformly at random from $\{2^k..2^{k+1}\}$ and computes e such that $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$. The challenger, then stores the secret $s \leftarrow (e, d, \phi(N))$. The challenger further picks $R > N$ chosen uniformly at random from $\mathcal{R}$ and computes K as shown in Figure 1.
- The adversary $\mathcal{M}$ gets to make as many $\text{CreatePuz}(Z_i)$ queries as it likes. In response, the challenger *(1)* creates $puz \leftarrow (X_i, Z_i, R, K, N)$ as shown in Figure 1, *(2)* computes $soln$ such that $\text{VerSoln}(s, puz, soln) = \text{true}$ and *(3)* outputs $(puz, soln)$.

Adversary $\mathcal{M}$ terminates the game by outputting an integer C. We say that $\mathcal{M}$ wins $\text{Exec}_{\mathcal{M}}(k)$ if $C \equiv 0 \mod \phi(N)$ (i.e., if $\mathcal{M}$ computes a multiple of $\phi(N)$). In this case, we set the output of $\text{Exec}_{\mathcal{M}}(k)$ to be 1 and otherwise to 0. We then define the success of $\mathcal{M}$ as $\text{Succ}_{\mathcal{M}}(k) = Pr[\text{Exec}_{\mathcal{M}}(k) = 1]$.

Theorem 1. *Computing a multiple of $\phi(N)$ and, in particular, computing d given (N, e) is computationally as hard as factoring (see [40] for the proof).*

Lemma 1. *$(N, R + K, d) \in \mathcal{A} \cup \mathcal{B}$ if $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$.*

Proof. Let $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$ satisfy the RSA key equation: $ed - 1 \equiv 0 \mod \phi(N)$. Recall that e is kept secret by the challenger. Since $K = e - (R \mod \phi(N))$, then $\exists t_1 \in \mathbb{N}^+$ (since $R > N$) such that $R + K = e + t_1\phi(N)$. This means that $d(R + K) \equiv de \equiv 1 \mod \phi(N)$.

Given Theorem 1, computing e from $(R + K)$ is computationally as hard as factoring[8]. Since d is the modular inverse of e, d is equally the modular inverse of $(R + K)$. More specifically, it is easy to see that since $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$, then $(N, (R + K), d)$ are RSA keys in $\mathcal{A} \cup \mathcal{B}$ (since $(R + K) > e$).

Lemma 2. *For any p.p.t. adversary $\mathcal{M}$, $\text{Succ}_{\mathcal{M}}(k)$ is a negligible function of k.*

[7] Note that the computational load incurred by prime number generation equally applies to all protocols that make use of modular exponentiation or repeated-squaring (e.g., [43,49]).

[8] $(R + K - e)$ is a multiple of $\phi(N)$.

Proof. We show that if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in the $\text{Exec}_{\mathcal{M}}(k)$ game, then we can construct a polynomial-time algorithm that uses $\mathcal{M}$ as a subroutine to solve the RSA problem in $\mathcal{A} \cup \mathcal{B}$, i.e., to compute $\bar{d}$ given a public RSA key $(\bar{N}, \bar{e})$ where $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$.

Let $\mathcal{M}$ be a p.p.t. adversary that outputs a multiple of $\phi(N)$ in the game $\text{Exec}_{\mathcal{M}}(k)$ with probability $\text{Succ}_{\mathcal{M}}(k)$. In the $\text{Exec}_{\mathcal{M}}(k)$ game, let $R + K = e'$. Recall that in $\text{Exec}_{\mathcal{M}}(k)$, $K = e - R \mod \phi(N)$, where e is chosen uniformly at random from $\mathcal{A} \cup \mathcal{B}$ and $R > N$. Given this, note that $e' > e + \phi(N)$ and $K \geq N^2 - \phi(N)$; this suggests that $R + K = e' > R + N^2 - \phi(N)$ and therefore $R < e' - N^2$.

Let $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$, where $N \in \mathcal{N}$, $d \in [2^k, .., 2^{k+1}[$, $\bar{e} \geq N^2 + \phi(N)$. Then, we construct a polynomial-time algorithm $\mathcal{E}$ that interacts with $\mathcal{M}$ as follows:

- Given the public key $(\bar{N}, \bar{e})$, $\mathcal{E}$ picks $\bar{R}$ at random from $\{N + 1, .., \bar{e} - N^2\}$.
- $\mathcal{E}$ computes $\bar{K} = \bar{e} - \bar{R}$ and constructs a transcript $\bar{T}$ that is composed of a number of tuples of the form $(\bar{X}_i, \bar{Z}_i, \bar{R}, \bar{K}, \bar{N}, \bar{X}_i^{\bar{R}} \mod \bar{N}, \bar{X}_i^{\bar{K}} \mod \bar{N})$, $i \in \mathbb{N}$, where $\bar{X}_i$ is a pseudorandom string that has a similar distribution as HMAC(.) and $\bar{Z}_i$ is a counter.

Note that since $\bar{R} > \bar{N}$, $\exists t_1 \in \mathbb{N}^+$ such that $\bar{R} - t_1\phi(\bar{N}) = (\bar{R} \mod \phi(\bar{N}))$. Let $\bar{e_1} = \bar{e} - t_1\phi(\bar{N})$. It is easy to see in this case that $\bar{K} = \bar{e_1} - (\bar{R} \mod \phi(\bar{N}))$. Furthermore, since $\bar{e_1} \equiv \bar{e} \mod \phi(\bar{N})$, then d is a modular inverse of $\bar{e_1}$. We point out that since $\bar{e_1} = \bar{e} - t_1\phi(\bar{N}) = \bar{e} - \bar{R} + (\bar{R} \mod \phi(\bar{N}))$, then $\bar{e_1} > \bar{e} - \bar{R} \geq N^2$, since by construction $\bar{R} \leq \bar{e} - N^2$. Therefore, $(\bar{N}, \bar{e_1}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$.

Given this, it is easy to see that the view of $\mathcal{M}$ when run as a subroutine by $\mathcal{E}$ is distributed identically to the view of $\mathcal{M}$ in the game $\text{Exec}_{\mathcal{M}}(k)$. Recall that in $\text{Exec}_{\mathcal{M}}(k)$, X_i is a pseudorandom string, $K = e - R \mod \phi(N)$, where e is a secret such that $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$ and $(N, R + K, d) \in \mathcal{A} \cup \mathcal{B}$.

Therefore, if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in the $\text{Exec}_{\mathcal{M}}(k)$ game, then it can solve the above RSA problem. By the hardness assumption on $\mathcal{A}$ and $\mathcal{B}$, it is computationally infeasible for $\mathcal{M}$ to compute $\bar{d}$, or equivalently a multiple of $\phi(\bar{N})$ (Theorem 1), from $(\bar{N}, \bar{e})$ when $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$. Therefore, $\text{Succ}_{\mathcal{M}}(k)$ is negligible, thus concluding the proof.

Given this, we can show that our puzzle construction is both unforgeable (UF) and difficult ($\text{DIFF}_{\text{k,R}}$).

Corollary 1. *The puzzle construction of Figure 1 is* UF.

Proof Sketch: Given a puzzle instance $puz \leftarrow (X_i, Z_i, R, K, N)$, VerAuth($s$, puz) = true if and only if $X_i \leftarrow \text{HMAC}(d, Z_i)$.

Therefore, the only viable way for $\mathcal{M}$ to construct $p\bar{u}z \leftarrow (\bar{X}_i, \bar{Z}_i, R, K, N)$ such that $\text{VerAuth}(p\bar{u}z) = \text{true}$ and $p\bar{u}z$, $\bar{X}_i$, $\bar{Z}_i$ were not previously created by the challenger is to construct $(\bar{X}_i, \bar{Z}_i)$ such that $\bar{X}_i \leftarrow \text{HMAC}(d, \bar{Z}_i)$. Since HMAC(.) is a pseudorandom collision-resistant function, $\mathcal{M}$ cannot construct $(\bar{X}_i, \bar{Z}_i)$ without the knowledge of d. Following from Lemma 2, the success probability for $\mathcal{M}$ in acquiring d from our puzzle – and therefore constructing $p\bar{u}z$ such that $\text{VerAuth}(p\bar{u}z) = \text{true}$ – is bounded by $O(2^{-k})$.

Corollary 2. *The puzzle construction of Figure 1 is* $\text{DIFF}_{\text{k,R}}$.

Proof Sketch: Following from Lemma 2, it is computationally infeasible for $\mathcal{M}$ to compute a multiple of $\phi(N)$ given our puzzle. Furthermore, $\mathcal{M}$ cannot precompute the solution of the puzzle since it cannot predict X_i ($|X_i| \geq k$) nor the outcome of $y_2 y_1$ (since e is kept secret by the verifier).

The fastest known way for $\mathcal{M}$ to solve our puzzle is to compute y_1 and y_2 correctly. Modular Multiplication is an inherently sequential process [43]. The running time of the fastest known algorithm for modular exponentiation is linear in the size of the exponent. Although $\mathcal{M}$ might try to parallelize the computation of y_1 and/or y_2, the parallelization advantage is expected to be negligible[9] [43,44].

Note that $\mathcal{M}$ might try to perform the computation of y_1 and y_2, in parallel, using different machines at its disposal. In typical cases, $R \gg K$; this means that the computation of y_1 and y_2 requires at least $O(\log R)$ sequential modular multiplications. We point out that the verifier can prevent the separate computation of y_1 and y_2, by sending K to the prover once it receives y_1 (see Figure 2).

$\mathcal{M}$ can equally try to compute y_1 and/or y_2 through intermediate results that it previously computed (or intercepted) (e.g., when the base X_i is the result of a multiplication of two previously used numbers). This also applies to the original time-lock puzzle proposed in [43]; this can be remedied, with high probability, by setting $|\text{HMAC}(.)| \gg |Z_i|$.

Given this, the success of $\mathcal{M}$ – restricted to τ clock cycles of execution – in solving our puzzle is bounded by $\epsilon_{k,R}(\tau) = \min(\lfloor \frac{\tau}{\log R} \rfloor + O(2^{-k}), 1)$; $\mathcal{M}$ needs to perform at least $\tau = \log(R)$ clock cycles of execution to solve our puzzle.

3.3 Low-Cost Variable-Exponent Modular Exponentiation Puzzle

In some settings, the verifier might need to change the puzzle difficulty (i.e., the exponent) "on the fly" (e.g., when subject to DoS attacks). We briefly discuss how this can be achieved based on the proposed fixed-exponent puzzle.

Our variable-exponent puzzle and the related protocol are depicted in Figure 2. Similar to the fixed-exponent puzzle (Figure 1), in round i, the verifier creates the RSA keys $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$, picks $Z_i \in \mathbb{N}$ and computes $X_i \leftarrow \text{HMAC}(d, Z_i)$. Here, in addition, the verifier computes $v_i \leftarrow \text{HMAC}(d, X_i)$ such that $|v_i| = k$ and $\gcd(v_i, d) = 1$.[10]

[9] $\mathcal{M}$ might try to parallelize the multiplication of large numbers by splitting the multiplicands into smaller "words" and involving other processors in the multiplication of these words. Further details about this process can be found in [36]. However, this attack incurs a significant communication overhead that prevents an $\mathcal{M}$ from gaining any substantial speedup; given a large number of squaring rounds, the RTT between the cooperating processors needs to be in the order of few nanoseconds to achieve even a modest speedup.

[10] The probability that any number is coprime with d is $\frac{6}{\pi^2} \approx 0.61$. Therefore, only two choices are sufficient, on average, to create such a v_i (i.e., if $\gcd(\text{HMAC}(d, X_i)), d) \neq 1$, then with high probability $\gcd(\text{HMAC}(d, X_{i+1})), d) = 1$).

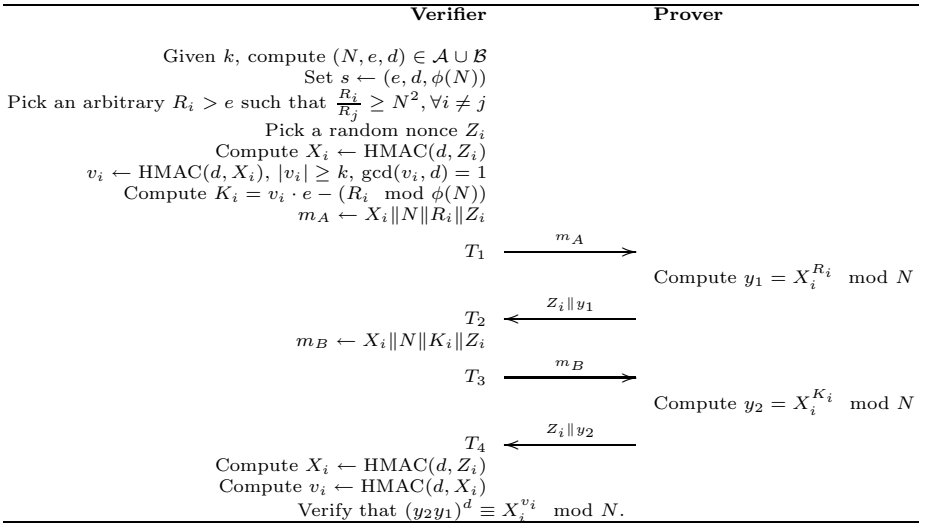


Fig. 2. Variable-Exponent Puzzle based on Modular Exponentiation. Note that y_1 and y_2 could be also transmitted in the same message. The separate transmission of y_1 and y_2, however, prevents the computation of y_1 and y_2 in parallel and enables the use of this puzzle to remotely verify the computing performance of devices (see Section 4.2).

The puzzle instance at round i is then comprised of the tuple $puz \leftarrow (X_i, Z_i, N, R_i, K_i)$, where $K_i = v_i e - (R_i \mod \phi(N))$, and $R_i \in \mathbb{N}$. Its solution is $soln \leftarrow (Z_i, X_i^{R_i} \mod N, X_i^{K_i} \mod N)$. To verify $soln$, the verifier checks if $(y_2 y_1)^d \equiv X_i^{v_i} \mod N$.

It is easy to see that the cost incurred on the verifier in this puzzle exceeds that of the fixed-exponent puzzle by $|v_i| = k = 80$ modular multiplications (mainly in puzzle solution verification). For instance, when $(N, e, d) \in \mathcal{A}$, $soln$ can be verified in 240 modular multiplication; the verification gain when compared to existing modular exponentiation puzzles is then $\frac{1536}{240} \approx 7$, given a 1024-bit N.

Corollary 3. *The puzzle construction of Figure 2 is* UF *and* $\text{DIFF}_{\text{k,R}_\text{i}}$ *when* (1) $(N, e, d) \in \mathcal{A}$ *and* $R_i > e$ *such that* $\frac{R_i}{R_j} \geq N^2, \forall i \neq j$, *or* (2) $(N, e, d) \in \mathcal{B}$.

Proof Sketch: Due to lack of space, we only provide the main intuition behind the proof.

Consider a variant of the aforementioned $\text{Exec}_{\mathcal{M}}(k)$ game where the transcript of interaction T between the adversary $\mathcal{M}$ and the challenger is composed of a number of tuples $(X_i, Z_i, R_i, K_i, N, X_i^{R_i} \mod N, X_i^{K_i} \mod N)$, $i \in \mathbb{N}$.

Similar to the analysis in Lemma 2, we can show that if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in this variant $\text{Exec}_{\mathcal{M}}(k)$ game, then it can compute a multiple of $\phi(N)$ given several instances of the generic RSA key equation $e_i x_i + y_i \equiv 0 \mod \phi(N)$ with common modulus and unknown x_i, y_i, where $e_i = R_i + K_i$, $x_i = d$ and $y_i = -v_i$. Note that $v_i \neq v_j, \forall i \neq j$. This is especially important for

Table 1. Construction and Verification Costs of Puzzles. "Mod. Mul." denotes modular multiplication and "Mul." refers to multiplication. $\mathcal{B}$-Puzzle and $\mathcal{A}$-Puzzle refer to our proposed puzzle created using classes $\mathcal{B}$ and $\mathcal{A}$, respectively, of RSA keys. $^{(*)}$ Note that $d \ll N$, $v \ll N$; $|v| = |d| = k \geq 80$.

	Verifier Cost	**Prover Cost**
Repeated-Squaring [43]	1 modulus, 1 mul. $O(\log(N))$ mod. mul.	$O(\log R)$ mod. mul.
Fixed Exponent $\mathcal{A}$-Puzzle	1 modulus, 1 HMAC $O(\log(d))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Variable Exponent $\mathcal{A}$-Puzzle	1 modulus, 2 HMAC $O(\log(d) + \log(v))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Fixed Exponent $\mathcal{B}$-Puzzle	1 modulus, 1 HMAC $O(\log d^2)$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Variable Exponent $\mathcal{B}$-Puzzle	1 modulus, 2 HMAC $O(\log d^2 + \log(v))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.

the security of our puzzle. Otherwise, $\mathcal{M}$ can compute a multiple of $\phi(N)$ solely from R_i and R_j ($(R_i - R_j) \equiv 0 \mod \phi(N)$).

When $(N, e, d) \in \mathcal{A}$ and $R_i > e$ such that $\frac{R_i}{R_j} \geq N^2, \forall i \neq j$, then $\frac{e_i}{e_j} = \frac{R_i + K_i}{(R_j + K_j} > N, \forall i \neq j$. In this case, all existing attacks on common modulus instances of RSA are defeated (refer to Remark 1 and the related Appendix B); the best known algorithm that computes $\phi(N)$ from (N, e_i) runs exponentially in time in $|x_i y_i| = |dv_i|$ since d and v_i are in lowest terms by construction (i.e., $\gcd(d, v_i) = 1$). In our case, $|dv_i| \geq 2k = 160$. We conclude that it is computationally infeasible for $\mathcal{M}$ to compute a multiple of $\phi(N)$ from T. Similarly, when $(N, e, d) \in \mathcal{B}$, $d = |x_i| > N^{0.5}$ [10,50], there exists no polynomial-time algorithm that can factor N in this case [27].

Similar to Corollaries 1 and 2, it can be shown that the puzzle construction of Figure 2 is UF and DIFF_{k,R_i}.

3.4 Performance Evaluation

Table 1 summarizes the costs incurred in our puzzles when compared with those in the repeated-squaring puzzle of [43]. To the best of our knowledge, there are no other proposed non-parallelizable puzzles that are based on modular exponentiation.

To evaluate the performance of our puzzle, we implemented it in JAVA on four different workstations. We evaluate the performance of our scheme on various other processors in Section 4.2. In our implementation, we used built-in JAVA functions for prime number generation, repeated-squaring using addition chains, etc.. While a faster implementation of our scheme could be achieved using lower-level programming and/or specialized hardware or software, we aim to demonstrate in this work the feasibility of our proposal using available standard algorithms and programming tools. Our findings (Table 2) show that our schemes considerably reduce the cost incurred on the generator of modular exponentiation puzzles.

Table 2. Implementation results on four different workstations equipped with Intel(R) Core(TM)2 Duo CPU T7500 processor running at 2.20 GHz. Here, N is 1024-bit composite integer, $k = 80$. We conducted our measurements over the LAN (max RTT = 100 ms). Our results are averaged over 10 distinct measurements. The puzzle verification time is interpolated from the number of squarings per second on each machine.

# Squaring (Size of R in bits)	Puzzle Runtime	Verification Time $\mathcal{A}$-Puzzle	Verification Time $\mathcal{B}$-Puzzle	Verification Time of [43]
6500000	154.067 s	1.89 ms	7.56 ms	24.3 ms
6500000	172.174 s	2.11 ms	8.5 ms	27.12 ms
6500000	170.611 s	2.1 ms	8.4 ms	26.9 ms
6500000	165.034 s	2.03 ms	8.12 ms	26 ms

4 Applications

4.1 Efficient Resilience to DoS Attacks

A natural application of our puzzles lies in the area of protection against DoS attacks. In this context, an online server requires that its clients solve our puzzles before attending to their requests in order to prevent DoS attacks.

When used in DoS protection, it is important, however, that the server ensures that puzzle instances and solution pairs are used only once. To achieve this, the server should filter out resubmitted correctly solved puzzles and solution pairs [14,32]. In our case, the storage is minimized since the server can simply store the hash of the nonce Z_i that corresponds to the most recent solved puzzle (where Z_i is a counter). The verifier will then accept to verify only recent puzzles.

4.2 Remote Verification of Computing Performance

To cope with the advances in processing power, the computing community is relying on the use of benchmarks. While several benchmarks [2,13,19,44] were proposed as a mean to evaluate a processor's computing power, most of these benchmarks are parallelizable (see Section 5).

Based on our variable-exponent puzzle (Figure 2), we construct a secure benchmark that enables any machine (even with modest computation power, e.g., a PDA device) to remotely *upper-bound* the computing performance of single-processors. Our benchmark differs from the puzzle in Figure 2 as follows: upon reception of y_1, the verifier estimates the number of squarings per second: $S = \frac{R_i}{T_2 - T_1}$ of the prover's machine. This estimate is accepted by the verifier if, after receiving y_2 at time T_4: *(1)* $(T_4 - T_3) \leq \epsilon \cdot (T_2 - T_1)$, given a negligible ϵ and *(2)* $(y_2 y_1)^d \equiv X^{v_i} \mod N$.

Corollary 4. *Given the puzzle depicted in Figure 2, the success of a p.p.t. adversary $\mathcal{M}$ in inflating the number of squarings that it can perform per second by more than a small ϵ is negligible.*

Proof Sketch: Recall that our puzzle is UF and DIFF_{k,R_i}. Therefore, the only viable method for $\mathcal{M}$ to inflate its performance claim is to send $\bar{y_1}$, chosen at random, ahead of time, compute y_1 correctly and distribute the computation of

the corresponding $\bar{y_2}$ and y_2, such that $\bar{y_2}\bar{y_1} \equiv y_2 y_1 \mod N$, to other nodes at its disposal. This would enable $\mathcal{M}$ to decrease the measured time corresponding to the computation of $O(\log R_i)$ modular multiplications by $\Delta = (T_4 - T_3)$ time units (Δ includes the communication delay D of the path between the verifier and $\mathcal{M}$). However, this is countered by the fact that the verifier does not accept the prover's performance claim unless $(T_4 - T_3) \leq \epsilon \cdot (T_2 - T_1)$.

In this case, the maximum performance claim that $\mathcal{M}$ can make is $S_{max} = \frac{|R_i|}{(1-\epsilon)\cdot(T_2-T_1)}$. Note that ϵ is interpolated from the measured number of squarings per second S; if it takes $(T_2 - T_1)$ time units for $\mathcal{M}$ to perform $\log(R_i)$ modular multiplications, then the computation of y_2 can be upper-bounded by choosing $\epsilon = \frac{\log(e-N)}{\log(R_i)}$. For a 1024-bit modulus N ($|\phi(N)| \approx |N|$), $|e| \approx |N^2|$ and $|R_i| > 100,000$, then $\epsilon \simeq \frac{S_{max}}{S} \simeq 0.03$ squarings per second.

Our protocol finds applicability in a multitude of application domains. For example, our benchmark can be used in online distributed computing applications (e.g., [3]) or in the secure ranking of supercomputers (e.g., [4]) to prevent possible frauds in performance claims[11].

We evaluated our benchmark on various processors running on 12 different PlanetLab nodes [1] (refer to Section 3.4 for implementation details). Our findings (see Table 3) suggest that our proposed benchmark reflects well the performance of various processors.

Table 3. Implementation results on 12 different PlanetLab Nodes. S refers to the number of squarings per ms.

CPU Description	Idle CPU	S
Intel(R) Pentium(R) D 3.20GHz	6.40%	7.48
Intel(R) Pentium(R) D 3.00GHz	26.20%	15.24
Intel(R) Pentium(R) 4 3.20GHz	30.70%	15.81
Intel(R) Pentium(R) D 3.40GHz	14.10%	18.22
Intel(R) Xeon(R) 3060 2.40GHz	46.60%	28.01
Intel(R) Pentium(R) D 3.20GHz	20.00%	29.35
Intel(R) Xeon(R) 3075 2.66GHz	19.70%	29.72
Intel(R) Pentium(R) 4 3.06GHz	92.00%	31.72
Intel(R) Duo E6550 2.33GHz	63.80%	36.05
Intel(R) Duo T7500 2.20GHz	76.00%	38.11
Intel(R) Xeon(R) X3220 2.40GHz	73.30%	41.67
Intel(R) Xeon(R) E5420 2.50GHz	87.70%	50.97

5 Related Work

Client puzzles found their application in several domains (e.g., prevention against DoS attacks [21,48], protection from connection depletion attacks [32], protection against collusion [41]). Several computational puzzles have been proposed in the recent years [43,45,49]. However, most of these puzzles are parallelizable; a comprehensive survey of existing client puzzles can be found in [45]. In [43], Rivest *et al.* proposed a non-parallelizable puzzle based on repeated-squaring to enable time-release cryptography. The drawback of this scheme, if used for DoS protection, is that it incurs an expensive cost on the puzzle generator. Wang *et al.*

[11] For instance, a supercomputer, connected to a hidden processor cluster, can inflate its performance claims by involving these other processors in the construction of the benchmark's solution. The literature contains a significant number of similar "anecdotes" where both individuals and manufacturers have tendencies to exaggerate their computing performance (e.g., [5,6]).

propose in [47] a scheme that enables the server to adjust the puzzle difficulty in the presence of an adversary whose computing power is unknown. In [14], Chen *et al.* provide a formal model for the security of client-puzzles. In this work, we use their model as a building block for analyzing the security of our proposed puzzle. Several other contributions address the problem of secure outsourcing of computations to untrusted servers (e.g., [7,12]). Clarke *et al.* present protocols for speeding up exponentiation using untrusted servers in [46]. In [29], Hohenberger *et al.* describe a scheme to outsource cryptographic computations where the verifier can use two untrusted exponentiation programs to assist him in the computations. Memory-bound puzzles were proposed in [22,37] to overcome the limitations of existing computational puzzles. However, memory-bound puzzles cannot entirely substitute their computational counterpart e.g., in applications where the client's memory is limited (e.g., PDA devices) or to evaluate the computing performance of devices, etc.. Other protocols for creating secure benchmarks to evaluate a machine's computing performance were also proposed [44]; these benchmarks can however be easily parallelized [18,34,39].

6 Conclusion

In this paper, we proposed low-cost fixed-exponent and variable-exponent puzzles based on modular exponentiation. Given a modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. Our constructions are based on a reasonable intractability assumption: essentially the difficulty of computing a small private exponent in RSA (or CRT-RSA) when the public key is larger by several orders of magnitude than the semi-prime modulus. As a by-product, our puzzle can be used to efficiently verify the integrity of outsourced exponentiations modular a semi-prime. We further showed how our puzzle can be integrated in a number of protocols, including those used for the remote verification of computing performance of devices and for protection from DoS attacks.

Acknowledgments

The authors thank Rolf Wagner for implementing the repeated-squaring protocols. The authors also thank Cas Cremers, Patrick Schaller, Stephano Tessaro and Divesh Aggarwal for helpful discussions. Finally, the authors would like to thank the anonymous reviewers for their insightful suggestions and feedback.

References

1. PlanetLab, An open platform for developing, deploying, and accessing planetary-scale services, http://www.planet-lab.org/
2. Linpack, http://www.netlib.org/linpack/
3. Distributed.Net, http://distributed.net/
4. TOP500 Supercomputing Sites, http://www.top500.org/
5. Conroe Performance Claim being Busted, http://sharikou.blogspot.com/2006/04/conroe-performance-claim-being-busted.html

6. Computer Software Manufacturer agrees to settle Charges, http://www.ftc.gov/opa/1996/07/softram.shtm
7. Atallah, M.J., Pantazopoulos, K.N., Rice, J.R., Spafford, E.H.: Secure Outsourcing of Scientific Computations. In: Advances in Computers (2001)
8. Blomer, J., May, A.: Low Secret Exponent RSA Revisited. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, p. 4. Springer, Heidelberg (2001)
9. Boneh, D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the American Mathematical Society, AMS (1999)
10. Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. IEEE Transactions on Information Theory, 1339–1349 (2000)
11. Boneh, D., Schackam, H.: Fast Variants of RSA. In: CryptoBytes (2002)
12. Burns, J., Mitchell, C.J.: On parameter selection for server-aided RSA computation schemes. IEEE Transactions on Computers (1994)
13. Cai, J., Nerurkar, A., Wu, M.: The Design of Uncheatable Benchmarks Using Complexity Theory, ftp://ftp.cs.buffalo.edu/pub/tech-reports/./97-10.ps.Z
14. Chen, L., Morrissey, P., Smart, N., Warinschi, B.: Security Notions and Generic Constructions for Client Puzzles. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 505–523. Springer, Heidelberg (2009)
15. Coppersmith, D.: Finding a Small Root of a Univariate Modular Equation. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996)
16. Coppersmith, D.: Small solutions to polynomial equations and low exponent vulnerabilities. Journal of Cryptology, 223–260 (1997)
17. Coppersmith, D., Franklin, M., Patarin, J., Reiter, M.: Low-exponent RSA with related messages. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 1–9. Springer, Heidelberg (1996)
18. Cui-xiang, Z., Guo-qiang, H., Ming-he, H.: Some New Parallel Fast Fourier Transform Algorithms. In: Proceedings of Parallel and Distributed Computing, Applications and Technologies (2005)
19. Curnow, H.J., Wichman, B.A.: A Synthetic Benchmark. Computer Journal (1976)
20. de Weger, B.: Cryptanalysis of RSA with small prime difference. In: Applicable Algebra in Engineering, Communication and Computing (2002)
21. Dean, D., Stubblefield, A.: Using client puzzles to protect TLS. In: Proceedings of the USENIX Security Symposium (2001)
22. Doshi, S., Monrose, F., Rubin, A.: Efficient Memory Bound Puzzles using Pattern Databases. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 98–113. Springer, Heidelberg (2006)
23. Durfee, G., Nguyen, P.: Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt 1999. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 14. Springer, Heidelberg (2000)
24. Gao, Y.: Efficient Trapdoor-Based Client Puzzle System against DoS Attacks (2005)
25. Hastad, J.: Solving Simultaneous Modular Equations of Low Degree. SIAM J. Computing (1988)
26. Hinek, M.J.: Cryptanalysis of RSA and its variants. In: Cryptography and Network Security, Chapman & Hall/CRC (2009)
27. Hinek, M.J., Lam, C.C.Y.: Common Modulus Attacks on Small Private Exponent RSA and Some Fast Variants (in Practice). In: Cryptology ePrint Archive (2009)
28. Hinek, M.J.: Another Look at Small RSA Exponents. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 82–98. Springer, Heidelberg (2006)
29. Hohenberger, S., Lysyanskaya, A.: How to Securely Outsource Cryptographic Computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005)

30. Howgrave-Graham, N., Seifert, J.P.: Extending Wiener's Attack in the Presence of Many Decrypting Exponents. In: Proceedings of the International Exhibition and Congress on Secure Networking (1999)
31. Jochemsz, E., May, A.: A Polynomial Time Attack on RSA with Private CRT-Exponents Smaller Than $N^{0.073}$. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 395–411. Springer, Heidelberg (2007)
32. Juels, A., Brainard, J.: Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In: Proceedings of NDSS (1999)
33. Katzenbeisser, S.: Recent Advanves in RSA Cryptography. In: Advances in Information Security, vol. 3 (2001)
34. Keqin, L.: Scalable Parallel Matrix Multiplication on Distributed Memory-Parallel Computers. In: Proceedings of IPDPS (2000)
35. Koblitz, N.: A Course in Number Theory (1987)
36. Kaya Koc, C., Acar, T., Kaliski, B.S.: Analyzing and Comparing Montgomery Multiplication Algorithms (1996)
37. Martin, A., Burrows, M., Manasse, M., Wobber, T.: Moderately Hard, Memory-Bound Functions. ACM Transcations on Internet Technologies (2005)
38. May, A.: Secret Exponent Attacks on RSA-type Schemes with Moduli $N = p^r q$. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 218–230. Springer, Heidelberg (2004)
39. McGinn, S.F., Shaw, R.E.: Parallel Gaussian elimination using OpenMP and MPI. In: Proceedings of the International Symposium on High Performance Computing Systems and Applications (2002)
40. Miller, G.L.: Riemann's Hypothesis and Tests for Primality. In: Proc. Seventh Annual ACM Symp. on the Theory of Computing (1975)
41. Reiter, M.K., Sekar, V., Spensky, C., Zhang, Z.: Making peer-assisted content distribution robust to collusion using bandwidth puzzles. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 132–147. Springer, Heidelberg (2009)
42. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 120–126 (1978)
43. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock Puzzles and Timed-release Crypto. MIT Technical Report (1996)
44. Sedgewick, R., Chi-Chih Yao, A.: Towards Uncheatable Benchmarks. In: Proceedings of The Structure in Complexity Theory Conference (1993)
45. Tritilanunt, S., Boyd, C., Gonzalez Nieto, J.M., Foo, E.: Toward Non-Parallelizable Client Puzzles. In: Bao, F., Ling, S., Okamoto, T., Wang, H., Xing, C. (eds.) CANS 2007. LNCS, vol. 4856, pp. 247–264. Springer, Heidelberg (2007)
46. van Dijk, M., Clarke, D., Gassend, B., Suh, G.E., Devadas, S.: Speeding up Exponentiation using an Untrusted Computational Resource. In: Designs, Codes and Cryptography, vol. 39, pp. 253–273 (2006)
47. Wang, X., Reiter, M.: Defending Against Denial-of-Service Attacks with Puzzle Auctions. In: Proceedings of the IEEE Symposium on Security and Privacy (2003)
48. Wang, X., Reiter, M.K.: A Multi-layer Framework for Puzzle-based Denial-of-Service Defense. International Journal of Information Security (2007)
49. Waters, B., Juels, A., Halderman, J.A., Felten, E.W.: New client puzzle outsourcing techniques for DoS resistance. In: Proceedings of the ACM Conference on Computer and Communications Security (2004)
50. Wiener, M.: Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory, 553–558 (1990)

A Cryptanalysis of RSA with Large Public Key and Small Private Exponent

Consider an RSA system (N, e, d), where $N = pq$, p and q are large primes, and $e \in \mathbb{N}^+$ such that $e \geq N^2$, $\gcd(e, \phi(N)) = 1$ and d is small. Recall that in RSA, $e \cdot d - 1 = k \cdot \phi(N)$, where $\phi(N) = (p-1)(q-1)$ and $k \in \mathbb{N}^+$.

A.1 Resilience to the Continued Fraction Attack

Theorem 2. *Let $a, b, c, d \in \mathbb{N}^+$ such that $\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}$, where $\gcd(a, b) = 1$ and $\gcd(c, d) = 1$ Then, $\frac{c}{d}$ is one of the convergents in the continued fraction expansion of $\frac{a}{b}$. Furthermore, the continued fraction expansion of $\frac{a}{b}$ is finite with the total number of convergents that is polynomial in $\log(b)$.*

In [50], Wiener describes a cryptanalytic attack on the use of an RSA private key $d < N^{0.25}$, when $e < pq$. The attack makes use of an algorithm based on continued fractions that finds the numerator and denominator of a fraction in polynomial time when a close enough estimate of the fraction is known. This will enable the retrieval of a multiple of $\phi(N)$, which will equally result in the factoring of N [40]. The convergence of the continued fraction algorithm is guaranteed when $kd < \frac{pq}{\frac{3}{2}(p+q)}$.

When $e \geq N^2$, $k \geq dpq$. Substituting $k = dpq$ in the equation above yields $d < 1$. More generally, when $e > N^{1.5}$, Wiener's attack will fail since the continued fraction algorithm will not work for any size of the secret exponent d [50].

A.2 Resilience to the Lattice-Based Attack

Boneh and Durfee [10] describe a scheme that solves the RSA small-inverse problem when $e < N^{\delta}$ and $d < N^{\alpha}$. As shown in [10], this attack is a heuristic that applies Coppersmith's techniques [15] to bivariate modular polynomials and can only succeed when $\alpha < \frac{7}{6} - \frac{1}{3}\sqrt{1 + 6\delta}$.

Indeed, when $\delta = 1$, $e \leq N$, we achieve the bounds reported in [10]: RSA is insecure when $d < N^{0.292}$. However, when $e \geq N^2$, $\delta > 2$, then this attack will definitely fail ($\alpha < -0.0.35$).

B Cryptanalysis of $ex + y \equiv 0 \mod \phi(N)$

B.1 Single Instance of $ex + y \equiv 0 \mod \phi(N)$

In [8], Blömer *et al.* describe a cryptanalytic attack (based on Wiener's continued fraction algorithm [50]) on a generic RSA key equation of the form $ex + y \equiv 0 \mod \phi(N)$, when $e \leq N, 0 < x < \frac{1}{3}N^{\frac{1}{4}}$ and $|y| < cN^{\frac{-3}{4}}ex$, where $c \leq 1$.

Let $ex + y = k\phi(N) = k(N - p - q + 1)$, where $k \in \mathbb{N}^+$. It then follows that:

$$\frac{e}{N} - \frac{k}{x} = \frac{-y - k(p + q - 1)}{Nx}.$$

The main intuition behind the attack in [8] is to estimate $\frac{k}{x}$ from $\frac{e}{N}$ using the continued fraction algorithm. For the attack to be successful, $\frac{k}{x}$ has to be one of the convergents of $\frac{e}{N}$. This is the case when $|\frac{e}{N} - \frac{k}{x}| = |\frac{-y-k(p+q-1)}{Nx}| < \frac{1}{2x^2}$ (see Theorem 2); that is, when $k(p+q-1)+y < \frac{N}{2x}$.

When $e \geq N^2$, $k \geq N\phi(N)$ ($|\phi(N)| \approx |N|$). It is easy to see in this case that the continued fraction algorithm will not converge ($k(p+q-1)+y \gg \frac{N}{2x}$).

B.2 Multiple Instances of $ex + y \equiv 0 \mod \phi(N)$ with Common Modulus)

Gao (described in [30]) and Howgrave-Graham and Seifert [30] extended Wiener's attack to factor the common modulus when several instances of RSA with $e \leq N$ and $d < N^{0.4-\epsilon}$ are given.

In what follows, we show that these attacks are defeated given several common modulus instances of $ex + y \equiv 0 \mod \phi(N)$ with $e \geq N^2$.

Let $(N_1, e_1), (N_2, e_2)$, be two instances of RSA, then there exists $k_1, k_2 \in \mathbb{N}^+$ such that:

$$e_1 x_1 = -y_1 + k_1 \phi(N)$$
$$e_2 x_2 = -y_2 + k_2 \phi(N)$$

Guo's main observation is that these equations can be combined to remove $\phi(N)$ as follows $k_2 e_1 x_1 - k_1 e_2 x_2 = k_1 y_2 - k_2 y_1$.

With this equation as a starting point, the attack then proceeds in a similar way as Wiener's continued fraction attack:

$$\frac{e_1}{e_2} - \frac{k_1 x_2}{k_2 x_1} = \frac{k_1 y_2 - k_2 y_1}{e_2 k_2 x_1}$$

Given Theorem 2, this suggests that $\frac{k_1 x_2}{k_2 x_1}$ can be obtained from the continued fraction expansion of $\frac{e_1}{e_2}$ when:

$$\left| \frac{k_1 y_2 - k_2 y_1}{e_2 k_2 x_1} \right| < \frac{1}{2(k_2 x_1)^2}$$
$$2k_2 x_1 |k_1 y_2 - k_2 y_1| < e_2$$

When $e_1 > N^2$ and $e_1 > N e_2$, then $2k_2 x_1 |k_1 y_2 - k_2 y_1| \approx \frac{e_2}{\phi(N)} x_1 (\frac{N e_2}{\phi(N)} y_2 - \frac{e_2}{\phi(N)} y_1) \gg e_2$. The continued fraction algorithm will not converge and this attack with then fail. This attack will fail even when $x_1 = x_2$.

Howgrave-Graham and Seifert's attack [30] combines Wiener's, Boneh's and Guo's attacks to factor N given $r \geq 2$ instances of RSA with common modulus. When $e_i > N^2$ and $e_i > N e_j, \forall i \neq j$, their attack will equally fail given any number of common modulus instances[12].

[12] The complexity of existing attacks on common modulus RSA instances increases exponentially with the number of instances; these are only practical for a small number of instances [30], [27].

Expressive, Efficient and Obfuscation Resilient Behavior Based IDS

Arnur G. Tokhtabayev, Victor A. Skormin, and Andrey M. Dolgikh

Center for Advanced Information Technologies, Binghamton University
{atokhta1,vskormin,adolgik1}@binghmaton.edu

Abstract. Behavior based intrusion detection systems (BIDS) offer the only effective solution against modern malware. While dynamic BIDS have obvious advantages, their success hinges upon three interrelated factors: signature expressiveness, vulnerability to behavioral obfuscation and run-time efficiency of signature matching. To achieve higher signature expressiveness, a new approach for formal specification of the malicious functionalities based on abstract activity diagrams (AD) which incorporate multiple realizations of the specified functionality. We analyzed both inter and intra-process behavioral obfuscation techniques that can compromise existing BIDS. As a solution, we proposed specification generalization that implies augmenting (generalizing) otherwise obfuscation prone specification into more generic, obfuscation resilient specification. We suggest colored Petri nets as a basis for functionality recognition at the system call level. We implemented a prototype IDS that has been evaluated on malicious and legitimate programs. The experimental results indicated extremely low false positives and negatives. Moreover, the IDS shows very low execution overhead and negligible overhead penalty due to anti-obfuscation generalization.

Keywords: IDS, Dynamic behavior detection, Behavior obfuscation, Behavior metamorphism, Colored Petri Nets.

1 Introduction

Behavior based intrusion detection systems (BIDS) offer the only effective solution against modern malware. While dynamic BIDS have obvious advantages, the success of BIDS is limited by three interrelated factors: signature expressiveness, vulnerability to behavioral obfuscation and signature matching efficiency. Signature expressiveness determines the success of IDS in detecting new realizations of the same malware. Since most malware are derivatives of other malware, the signature must capture invariant generic features of the entire malware family and be expressive enough to reflect most possible malware realizations. Behavioral obfuscation/ metamorphism is an emerging threat that, given the extensive development of BIDS, is expected to become a standard feature of future malware[1].

Based on this reality, we developed a novel system call domain IDS addressing limitations of BIDS. To enhance signature expressiveness, we proposed an approach to specify the functionalities of interest, specifically malicious ones, via abstract activity

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 698–715, 2010.

diagrams (AD) which can incorporate multiple realizations of the functionality. We studied possible inter and intra-process behavioral obfuscation approaches. As a mitigation solution, we proposed the concept of the specification generalization resulting in resiliency to obfuscations. Finally, we utilized Colored Petri Nets (CPN) for run-time detection of the specified functionalities at the level of system calls, and developed a procedure that automatically converts ADs into CPNs.

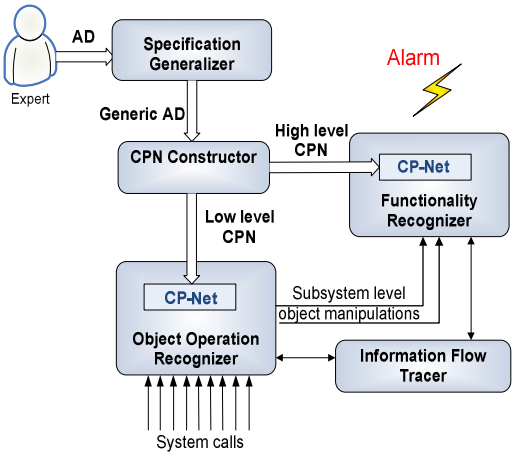


Fig. 1. System architecture

The IDS architecture is shown in Figure 1. At the learning phase, an expert designs ADs of known malicious functionalities. The Specification Generalizer module augments the ADs into more generic, obfuscation resilient ones. The CPN Constructor generates low and high-level CPNs from generic ADs. At the detection phase, the low-level CPN recognizes individual subsystem (user level) object operations in the system call domain thus aggregating system call information for processing at the higher level. The high-level CPN detects the functionalities in the domain of OS object operations. Also, the recognizers have an access to Information Flow Tracer to feed data dependencies for particular transitions of CPNs.

This paper delivers the following contributions.

- Formal functionality specification by AD defined at the system object level.
- AD abstraction via functional objects encapsulating various system resources.
- Analysis and classification of possible behavioral obfuscations.
- Automatic AD generalization neutralizing the effects of behavioral obfuscations.
- Automatic AD to CPN transformation. Such CPN recognizes the functionality in the system call / information flow domain.
- Efficient CPN simulator for recognizing functionalities in the system call / information flow domain.

2 Functionality Specification and Abstraction

Processes invoke system calls to operate system objects to facilitate some semantically distinct actions, such as sending a file to a specified IP address. We define functionality as a chain of such actions achieving a certain high-level objective.

Analogously with [3] and [4], we formulate the following requirements for the functional specifications:

1. *The specification must define the control flow for object operations*. It must support conditional branching and concurrent execution allowing for specifying alternative/independent object manipulation sessions.

2. *The specification must define data/information flow among object operations.* An attribute data flow determines the discriminatory power of the specification. We should point out that data may be passed by value constituting data flow or may be passed by information constituting information flow [5].

In addition, we introduce the third requirement to overcome multi-processes limitations such as in [4].

3. *The specification shall not be constrained to the context of one process.* This allows for specifying inter-process functionalities by relating objects and operations invoked by different processes.

The above requirements can be met by utilizing UML Activity Diagrams (AD) [6]. As an example, figure 2 depicts the AD of the Remote Shell functionality in both graphical and analytical forms. This functionality establishes a backdoor allowing an attacker to remotely execute system commands. In the figure 2, the decision node "a" starts two alternative realizations of the functionality. The left branch (nodes 1-2) represents the first step of the Reverse Shell realization. The right branch (nodes 3-6) represents the first step of the Bind Shell realization that has two independent sessions (nodes 3-4 and 5-6). Node 7 is the common step for both realizations.

Formally, functionality is specified is as an AD tuple:

$$F=(Nodes, Arcs, Assign, Vars) \qquad (1), \text{ where}$$

Nodes is a multi-set defined in Line 2 (Fig. 2). It consists of State and Pseudo nodes. There are two types of State nodes: Instances and Manipulations (line 3). Each

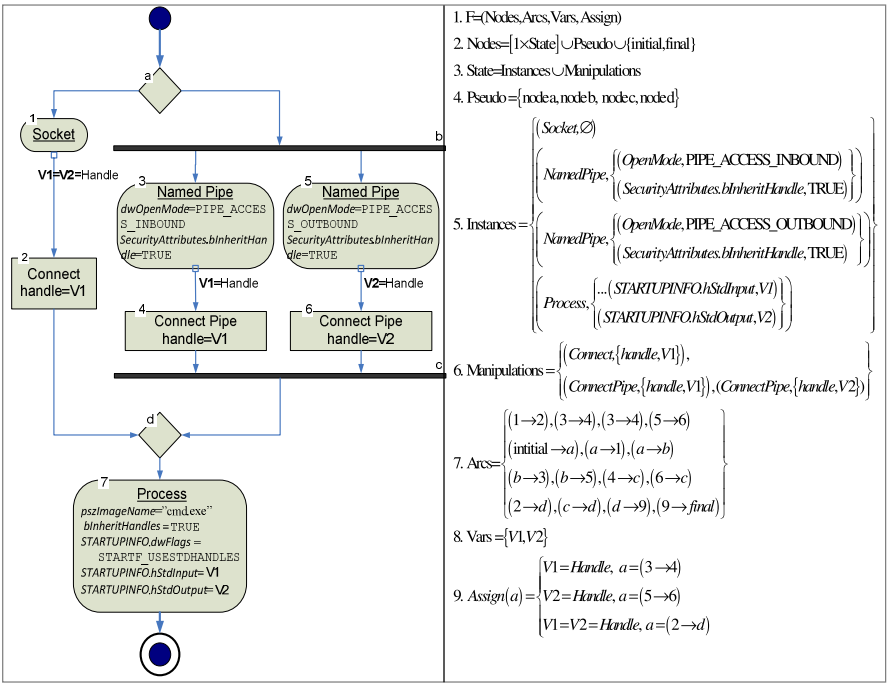


Fig. 2. Activity diagram of Remote Shell functionality

Instance node represents a created object and its attributes (e.g. line 5). Each *Manipulation* node represents an object operation with the parameters (e.g. line 6).

Arcs is a set of directed arcs showing the execution flow of the AD nodes.

Vars is a set of local variables utilized to define data flow among object operations. An information flow is specified via special transformation notation *T*().

Assign is a function that binds variable assignment expressions to corresponding arcs. Line 9 shows the definition of such a function indicating that V1 is assigned with handle created in node 2 or node 3.

To address the third specification requirement, each *State* node is assigned with a unique index of the process that performs the manipulation represented by the node.

Since our system is expert driven a specification must be as generic as possible enabling an expert to concentrate only on the conceptual realizations. To achieve this, we introduced a so-called functional objects that represents certain semantically distinct functionalities such as Inter-Process Communication (IPC), File Download, etc. Functional objects have a set of operations representing high-level activities. While specifying an AD, the expert may create and manipulate functional objects as ordinary OS objects. Technically, each functional object abstracts several alternative realizations of the particular functionality by encapsulating necessary OS objects utilized in these realizations.

Due to space limitations, in this paper we describe only limited set of functional objects, particularly the ones facilitating data transfer (Table 1). This set is just an example; however, it demonstrates the generalization power of our specification, which is the ability of utilization functional objects by other functional objects as with FileTransfer object. Object "GenericFile" abstracts file access operations; it encapsulates both "file" and "file mapping" objects. "RemoteIPC" represents IPC encapsulating socket, pipe and mailslot. For an expert, the utilization of such functional objects is transparent. For instance, when using RemoteRPC in a specification, the expert should not make any assumptions on how a malware will perform IPC, through socket, pipe or mailslot. Armed with such functional objects, an expert can build quite generic specifications yet preserving its discriminatory power.

Table 1. Functional objects for data transfer

Functional object	Based on objects	Operations	Attributes (input => output)
GenericFile	File, FileMapping	Create	FileName
		Read	BufferLength => Buffer
		Write	FileName, Buffer
RemoteIPC	Socket, Pipe, MailSlot	Create	EndPoint (server, client), ID => Type, Handle
		Accept	Type, Handle
		Recv	Type, Handle => Buffer
		Send	Buffer, Type, Handle => Buffer
LocalIPC	GereicFile, FileMapping RemoteIPC	Create	EndPoint (server, client), ID => Type
		Recv	=>Buffer
		Send	Buffer
FileTransfer	GenericFile, RemoteIPC	Create	FileName, RemoteHost => Type, FileHandle, IPC
		Send	(Type, FileHandle, IPCHandle)
		Recv	(Type, FileHandle, IPCHandle)

3 Mitigating Behavior Obfuscations

Since our approach is signature based, the discriminatory power of a specification mostly depends on an expert. Using functional objects, the expert may encompass most of the functionality realizations. Consequently, an attacker will have difficulty to discover yet another conceptually different realization with different system objects. However, to evade detection, the attacker does not have to implement a completely new realization. It is enough to simply obfuscate a known realization breaking the specification. We distinguish inter and intra-process approaches to obfuscation. Inter-process obfuscation techniques utilize multiple processes. Intra-process obfuscation locally alters a functionality realization while preserving its behavioral semantics.

Consider possible inter-process obfuscation approaches.

1. *Utilization of legitimate third party utilities to perform required activity.* A malicious process may run legitimate utilities to execute some important tasks being a part of the functionality. In this way, the process executes the functionality without performing some key object manipulations involved in the task. For instance, a file virus usually searches for executables using "FindFirstFile" and "FindNextFile" API. Instead, the virus may utilize system command interpreter to retrieve a list of executable files in a folder and then access the files one by one.

2. *Inter-process functionality distribution (multipartite approach).* A multipartite malware distributes its functionality among several benign processes by self-injecting to the legitimate processes or by creating the new ones. While such processes individually exhibit no malicious behavior, their combined activity represents an inter-process malicious functionality. Examples of such malware are Key-Logger or K-ary virus consisting of two processes [7]. Process A opens an executable file and passes the file handle to process B. Process B attaches the code of process A to the opened victim file. This scenario does not involve actions deemed malicious: process B does not open the victim file and does not inject its code; process A gets replicated into the victim file without performing write operation or self-access operations.

Now, consider possible intra-process behavioral obfuscation.

3. *Object relocation and duplication.* Since a functionality may be constrained by a particular object parameter (e.g. file name), an attacker can change the parameter of the object (e.g. copy, rename or move an image) before manipulating it. An object handle may be duplicated during the manipulation sequence to break system call binding. Also, an attacker may access objects via symbolic links instead of handles.

4. *Non-direct object manipulation.* It is achieved by specific, low-level system tricks such as utilization of non-trivial OS resources that allow for accessing objects whether in unusual way or via a "middleman" object. For instance, an attacker can create reparse points or access files by their streams. He also may add an alternative path to a target file through relinking system calls.

To mitigate the behavioral obfuscations, we propose the concept of *specification generalization* that would fill experts' experience/attention gap. Generalization algorithms suggested below augment a given AD making it less prone to obfuscations.

TraceFiles – Augments the given AD with functionalities tracing renaming and relocation of all files involved in the specification (addresses obfuscation #3).

TraceHandles – Augments the given AD with functionalities that trace object handle propagation among processes, which requires tracking handle duplication and IPC used for handle transfer. It addresses the first three obfuscation techniques.

TraceProcesses – Augments the given AD with functionalities that track process generation, remote code injection and inter-process coordination. This involves detecting several realizations of code injection including remote thread based and remote hook based. The augmented AD would relate object manipulations performed by multiple processes. This algorithm mitigates obfuscation techniques #1 and #2.

To address obfuscation #4, we do not need any post-processing of the AD. Instead, we can simply extend functional objects with necessary semantics that would trace low-level objects involved in the obfuscation. This addresses the obfuscation at the stage of AD specification, rather than automatic post-generalization. Particularly, we add reparse points and file streams into "GenericFile" functional object.

The proposed algorithms augment AD with special *generalization* functionalities which trace certain activity involved in a particular obfuscation. These functionalities maintain certain global variables that qualify the traced activities, e.g. generated processes, duplicated files or established IPC channels. The algorithms also utilize several *primitive functions* that process and modify an AD.

TraceFiles algorithm iterates over operations and object instances (line 1). If an operation has "file name" as an argument (line 2), the procedure provides the AD with "FileRelocation" generalization functionality (line 6) that traces all duplicates of a given file. If the file name is a variable, we start the parallel flow right after the node where the variable is assigned for the last time. Finally, we join "FileRelocation" parallel flow with the original AD right before the node performing the operation on the target file. The "FileRelocation" is added by *AddParallelFunct(Origin,New, Start,Merge)* function. It adds an AD named *New* to an AD named *Origin* as a parallel flow that starts right after the node *Start* and joins to the AD *Origin* just before the node *Merge*. The node *Start* is determined in lines 4, 5. If the file name is a variable, the node *Start* is determined through the function - *GetAssignNode(x)* that returns the node which output arc has an assignment expression for variable *x*. Line 7 modifies the AD to make it consistent with the AD formalism (1).

TraceHandles augments the original functionality, in Line 1, with parallel flow with "HandleDuplication" functionality that traces all handles derived from the given object handle. Lines 2-8 constitute a loop that iterates over all object instances of the

Algorithm **`TraceFiles`**

Input: **AD** - An activity diagram specification

Output: Generilized AD

---

1. **foreach** $Operation \in \{\text{AD.Instances} \cup \text{AD.Manipulations}\}$**:**
2. **if** $(lpFileName \in \text{AttList}(Operation))$ **:**
3. $TargetFileName := \text{GetAttributeValue}(Operation, lpFileName)$;
4. **if** $(isVariable(TargetFileName))$ **:** $RelocStartNode = GetAssignNode(TargetFileName)$
5. **else :** $RelocStartNode = \text{AD.initial}$;
6. $\text{AddParallelFunct}(\text{AD}, \text{FileRelocation}(TargetFileName), RelocStartNode, Operation)$;
7. $\text{SetAttributeValueExpression}\left(\begin{array}{l} Operation, lpFileName, \\ "lpFileName\ in\ FList["+TargetFileName+"]" \end{array}\right)$;

AD. For each instance, a new element in *DupH* dictionary is initialized with *PID* and *Handle* of the instance (lines 3, 4). It sets "HandleDuplication" to trace handle duplicates of the current object instance. Line 6 iterates over object operations performed on the current object instance. For each object operation, the algorithm redefines *PID* and *Handle* expressions to allow the operation to utilize any duplicated handle belonging to original object instance.

In the first obfuscation, a malware performs system tasks via an external utility that, in its turn, uses the OS resources the same way as the malware would. Hence, malware simply outsources its operations to the utility. From this perspective, starting a utility to perform a part of the malicious functionality represents a multipartite approach. Hence, the first two obfuscations should be addressed similarly: by tracing functionality distribution among several processes. This requires tracing processes generated by the malware as well as processes to which malware injected its code. Then we attribute object operations to the generated/infected processes.

TraceProcesses addresses the first two obfuscations. It introduces "ProcessGeneration", "CodeInjection" and "LocalIPCEstablishment" functionalities to the input AD. "ProcessGeneration" and "CodeInjection" generalization functionalities trace all descendant and injected processes from the original given process. "LocalIPCEstablishment" functionality tracks all IPC channels (with handles and IDs) established by the given set of processes. The algorithm also introduces IPC required for coordinating multipartite agents and/or communicating with the utility.

We additionally trace data transmission between processes that represents technical yet vital activity. For instance, a process retrieves (reads) data through an object, representing data source, and then this data or its informational dependency is transferred (written) through another object, called data sink. Distributing this activity in a way that one process would access a source object and another process would access a sink object requires using IPC responsible for data transmission from the source process to the sink process. Such functionality in fact implements an inter-process information link between source and sink objects. Data source and sink could be presented by the following OS/functional objects: File, Pipe, Mailslot (kernel32.dll - Read, Write), Socket (ws2_32.dll - Recv, Send), Registry (Advapi32.dll - ReadValue, WriteValue) and RemoteIPC, LocalIPC (functional - Recv, Send).

Algorithm **TraceHandles**

Input: **AD** - An activity diagram specification

Output: Generilized **AD**

---

1. AddParallelFunct(AD, HandleDuplication, AD.initial, AD.final);
2. **foreach** *Object* ∈ AD.Instances **:**
3. SetAssignExpression(OutputArc(*Object*), "DupH[*Handle*][*PID*] = {*Handle*}");
4. SetAssignExpression(OutputArc(*Object*), "DupP[*Handle*] = {*PID*}");
5. *HandleVarName* = CreateNewVar(OutputArc(*Object*), "*Handle*");
6. **foreach** *Operation* ∈ GetObjectOperations(AD,*Object*) **:**
7. SetNodePIDExpression(*Operation*, "PID in DupP["+ *HandleVarName* + "]");
8. SetAttributeValueExpression(*Operation*, *Handle*, "*Handle* in DupH["+ *HandleVarName* + "][*PID*]");

As an example, we generalized "Remote Shell" AD (Fig. 3). Here, the fork node "b" starts two sessions. The left session corresponds to the first steps of "Reverse Shell" and "Bind Shell" realizations (nodes 1-3). The right session constitutes single operation - "FileRelocation" (node 4) that traces "cmd.exe" file and outputs a list of files descending from it. Nodes 1, 2 create RemoteIPC objects which handle is traced by "HandleDuplication" functionality (node 6). Expression "PID in PList" in nodes 1, 2 means that PID of the process performing the operation must belong to the PList. Nodes 1, 2 and 3 represent inter-process part of the "Remote Shell" functionality. Such inter-process part along with node 7 addresses obfuscation techniques #2 and #3. Indeed, nodes 1, 2, 3 outsource IPC creation to other processes.

The final step of the "Remote Shell" is to run "cmd.exe". Node 5 creates a process which image belongs to the list of files originated from "cmd.exe". Note that this FList was produced by "FileRelocation" functional operation (node 4). Moreover, the process is created with standard input set to duplicate/original handle of the IPC end-point, server or client. Note, the generic specification AD (Fig. 3) defines six different realizations against two of the original AD (Fig. 2).

The more obfuscation types we address, the more complex the generalized specification is expected to be. However, the specification is not yet a recognition mechanism since it merely represents how the functionality is implemented in terms of object manipulations. Hence, the efficiency of a recognition mechanism determines how many obfuscations we can address. We proposed highly efficient recognition model that is scalable enough to detect specifications with all discussed obfuscations.

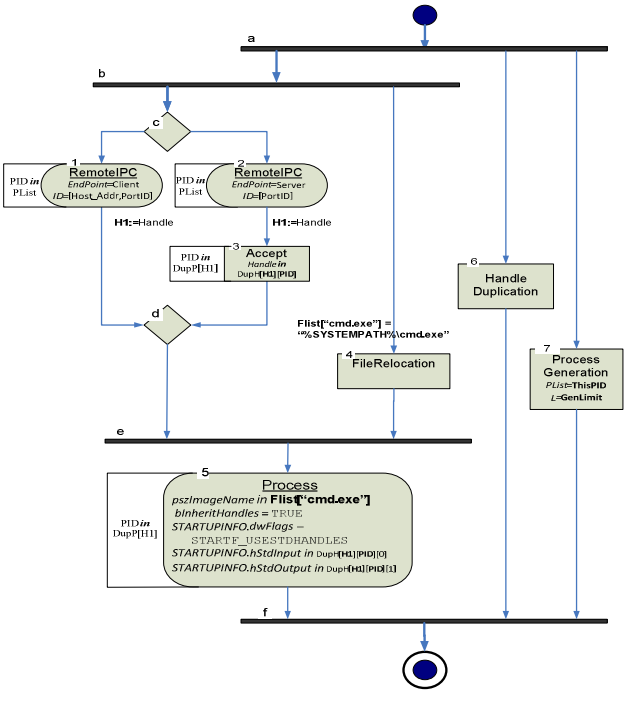


Fig. 3. Generalized AD for Remote Shell

4 Functionality Recognition

Functionalities are recognized in two stages (Fig. 1). At the first stage, we recognize high-level object manipulations as their dedicated API functions in the system call domain. At the second stage, we recognize functionalities at the level of the identified object manipulations, i.e. APIs. Since an API may invoke several minor system calls that are not critical for the manipulation implementation, only the essential, semantically critical part of the API is recognized. It mitigates an evasion technique when a malware does not invoke the entire API but only its critical system calls, thus only partially realizing the API yet achieving the required manipulation.

Since AD specification (1) defines data dependencies among system call attributes, it represents a context sensitive language. Therefore, in general, a functionality defined by AD formalism (1), can be recognized by any contest-sensitive acceptor such as linear bound automata (LBA) [9]. However, state machines such as LBA are disadvantageous in processing multiple instances of the operation chains (words): should a chain be executed more than once, an LBA model reserves extra states for each instance of the chain. In contrast, a CPN is free of such a drawback, because it represents an executed operation chain as one token residing in the corresponding place that allows for processing of multiple chain instances with low overhead [10]. In a way, this justifies the choice of CPN as a recognition model.

A CPN is defined as a tuple [11]: $CPN=(S,P,T,A,N,C,G,E,I)$ (2)

where: S – color set, P – set of places, T – set of transitions, A – set of arcs , N – node function, C – color function, G – guard function, E – arc expression function, I – initialization function.

CPN places must represent such states as: object instances; object manipulations; pseudo states routing the control flow of AD and functionalities.

Hence, formally *set of places* of the CPN consists of four disjoint dedicated subsets – *Object* places, *Manipulation* places, *Functional* places and *Pseudo* places:

$$P=P_{obj} \cup P_{manip} \cup P_{fun} \cup P_{pseudo},$$

such that, each Object place is associated with a unique OS object; every Manipulation place represents a particular (individual) operation of an object; any Functional place corresponds to a unique functionality and a Pseudo place is associated with pseudo states of the AD.

Object place tokens represent instances of the object. Such a token is defined as a tuple: handle of the object instance and set of object parameters. The color set of Object places typically constitutes a pair of two sets: system handles and a set of object attributes such as: file names, access flags, buffer address, etc.

Manipulation place tokens represent successful execution of corresponding operation with an object instance. Each token comprises a handle of the manipulated object and critical parameters of the operation represented by the place. The color set of Manipulation places consists of space of system handles and set a set of selected parameters of possible operations on objects involved in the functionality.

Functional place tokens represent successful recognition of the given functionality. Color set of Functional places includes selected attributes of necessary objects involved in the respective functionality as well as objects operation parameters that individualize the functionality.

Places of CP-nets represent executed object operations; therefore a transition must be attributed to execution of one of the equivalent API functions or system calls implementing the respective manipulations. Hence, the set of transitions consists of three disjoint sets: $T = T_{man} \cup T_{fun} \cup T_{pseudo}$, where

T_{man}- manipulation transitions representing system calls/API. T_{fun} - functional transitions such that their input and output places constitute functionalities, T_{pseudo}- pseudo transitions reflecting AD pseudo states.

Each manipulation transition corresponds to a particular system call or group of functionally equivalent APIs performing the same manipulation, and the transition is enabled when the system call or one of the equivalent APIs has been invoked. Transition guard expressions check manipulation handles and parameters to ensure that transitions are enabled only by manipulations with correct attributes, specified in the AD. The arc expressions generate tokens containing necessary attributes of the system calls/APIs utilized in AD. This provides enough flexibility to distinguish between similar yet semantically different functionalities.

We developed procedure "*ADtoCPN*" which translates the given AD to CPN possessing the necessary execution semantics for functionality recognition. Due to limitations, we describe only high-level steps.

Procedure *ADtoCPN*
Input: F – an AD of the functionality defined by the formalism (1).
Output: CPN – a CP-net that recognizes the given functionality F.

1. Compose the CPN structure (P, T, A) corresponding to the constructs of the AD of the functionality. Arcs of the AD are replaced by transitions and nodes are replaced by places.
   - 1.1 Form a set of places P and set of transitions T that correspond to the state and pseudo state nodes of the functionality F.
   - 1.2 Form a set of the CPN arcs (A) connecting the places and transitions created in the previous step (1.1)
   - 1.3 Form a set of functional places, transitions and corresponding arcs.
2. Define place colors (C), guard expressions (G) and arc expressions (E) that define execution semantics of the functionality F in the given domain.
   - 2.1 Define guard expressions of the manipulation transitions that check the executed manipulation parameters against parameters specified in the functionality's AD.
   - 2.2 Define guard expressions at the transitions that represent branching arcs of the AD decision nodes.
   - 2.3 Define a color function (C) that would reflect variables of the functionality.
   - 2.4 Define arc expressions representing variable assignment in the functionality's AD.
   - 2.5 Induce Color set (S) and the rest of the arc expressions from the color function (C) and the CPN structure (P,T,A)
3. Compile a CP-net ($CPN=(S,P,T,A,N,C,G,E,I)$) from the component sets obtained in Steps 1 and 2.

5 Experimental Evaluation

All experiments were conducted in the network testbed at Binghamton University [19-20]. Our network comprised dozens victim hosts, i.e. virtual machines with vulnerable Windows OS equipped with our IDS. We experimented with various malware known to expose the following potentially malicious functionalities.

Replication engines:

- R1. *Self code injection* – a malware infects an executable file through injecting its code into the executable body and replacing code entry points. It is used by file viruses.
- R2. *Self mailing* - a malware emails its image as an attachment. It is used by e-mail worms
- R3. *Executable Download and Execute* – Downloads a file from Internet and executes it. Used as a part of self-propagation engine of network worms [19], hence exposed by exploited processes. Also, exposed by network bot agents such as Trojan-downloaders.
- R4**.** *Remote shell* - Exposed by net-bots payloads and worms propagation engines (see section 2).

Malicious payloads:

- P1. *Dll/thread injection* - Injects DLL/thread to a process for password stealing or control highjacking.
- P2. *Self manage cmd script create and execute* – Malwrare creates cmd script and executes it by cmd interpreter. The script manages the malware image/dlls after its termination. Usually, this functionality relocates/deletes the malware image to conceal its footprint. Afterwards, the script usually erases itself.
- P3. *Remote hook* - sets a remote hook into victim process for a particular event. Used for keylogging.

These functionalities were specified, generalized and translated to CPNs. To decrease simulation overhead, we integrated the high-level CPNs into a single universal CPN having a recognition place for each given malicious functionality. Since the given functionalities share the same object operation sessions, by integrating recognition CPNs, we eliminated CPNs structural redundancy, hence minimized combined recognition overhead. The low-level CPNs are also integrated into a single CPN detecting object operations involved in the functionalities. The CPN configurations finally were loaded to the Recognizer modules.

To verify the detection rate, we experimented with the following malware, that according to AV descriptions, are known for perpetrating at least one of the malicious functionalities:

- *7 File viruses* – Neo, Abigor, Crucio, Savior, Nother, Halen, HempHoper
- *10 Network worms* - Welchia.A, Sasser.C, Bozori, Iberio, HLLW.Raleka.A, Alasrou.A, Kassbot, Shelp.A, Blaster, Francette
- *9 E-mail worms* –5 variants of w32.Netsky and 4 variants of w32.Beagle
- *Network bots/Trojans* – SpyBot.gen, IRC.SdBot, RxBot families, Win32.Banker, Win32.lespy

We run malware in the corresponding environment enabling it to execute its payloads or replicate properly. To trigger replication activity, we utilized our previous setup to attack victim hosts with various worms [19], [21]. To invoke malicious payloads, we executed malware in certain preset conditions, for instance, we established an ftp/tftp server for executable download and execute functionality. In some cases, we had to enforce malware strains to run their payloads through debugging with run-time code modification.

In order to evaluate the false positive rate, we run multitude of benign software that include web-browsers, messengers, email clients, file utilities, network and system utilities and office tools. We run the tested software under various conditions/inputs to expose their functionalities. We should point out that our experiments hardly covered all execution branches of the tested programs missing certain minor behavior. Nevertheless, we believe that the main activity of the tested software was exposed in our experiments.

The experiment results are depicted in Table 2, where the upper part indicates detection results for the legitimate software. Here, each cell indicates how many legitimate programs were detected by our IDS with the given functionality. The lower part of the table summarizes results for malicious software. For each malware set, we indicate how many instances, possessing the given functionality, were actually detected by our IDS. For example, 4/4 means that there are four instances from the set that have the given functionality and all four exposed it and were detected by IDS.

Table 2. Functionalities detection rate and false positive rate

			Self-replication		Replication/ payloads		Payloads		
			R1	R2	R3	R4	P1	P2	P3
Legitimate	201	System tools, office apps, other utilities					1	1	
	6	Web and file browsers (Opera, IE, FAR, Win Explorer)			3				1
	3	E-mail/messang. clients (Outlook, Eudora, Yahoo messang.)			1				1
		Total detected			4/210		1/210		2/210
Malware	7	File viruses	7/7						
	10	Network worm shell codes			2/2	8/8			
	6	Network worm payloads			4/4	1/1	1/1		1/1
	9	E-mail worms		9/9					
		SpyBot.gen family			all	all			all
		IRC.SdBot family			all	all		all	all
		RxBot family (11)			all	all	all	all	all
False positive rate			0%		1.92%	0%	0.48%		0.96%
Detection rate			100%						

False negatives (detection rate). As Table 2 indicates, for each malware that has the given functionality, our IDS successfully detected the functionality showing no false negatives. Such low false negative rate could be attributed to the signature generalization.

For instance, the Beagle worm drops itself into the system folder, and then it e-mails its dropper. However, our prototype system successfully detected the self-mailing activity because it traced the dropper as object relocation functionality.

During the experiment, we realized that malware strains within the same family rarely change/add a conceptually novel functionality realization. However, the new malware strains frequently introduce minor alterations to their functionality realizations such as utilization of alternative APIs or changing Local IPC, i.e. switching from named pipe to shared files. We see two reasons for this trend. Firstly, authors try to change malware system footprint in order to avoid certain AV signatures. Secondly, in case of net-bots, the authors simply try to increase performance of malware by optimizing or simplifying their implementation.

False positives. For the experiment, we run 210 legitimate programs including web browsers, e-mail clients, file managers, system and office tools, hooking software etc. Note that we mostly focused on the main features of each program. In Table 2, we distinguished and grouped 9 programs most of which exposed false positives. In the first row of the table, we summarize the rest 201 programs, almost free of false positives, including various applications (office, utilities) and system tools (from Windows system folder).

Table 2 indicates that eight programs out of 210 showed false positives. Below, we give possible reasons of why a particular functionality was exposed by certain legitimate software.

- *Executable Download and Execute* can be performed by internet browsers or file managers. Mostly, such activity is performed on behalf of an end-user. In addition, many programs periodically perform checking for updates. If there is an update available, the program downloads it and then executes it.
- *DLL/thread injection*. It can be performed by user/system monitoring software. Particularly, Easy Hook library injects DLL to trace API calls performed by arbitrary program. WinSpy program performs DLL injection in order to retrieve window objects data of a foreign program.
- *Self manage cmd script create and execute*. To uninstall hooks, the Easy Hook exiting functions run a *cmd* script that waits the hooking process to end, then removes the hooked DLLs.
- *Remote hook.* Hooking can be performed by chat programs to indentify whether a user is idle. These programs hook into other processes for the input events such as keystroke and mouse message.

Note that Table 2 also illustrates the discriminatory power of the functionalities frequently exposed by malware. For instance, according to Table 2, “Self-code inject”, “Self-mailing” and “Remote shell” have never been exposed by benign software, thus they have perfect discriminatory power and can be dependably used for malware detection. However, functionality “Executable Download and Execute” is too often exposed by benign software, such as web browser, consequently its discriminatory power is low, and it can not be recommended as a behavior signature. Hence, the detection capability of a particular functionality *should not be* attributed to the credibility of our approach. Indeed, our methodology allows for specifying, generalizing and detecting of functionalities, but selecting particular ones for the detection purpose presents a specific task that can be accomplished based on comparative analysis of their discriminatory powers.

6 Performance Overhead

The experiments were executed on Windows XP Professional SP2 running on an AMD Athlon 64 X2 (2200 Mhz) processor with 2 Gb of memory. We measured IDS overhead imposed on system and application tasks using commercial benchmarks and manual setup. Moreover, we estimated performance penalty for behavioral de-obfuscation. To achieve consistent results on Windows XP, we deactivated Windows prefetcher, scheduled tasks and accounted only for warm runs (to minimize cache influence). Some tests such, as file search and software installation were performed in virtual machine with reverting initial snapshot state for each run.

The test results are showed in Table 3 for Remote Shell functionality. We showed here a selected set of standard tests that are representative with respect to execution overhead. The table depicts five system and application tasks that intensively utilized OS resources (services) resulting in a large number of invoked system calls. Some tasks involved user interaction with GUI of the corresponding application. In these cases, we utilized TestComplete software [22] to simulate user behavior. We also run series of benchmarks using well-known PC Mark 05 suite [23]. Internet Explorer was tested with Peacekeeper benchmark [23]. We run each task/benchmark several dozen times with identical initial conditions and computed mean value of the execution time/score assuming normal distribution.

Table 3. Execution overhead imposed by IDS

	Benchmark/Application (Task description)	Execution time without IDS	Overhead (IDS On)	
			Basic CPN	Full CPN
Application/sys-tasks	Files Search (Search *.exe in c:\)	58.96	5.2%	7.96%
	Apps Installation (Install DirectX 9.0c)	112.3	1.15%	1.15%
	MS Word (Save a big file as rtf)	35.9	4.18%	4.18%
	WinRar (Compress Windows system folder)	292	2.05%	2.05%
	Internet Explorer 8 (Peacekeeper Browser Benchmark, www.futuremark.com)	702 (score)	5.3%	6.4%
PC Mark 2005	Application loading (Mb/sec)	4.96	1.84%	1.84%
	Web page rendering (pages/sec)	2.0332	7.08%	7.08%
	File Encryption (Mb/sec)	36.827	2.93%	2.93%
	XP Startup (Mb/sec)	5.88	2.21%	2.21%
Average execution overhead			3.55%	3.98%

To estimate qualitative the scalability of our IDS, we tested each task against two CPN configurations: *Basic* and *Full*. The Basic configuration covers alternative realizations of the functionality in question, but it does not employ functional objects. In contrast, the Full CPN is obtained by generalizing the Basic CPN. As a result, the Full

CPN uses functional (generic) objects and addresses all three obfuscations. To estimate quantitative scalability, our IDS observed all processes, but CPN recognizer in all performed tests.

For each task, Table 3 shows: base execution time when IDS is disabled (no system call monitoring or processing) and execution time overhead when IDS is enabled with both Basic and Full CPNs recognizing Remote Shell functionality (with monitoring all active processes). The table indicates that even Full CPN IDS does not impose much overhead (less than 4% in average), while monitoring more than 50 (all active) processes. In fact, we also run IDS with highly loaded Windows XP with more than 100 processes without any significant overhead. This result shows sufficient scalability to protect all processes of a modern OS.

It could be seen that generalization and de-obfuscation does not impose much overhead penalty which is 0.43% in average. Note that in some tests Full and Base CPN overheads are considered to be invariant under statistical hypothesis with 80% power. This shows that our IDS is highly scalable and can address much more behavioral obfuscations.

7 Related Work

Success of a Behavior based IDS is determined by two aspects: expressiveness of the signature specification language and efficiency of the recognition mechanism. Moreover, IDS usability depends on degree of abstractness of the specification language. We survey existing behavioral specification languages and discuss advantages of our approach. Then we show in what way our system is different and better than existing system call domain behavioral IDSs.

There are two types of specification languages widely used in the literature: State-Transition/CPN based specifications [24-27] and Declarative/Analytical Specifications [28-32]. CPN specifications are very efficient in recognition, however they are not abstract and less expressive. On the other hand, declarative and analytical specifications could be very generic and highly expressive, however in general, run-time recognition of such specifications may impose high overhead.

With respect to the specification language, our method takes the best of the both approaches. On one hand, the proposed AD-based specification is abstract due to functional objects, which allows for creating highly generic and expressive signatures free of implementation details. On the other hand, we proposed a highly efficient recognition mechanism via hierarchal CPNs defined in the detection domain – system calls with information flow. In our case, the separation of the specification domain from the detection domain was possible due to automatic translation of an abstract AD to the system coherent CPN.

The above papers do not address specification generalization and behavioral obfuscation issues, which is one of our main contributions. Since our goal is recognizing malicious functionality rather than malicious behavior, we have to provide all realizations (behaviors) of the functionality. Hence, automatic specification generalization becomes a critical issue. We augment specification with most realizations via both functional objects and generalization that also provides obfuscation resiliency. We understand that we addressed rather limited set of obfuscations, but given the flexibility of our functionality specification, developing new generalization algorithms for anti-obfuscation should be feasible for an expert.

8 Conclusion

In this paper, we addressed present and future limitations of the current Behavior Based IDS (BBIDS) in terms of signature expressiveness, behavioral obfuscation and detection efficiency. We proposed a formal specification of the malicious functionalities based on activity diagrams (AD) defined in the abstract domain. We advocated for the separation of the specification and detection domains, which is achieved via automatic translation of an abstract AD to a hierarchical CPN defined in the system call domain. In practice, such CPN showed itself to be a very efficient recognition mechanism. To achieve fine-grained recognition, we also incorporated information dependencies into AD specification and consequently to CPNs.

We analyzed both inter and intra-process behavioral obfuscation techniques that can compromise existing BBIDS. As a solution, we proposed a concept of specification generalization that implies augmenting (generalizing) otherwise obfuscation prone specification into more generic obfuscation resilient specification. We presented generalization algorithms making an AD immune to the obfuscations.

We implemented a prototype IDS with CPN simulator and Information Flow Tracer. The AD formalism is specified using standard UML AD constructs. In the prototype, the entire process of signature generation is automated, which includes computer aided AD design, automatic AD generalization and finally automatic AD translation to CPN. Experiments demonstrated that such an approach minimizes designing routine for an expert allowing him to concentrate on conceptual realizations omitting certain implementation details.

The IDS was evaluated on dozens of malware and hundreds of legitimate programs. In the experiments, we detected various malicious functionalities including self-replication engines as well as payloads. The results showed extremely low false positives and negatives. Finally, we performed series of experiments to estimate runtime overhead due to IDS. The results indicated two practical advantages. First, IDS causes extremely low execution overhead that is less than 4%. Second, the overhead increase due to the anti-obfuscation generalization is only 0.43%. Such low overhead difference between generalized and original CPN indicates that an expert can always address even more obfuscations with negligible execution cost.

Acknowledgment

This research is funded by the Air Force Office of Scientific Research (AFOSR). The authors are grateful to Dr. Robert Herklotz of AFOSR for supporting this effort.

References

[1] Parampalli, C., Sekar, R., Johnson, R.: A practical mimicry attack against powerful system-call monitors. In: Proc. ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008 (2008)

[2] Russinovich, M.E., Solomon, D.A.: Microsoft Windows Internals, 4th edn. Microsoft Press, Redmond (2005)

[3] Christodorescu, M., Jha, S., Kruegel, C.: Mining specications of malicious behavior. In: Proc. ACM SIGSOFT Symposium on the Foundations of Software Engineering (August 2007)
[4] Martignoni, L., et al.: A Layered Architecture for Detecting Malicious Behaviors. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 78–97. Springer, Heidelberg (2008)
[5] Cavallaro, L., Saxena, P., Sekar, R.: On the Limits of Information Flow Techniques for Malware Analysis and Containment. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 143–163. Springer, Heidelberg (2008)
[6] UML (2010), http://www.uml.org/
[7] Filiol, E.: Formalization and Implementation Aspects of K-ary (malicious) Codes. In: Brouck, V. (ed.) EICAR 2007 Special Issue (2007); Journal in Computer Virology 3(2) (2007)
[8] Visual Paradigm for UML (2009), http://www.visual-paradigm.com/
[9] Linz, P.: An Introduction to Formal Language and Automata, 4th edn. Jones & Bartlett Pub., USA (2006)
[10] Jones, N.D., et al.: Complexity of Some Problems in Petri Nets. Theoretical Computer Science 4, 277–299 (1977)
[11] Jensen, K.: Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use, 2nd edn., vol. 1. Springer, Berlin (1996)
[12] Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. 12th Annual Network and Distributed System Security Symposium, NDSS (2005)
[13] Egele, M., Kruegel, C., Kirda, E., Yin, H., Song, D.: Dynamic spyware analysis. In: Proc. USENIX Annual Technical Conference (June 2007)
[14] Volpano, D.M.: Safety versus secrecy. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, p. 303. Springer, Heidelberg (1999)
[15] Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, Behavior-Based Malware Clustering. In: Proc. NDSS (2009)
[16] Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proc. IEEE Security and Privacy (2007)
[17] Skormin, V., Volynkin, A., Summerville, D., Moronski, J.: Run-Time Detection of Malicious Self-Replication in Binary Executables. Journal of Computer Security 15(2), 273–301 (2007)
[18] Hoglund, G., Butler, J.: Subverting the Windows Kernel – Rootkits. Addison Wesley, Reading (2006)
[19] Tokhtabayev, A.G., Skormin, V.A., Dolgikh, A.M.: Detection of Worm Propagation Engines in the System Call Domain using Colored Petri Nets. In: Proc. 27th IEEE International Performance Computing and Communications Conference (IPCCC), Austin, TX (December 2008)
[20] Volynkin, A., Skormin, V.: Large-scale Reconfigurable Virtual Testbed for Information Security Experiments. In: Proc. of the 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, Orlando, FL, May 21-23 (2007)
[21] Tokhtabayev, A., Skormin, V., Dolgikh, A., Beisenbi, M., Kargaldayeva, M.: Detection of Specific Semantic Functionalities, such as Self-Replication Mechanism, in Malware Using Colored Petri Nets. In: Proc. SAM 2009, Las Vegas, NV (July 2009)
[22] TestComplete (2010), http://www.automatedqa.com/
[23] PCMark 2005, Peacekeeper (2010), http://www.futuremark.com/

[24] Kumar, S., Spafford, E.H.: A Pattern Matching Model for Misuse Intrusion Detection Approach. In: Proc. of the 17th National Computer Security Conference (1994)
[25] Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., Wang, Y., Wang, X., Stakhanova, N.: A Software Fault Tree and Colored Petri Nets based specification, Design and Implementation of Agent Based Intrusion Detection Systems. International Journal of Information and Computer Security 1(1/2), 109–142 (2007)
[26] Ho, Y., Frincke, D., Tobin Jr., D.: Planning, Petri Nets, and Intrusion Detection. In: Proceedings of the 21st National Information Systems Security Conference, Crystal City, Virginia (October 1998)
[27] Eckmann, S., Vigna, G., Kemmerer, R.: STATL: an Attack Language for State-based Intrusion Detection. In: Proc. of the ACM Workshop on Intrusion Detection, Athens, Greece (November 2000)
[28] Cuppens, F., Ortalo, R.: LAMBDA: A Language to Model a Database for Detection of Attacks. In: Debar, H., Mé, L., Wu, S.F. (eds.) RAID 2000. LNCS, vol. 1907, p. 197. Springer, Heidelberg (2000)
[29] Michel, C., Me, L.: ADeLe: An Attack Description Language for Knowledge-based Intrusion Detection. In: Proc. International Conference on Information Security, June 2001, Kluwer, Dordrecht (2001)
[30] Pouzol, J.-P., Ducassé, M.: From Declarative Signatures to Misuse IDS. In: Lee, W., Mé, L., Wespi, A. (eds.) RAID 2001. LNCS, vol. 2212, pp. 1–21. Springer, Heidelberg (2001)
[31] Ning, P., Jajodia, S., Wang, X.S.: Abstraction-Based Intrusion Detection In Distributed Environments. ACM Transactions on Information and System Security 4(4), 407–452 (2001)
[32] Meier, M., Bischof, N., Holz, T.: SHEDEL - A Simple Hierarchical Event Description Language for Specifying Attack Signatures. In: Proc. 17th International Conference on Information Security, pp. 559–571. Kluwer, Dordrecht (2002)

Author Index