Sushil Jajodia
Chandan Mazumdar (Eds.)

LNCS 7093

Information Systems Security

7th International Conference, ICISS 2011
Kolkata, India, December 2011
Proceedings

Springer

Sushil Jajodia Chandan Mazumdar (Eds.)

Information Systems Security

7th International Conference, ICISS 2011
Kolkata, India, December 15-19, 2011
Proceedings

Springer

Volume Editors

Sushil Jajodia
George Mason University, Center for Secure Information Systems
4400 University Drive, Fairfax, VA 22030-4422, USA
E-mail: jajodia@gmu.edu

Chandan Mazumdar
Jadavpur University, Center for Distributed Computing
Kolkata 7000032, India
E-mail: chandan.mazumdar@gmail.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25559-5 e-ISBN 978-3-642-25560-1
DOI 10.1007/978-3-642-25560-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011941662

CR Subject Classification (1998): C.2.0, C.2, E.3, H.3, D.4.6, K.6.5, K.4.4, H.2.4

LNCS Sublibrary: SL 4 – Security and Cryptology

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

Printed on acid-free paper

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

Foreword from the General Chairs

It was a great pleasure for us to organize the 7th International Conference on Information Systems Security Conference (ICISS) during December 15–19, 2011, at Jadavpur University, Kolkata, at the same venue where ICISS began its journey in 2005. The conference has been held every year since then at different cities in India, the last one (ICISS 2010) was successfully held at Gandhinagar, Gujrat. We are also happy that this year ICISS was held under the aegis of the newly formed Society for Research in Information Security and Privacy (SRISP), which aims to promote research and development in this arena. In the span of the last 7 years, ICISS has followed a strict reviewing policy and the acceptance ratio on average has been 25%. This year, out of 105 submissions, the Program Committee selected 20 full papers and 4 short papers for presentation.

The Program Chairs, Sushil Jajodia and Chandan Mazumdar, with the help of committed Program Committee members and reviewers did an excellent job in completing the review process well within the deadline. They were also able to arrange keynote talks by eminent researchers and practitioners in this field. We would like to record our appreciation to the Program Committee members for their painstaking effort in drawing up a high-quality technical program. We are indebted to David Evans, William Enck, Anupam Dutta and Vipul Goyal for accepting our invitation to deliver keynote talks. The Tutorial Chair, Sarmistha Neogy, had to work hard to come up with four tutorial sessions which were of great help for students and researchers to learn about topics of contemporary interest in the information security field. We would like to thank the tutoral speakers, Bjornan Solhaug, Amiya Bhattacharya, Sourav Sengupta and Rajat Subhra Chakraborty, for agreeing to share their experience.

The Organizing Committee, chaired by Mridul Sankar Barik and Sanjoy Kumar Saha, and the Finance Chair, Anirban Sengupta, worked tirelessly to ensure that the conference can be conducted without any glitch. The effort made by the Publicity Chairs, Claudio Agostino Ardagna and Anil K. Kaushik, in promoting the conference in the international forum is appreciated. We also take this opportunity to thank our sponsors and the Industry Chair, Kushal Banerjee, for their contributions.

December 2011

Arun Kumar Majumdar
Aditya Bagchi

Foreword from the Technical Program Chairs

This volume contains the papers selected for presentation at the 7th International Conference on Information Systems Security (ICISS 2011) held December 15–19, 2011 in Kolkata, India. Although ICISS was started 7 years ago as an initiative to promote information security-related research in India, from the very beginning it was decidedly an international conference attracting strong participation from researchers from all corners of the globe.

This volume contains four invited papers and 20 long and four short refereed papers that were presented at the conference. The refereed papers, which were selected from the 105 submissions, were rigorously reviewed by the Program Committee members. The resulting volume provides researchers with a broad perspective of recent developments in information systems security.

A special note of thanks goes to the many volunteers whose efforts made this conference a success. We wish to thank Anupam Datta, David Evans, Vipul Goyal, and William Enck for agreeing to deliver the invited talks, the authors for their worthy contributions, and the referees for their time and effort in reviewing the papers. We are grateful to Aditya Bagchi and Arun Majumdar for serving as the General Chairs.

Last, but certainly not least, our thanks go to the members of the Steering Committee on whom we frequently relied upon for advice throughout the year and to Jadavpur University, Kolkata, for hosting the conference.

Finally, this was the first year this conference was held under the aegis of the newly formed Society for Research in Information Security and Privacy (http://www.srisp.org.in/). This is a necessary step to ensure that information systems security research continues to expand in India and that this conference brings together the best in security research from all over the world.

December 2011

Sushil Jajodia
Chandan Mazumdar

Conference Organization

Steering Committee

Sushil Jajodia (Chair)	George Mason University, USA
Chandan Mazumdar (Convener)	Jadavpur University, Kolkata, India
Aditya Bagchi	Indian Statistical Institute, Kolkata, India
Somesh Jha	University of Wisconsin, USA
Arun Kumar Majumdar	IIT Kharagpur, India
Anish Mathuria	DA-IICT, India
Atul Prakash	University of Michigan, USA
Gulshan Rai	Department of Information Technology, Govt. of India
Sriram K. Rajamani	Microsoft Research, India
Pierangela Samarati	University of Milan, Italy
R. Sekar	SUNY, Stonybrook, USA

General Chair

A. K. Majumdar	IIT, Kharagpur, India
Aditya Bagchi	ISI, Kolkata, India

Program Chair

Sushil Jajodia	George Mason University, USA
Chandan Mazumdar	Jadavpur University, India

Organizing Chair

Mridul S. Barik	Jadavpur University, India
Sanjay Kumar Saha	Jadavpur University, India

Publicity Chair

Claudio Agostino Ardagna	University of Milan, Italy
Anil K. Kaushik	Department of Information Technology, Govt. of India

Tutorial Chair

Sarmistha Neogy	Jadavpur University, India

Finance Chair

Anirban Sengupta	Jadavpur University, India

Industry Chair

Kushal Banerjee	TCS, Kolkata, India

Program Committee

Anish Mathuria	DA-IICT, Gandhinagar, India
Atul Prakash	University of Michigan, Ann Arbor, USA
Bezawada Bruhadeshwar	IIIT, Hyderabad, India
Fabio Massacci	University of Trento, Italy
Frédéric Cuppens	ENST, France
Goutam Kumar Paul	Jadavpur University, India
Guenter Karjoth	IBM Zurich Research Laboratory, Switzerland
Indrajit Ray	Colorado State University, USA
Indrakshi Ray	Colorado State University, USA
Indranil Sengupta	IIT, Kharagpur, India
Javier Lopez	University of Malaga, Spain
Jonathan Giffin	Georgia Tech University, USA
Michiharu Kudo	IBM TRL, Japan
Mihai Christodorescu	IBM T.J. Watson Research Center, USA
Nasir Memon	Polytechnic University, USA
Patrick McDaniel	Penn State University USA
Pierangela Samarati	University of Milan, Italy
R. Ramanujam	Institute of Mathematical Sciences, India
R. Sekar	SUNY, Stony Brook, USA
S.K. Gupta	IIT, Delhi, India
Sabrina De Capitani di Vimercati	University of Milan, Italy
Samiran Chattopadhyay	Jadavpur University, India
Shamik Sural	IIT, Kharagpur, India
Shankardas Roy	Howard University, USA
Sharad Mehrotra	UC Irvine, USA
Shishir Nagaraja	IIIT Delhi, India
Shiuh-Pyng Shieh	NCTU, Taiwan
Somesh Jha	University of Wisconsin, Madison, USA
Steve Barker	King's College London, UK
Subhomoy Maitra	ISI Kolkata, India
Subrat Kumar Dash	LNMIIT, India
Sukumar Nandi	IIT, Guwahati, India

V.N. Venkatakrishnan	University of Illinois, Chicago, USA
Vijay Atluri	Rutgers University , USA
Vijay Varadharajan	Macquarie University, Australia
Yingjiu Li	SMU, Singapore
Zutao Zhu	Google Inc., USA

External Reviewers

Ardagna, Claudio
Barik, Mridul Sankar
Baskar, A.
Bayram, Sevinc
Bedi, Harkeerat Singh
Bisht, Prithvi
Chakraborty, Sandip
Chakraborty, Suchetana
Cuppens-Boulahia, Nora
Davidson, Drew
De Carli, Lorenzo
Devriese, Dominique
Doudalis, Stelios
Fredrikson, Matthew
Garcia-Alfaro, Joaquin
Gheorghe, Gabriela
Gkoulalas-Divanis, Aris
Harris, William
Hore, Bijit
Hsu, Chia-Wei
Jarecki, Stanislaw
Khadilkar, Vaibhav
Konidala, Divyan
Lai, Junzuo
Li, Bing-Han
Li, Liyun
Luchaup, Daniel
Oktay, Kerim
Paci, Federica
Pelizzi, Riccardo
Saha, Sudip
Saidane, Ayda
Sengupta, Anirban
Shi, Jie
Suresh, S. P.
Tupakula, Uday
Vhaduri, Sudip
Wang, Chia-Wei
Weinmann, Ralf-Philipp

Table of Contents

Short Papers

Understanding and Protecting Privacy: Formal Semantics and Principled Audit Mechanisms*

Anupam Datta[1], Jeremiah Blocki[1], Nicolas Christin[1], Henry DeYoung[1], Deepak Garg[2], Limin Jia[1], Dilsun Kaynar[1], and Arunesh Sinha[1]

[1] Carnegie Mellon University
[2] Max Planck Institute for Software Systems

Abstract. Privacy has become a significant concern in modern society as personal information about individuals is increasingly collected, used, and shared, often using digital technologies, by a wide range of organizations. Certain information handling practices of organizations that monitor individuals' activities on the Web, data aggregation companies that compile massive databases of personal information, cell phone companies that collect and use location data about individuals, online social networks and search engines—while enabling useful services—have aroused much indignation and protest in the name of privacy. Similarly, as healthcare organizations are embracing electronic health record systems and patient portals to enable patients, employees, and business affiliates more efficient access to personal health information, there is trepidation that the privacy of patients may not be adequately protected if information handling practices are not carefully designed and enforced.

Given this state of affairs, it is very important to arrive at a general understanding of (a) why certain information handling practices arouse moral indignation, what practices or policies are appropriate in a given setting, and (b) how to represent and enforce such policies using information processing systems. This article summarizes progress on a research program driven by goal (b). We describe a *semantic model* and *logic of privacy* that formalizes privacy as a right to *appropriate flows of personal information*—a position taken by *contextual integrity*, a philosphical theory of privacy for answering questions of the form identified in (a). The logic is designed with the goal of enabling specification and enforcement

* This work was partially supported by the U.S. Army Research Office contract "Perpetually Available and Secure Information Systems" (DAAD19-02-1-0389) to Carnegie Mellon CyLab, the NSF Science and Technology Center TRUST, the NSF CyberTrust grant "Privacy, Compliance and Information Risk in Complex Organizational Processes," the AFOSR MURI "Collaborative Policies and Assured Information Sharing," and HHS Grant no. HHS 90TR0003/01. Jeremiah Blocki and Henry DeYoung were also partially supported by NSF Graduate Fellowships. This work was mainly performed when Deepak Garg was at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

of practical privacy policies. It has been used to develop the first complete formalization of two US privacy laws—the HIPAA Privacy Rule that prescribes and proscribes flows of personal health information, and the Gramm-Leach-Bliley Act that similarly governs flows of personal financial information. Observing that preventive access control mechanisms are not sufficient to enforce such privacy policies, we develop two complementary *audit mechanisms* for policy enforcement. These mechanisms enable auditing of practical privacy policies, including the entire HIPAA Privacy Rule. The article concludes with a vision for further research in this area.

1 Introduction

Privacy has become a significant concern in modern society as personal information about individuals is increasingly collected, used, and shared, often using digital technologies, by a wide range of organizations. Certain information handling practices of organizations that monitor individuals' activities on the Web, data aggregation companies that compile massive databases of personal information, cell phone companies that collect and use location data about individuals, online social networks and search engines—while enabling useful services—have aroused much indignation and protest in the name of privacy (see, for example, a series of articles in the Wall Street Journal [51]). Similarly, as healthcare organizations are embracing electronic health record systems and patient portals to enable patients, employees, and business affiliates more efficient access to personal health information, there is trepidation that the privacy of patients may not be adequately protected if information handling practices are not carefully designed and enforced [29, 43, 49].

Given this state of affairs, it is very important to arrive at a general understanding of (a) why certain information handling practices arouse moral indignation, what practices or policies are appropriate in a given setting, and (b) how to represent and enforce such policies using information processing systems. This article summarizes progress on a research program driven by goal (b) [8, 9, 14, 22, 25]. The semantic model in this work is informed by *contextual integrity*—a philosphical theory of privacy for answering questions of the form identified in (a) [40]. Healthcare privacy has been a focus area of application for much of this work and consequently the examples in the paper are drawn from that domain. The article concludes with a vision for further research in this area.

Contextual Integrity. The central thesis of contextual integrity is that *privacy is a right to appropriate flow of personal information.* The building blocks of this theory are *social contexts* and *context-relative informational norms.* A context captures the idea that people act and transact in society not simply as individuals in an undifferentiated social world, but as individuals in certain capacities (roles) in distinctive social contexts, such as healthcare, education, friendship and employment. Norms prescribe the flow of personal information in a given context, e.g., in a healthcare context a norm might prescribe flow of personal health information from a patient to a doctor and proscribe flows from the doctor to other parties who are not involved in providing treatment. Norms are a function of the following parameters: the respective roles of the sender, the subject, and the recipient of the information, the type of information, and the principle under which the information is sent to the recipient. Examples of transmission principles include confidentiality (prohibiting agents receiving the information from sharing it with others), reciprocity (requiring bi-directional information flow, e.g., in a friendship context), consent (requiring permission from the information subject before transmission), and notice (informing the information subject that a transmission has occured). When norms are contravened, people experience a violation of privacy. This theory has been used to explain why a number of technology-based systems and practices threaten privacy by violating entrenched informational norms. In addition, it provides a prescriptive method for determining appropriate norms for a context (see [40]).

Semantic Model and Logic of Privacy. The idea that privacy expectations can be stated using context-relative informational norms is formalized in a *semantic model* and *logic of privacy* proposed by the first author and colleagues [8] and developed further in our follow-up work [22]. At a high-level, the model consists of a set of interacting agents in roles who perform actions involving personal information in a given context. For example, Alice (a patient) may send her personal health information to Bob (her doctor). Following the structure of context-relative informational norms, each transmission action is characterized by the roles of the sender, subject, receipient and the type of the information sent. Interactions among agents give rise to *traces* where each trace is an alternating sequence of states (capturing roles and knowledge of agents) and actions performed by agents that update state (e.g., an agent's knowledge may increase upon receiving a message or his role might change).

Transmission principles prescribe which traces respect privacy and which traces don't. While contextual integrity talks about transmission principles in the abstract, we require a precise logic for expressing them since our goal is to use information processing systems to check for violation of such principles. We were guided by two considerations in designing the logic: (a) *expressivity*—the logic should be able to represent practical privacy policies; and (b) *enforceability*—it should be possible to provide automated support for checking whether traces satisfy policies expressed in the logic.

A logic of privacy that meets these goals is presented in our recent work [25]. We arrive at this enforceable logic by restricting the syntax of the expressive first-order logic we used in our earlier work to develop the first complete formalization of two US privacy laws—the HIPAA Privacy Rule for healthcare organizations and the Gramm-Leach-Bliley Act for financial institutions [22][1]. These comprehensive case studies shed light on common concepts that arise in transmission principles in practice—data attributes, dynamic roles, notice and consent (formalized as temporal properties), purposes of uses and disclosures,

[1] This logic, in turn, generalizes the enforceable propositional temporal logic in [8].

and principals' beliefs—as well as how individual transmission principles are composed in privacy policies[2]. We discuss these insights in Section 2 and their formalization in the semantic model and logic in Section 3.

Audit Mechanisms for Enforcing Privacy Policies. We observe that access control mechanisms are not sufficient for enforcing all privacy policies because at run-time there may not be sufficient information to decide whether certain policy concepts (e.g., future obligations, purposes of uses and disclosures, and principals' beliefs) are satisfied or not. We therefore take the position that *audit mechanisms are essential for privacy policy enforcement.* The importance of audits has been recognized in the computer security literature. For example, Lampson [34] takes the position that audit logs that record relevant evidence during system execution can be used to detect violations of policy, establish accountability and punish the violators. More recently, Weitzner et al. [52] also recognize the importance of audit and accountability, and the inadequacy of preventive access control mechanisms as the sole basis for privacy protection in today's open information environment. However, while the principles of access control have been extensively studied, there is comparatively little work on the principles of audit. Our work is aimed at filling this gap. Specifically, we develop two complementary audit mechanisms for policy enforcement.

Our first insight is that *incomplete audit logs* provide a suitable abstraction to model situations (commonly encountered in practice) in which the log does not contain sufficient information to determine whether a policy is satisfied or violated, e.g., because of the policy concepts alluded to earlier—future obligations, purposes of uses and disclosures, and principals' beliefs. We formalize incomplete logs as partial structures that map each atomic formula to true, false or unknown. We design an algorithm, which we name reduce, to operate iteratively over such incomplete logs that evolve over time. In each iteration, reduce provably checks as much of the policy as possible over the current log and outputs a residual policy that can only be checked when the log is extended with additional information. We implement reduce and use it to check simulated audit logs for compliance with the entire HIPAA Privacy Rule. Our experimental results demonstrate that the algorithm scales to realistic audit logs. These results are summarized in Section 4 (see [25] for details).

Since privacy policies constrain flows of personal information based on subjective conditions (such as beliefs) that may not be mechanically checkable, reduce will output such conditions in the final residual policy leaving them to be checked by other means (e.g., by human auditors). The second audit algorithm, which we name Regret Minimizing Audits (RMA), learns from experience to provide operational guidance to human auditors about the coverage and frequency of auditing such subjective conditions. At a technical level, we formalize periodic audits in adversarial environments as an online learning problem over repeated games of imperfect information. The model takes pragmatic considerations into

[2] The model and logic supports information use actions in addition to transmission actions, so, strictly speaking, it can express policies that are more general than transmission principles.

account, such as the periodic nature of audits, the audit budget and the loss that an organization incurs from privacy violations. RMA is a new regret minimization algorithm for this game model. These results are summarized in Section 5 (see [14] for details).

We conclude in Section 6 with a discussion of research directions in this area, including support for policy composition and evolution, formalizing seemingly subjective conditions (such as purposes and beliefs), and remaining challenges in the design of audit mechanisms for detecting policy violations, accountability mechanisms for appropriately assigning blame when violations are detected, and incentive mechanisms to deter adversaries from committing violations.

2 Concepts in Privacy Policies

Before discussing the formal details of our semantic model, logic of privacy and enforcement mechanisms, we provide an informal overview of the basic concepts in, and the overall structure of, practical privacy policies. Both the concepts and the overall structure are derived from a thorough analysis of all privacy requirements in the U.S. laws HIPAA and GLBA, which was started in [8] and completed in [22]. These concepts are the structure of the privacy laws; abstract data attributes of a transmission; the ability of agents, which we call principals, to dynamically alter the role in which they are active and roles to which they belong; the purpose of a transmission; agents' beliefs about their environment; and temporal conditions for both past provisions and future obligations. This overview simultaneously serves to justify the features of our logic of privacy, PrivacyLFP, which we formally describe in Section 3.

2.1 Structure of Privacy Policies

Positive and Negative Norms of Transmission. In prior work, the first author and several colleagues applied the framework of contextual integrity to observe that privacy expectations inherent in laws like HIPAA and GLBA can, in general, be stated using context-relative informational norms of two kinds: *positive norms* ("may" conditions) and *negative norms* ("must" conditions) [8]. A transmission satisfies privacy expectations if any one positive norm and all negative norms applicable to the context of the transmission are satisfied. In subsequent work, several of the present authors demonstrated the entire privacy laws in HIPAA and GLBA can be formalized using this classification [22].

Practically, positive norms represent clauses of a law or policy which state that a transmission may occur *if* a condition is satisfied. For example, §164.506(c)(2) of HIPAA is a positive norm since it allows protected health information to be disclosed *if* the disclosure's purpose is treatment:

> "A covered entity may disclose protected health information for treatment activities of a health care provider."

In this way, positive norms capture the permitting clauses of a regulation. In general, out of all positive norms of a policy that apply to a disclosure, only one needs to be satisfied to deem the disclosure non-contradictory with the policy.

Negative norms represent policy clauses which state that a transmission may occur *only if* a condition is satisfied. For example, the core of HIPAA §164.508(a)(2) is a negative norm since it allows disclosure of psychotherapy notes *only if* it is authorized by the patient (modulo a few exceptions):

> "A covered entity must obtain an authorization for any use or disclosure of psychotherapy notes, except [. . .]."

Negative norms capture the denying clauses of a regulation because transmissions that do not satisfy the negative norms' conditions are disallowed. All negative norms of a policy that apply to a disclosure must be satisfied to prevent a violation of the policy when the disclosure occurs.

Exceptions to Norms of Transmission. Both positive and negative norms may contain *exceptions*. For instance, HIPAA §164.508(a)(2) presented above as a negative norm has an exception elided by [. . .] earlier: "use [of the notes] by the originator of the psychotherapy notes for treatment." Taking this clause as a canonical example of a negative norm with an exception, we see that exceptions provide a choice: a disclosure satisfies the policy if either there is evidence of the patient's authorization (thereby satisfying the norm's core), or there is evidence that the disclosure is a use by the notes' originator for treatment (thereby satisfying the norm's exception).

Similarly, positive norms can also have exceptions, though they have a different flavor. For example, §164.512(c)(1) of HIPAA allows a covered entity to disclose protected health information in reporting cases of abuse or domestic violence, but §164.512(c)(2) makes the exception that such disclosures are allowed only if the covered entity informs the victim of the report. These kind of exceptions simply refine the positive norm to a more specific set of conditions.

2.2 Common Concepts in Privacy Policies

Data Attributes. Practical privacy policies define disclosure norms over abstract attributes of data such as "protected health information" or "psychotherapy notes". These abstract attributes of data often possess a hierarchical structure which the norms must respect. For example, psychotherapy notes are a particular type of protected health information, so every norm that prohibits flows of protected health information should also deny the same flows of psychotherapy notes (unless stated otherwise).

Dynamic Roles. Just as policy norms refer to data attributes, but not to raw data, so also they usually refer to agents by their *roles*, not by their names. Thus, the legality of a particular disclosure usually depends on the role of the sender and recipient of the disclosure (e.g., a *psychiatrist* may legally disclose information to a *law enforcement official*, etc) and not their identities.

The roles held by an agent are not static; instead, they evolve over time. For example, §6803(a) of GLBA suggests that principals may become and cease to be customers of a financial institution, i.e., roles are *dynamic*:

> "At the time of establishing a customer relationship with a consumer and not less than annually during the continuation of such relationship, a financial institution shall provide a clear and conspicuous disclosure to such consumer [...], of such financial institution's policies and practices with respect to [disclosing nonpublic personal information]."

Moreover, this norm suggests that roles can be long-standing. The customer relationship is one such typically long-standing role since provisions for annual notices are required. But an agent is not active in the customer role at each moment of the several years during which he is in a customer relationship. Instead, he is variously active in the roles of parent, professor, patient, and, occasionally, customer during those years.

Past Provisions and Future Obligations. Policy norms often refer to events at different points of time; allowing an agent to opt-in or opt-out of disclosures is a common example. For example, GLBA §6802(b)(1) requires a financial institution to allow opt-out:

> "A financial institution may not disclose nonpublic personal information to a nonaffiliated third party unless— [...] the consumer is given the opportunity, before the time that such information is initially disclosed, to direct that such information not be disclosed to such third party."

In other words, this norm makes the temporal requirement that, at some past time, the consumer was given the opportunity to opt-out of the disclosure and has not since exercised that opportunity. We use the term *provision* to refer to such requirements about past events.

Temporal (time-based) relations in privacy policies are not limited to provisions about the past. For example, HIPAA §164.510(a)(2) requires that covered entities provide an opportunity to opt-out of disclosures of directory information, with an exception for cases in which it is not practicable to provide that opportunity (e.g., when a patient is in a coma). However, if this exception is used, then §164.510(a)(3)(ii) demands that:

> "The covered health care provider must inform the individual and provide an opportunity to [opt-out of] uses or disclosures for directory purposes as required by paragraph (a)(2) of this section when it becomes practicable to do so."

This imposes a requirement for an event to occur in the future, though there is no concrete time limit since one cannot predict the time at which it will become practicable to provide an opportunity to opt-out. We use the term *obligation* to refer to such requirements for future events.

2.3 Subjective Concepts

The three concepts presented in Sections 2.2 are all *objective*, i.e, information required to evaluate the concepts in the context of a disclosure may possibly be available in mechanical form. For example, the attributes of data can be inferred by analyzing the data, the role of an agent at any given time will usually be available in a roles' database and the relative precedence of two events can be determined from the time stamps of their entries in their respective event logs. However, privacy policies also often depend on concepts that are *subjective*, i.e, have no representation in mechanical form. It is due to dependence on such concepts that enforcement of practical privacy policies cannot be completely automated and requires human intervention. In the following we discuss two such concepts, *viz.*, purposes of use and disclosure and individual beliefs.

Purposes of Uses and Disclosures. Norms for use and disclosure of individual information often mention the purpose of the use or disclosure, as in §164.506(c)(2) of HIPAA:

> "A covered entity may disclose protected health information for treatment activities of a health care provider."

In general, determing whether such purpose requirements are respected may require human input[3] Like data attributes, purposes also obey a hierarchical structure, which must be reflected in PrivacyLFP. For example, the purpose of administering a blood test should be a refinement, or subpurpose, of the treatment purpose.

Agents' Beliefs. Just as a transmission's intended purpose introduces an element of subjectivity, so do agents' beliefs and professional judgment. For example, HIPAA §164.512(f)(4) states:

> "A covered entity may disclose protected health information about an individual who has died to a law enforcement official for the purpose of alerting law enforcement of the death of the individual if the covered entity has a suspicion that such death may have resulted from criminal conduct."

The covered entity's belief that the death may have resulted from criminal conduct is absolutely crucial to the norm's meaning. Without this constraint, §164.512(f)(4) would permit a covered entity to disclose the protected health information of *any* deceased person to law enforcement officials.

3 Logic of Privacy and Its Semantic Model

Having informally described the structure of and common concepts in privacy policies, we present in this section a logic, PrivacyLFP, for representing privacy

[3] See Section 6 for a pointer to ongoing work on providing semantics to purpose requirements in privacy policies.

policies. We also present the logic's semantic model. Whereas the syntax of the logic is used to represent norms of privacy policies and their relation to each other, the semantic model formalizes relevant contextual information against which the truth or falsity of such norms is checked during enforcement. Such contextual information includes, but is not limited to, use and disclosure event logs, roles' databases and data attribute information.

3.1 Overview

Technically, PrivacyLFP is first-order logic (predicate logic) with a slightly reduced syntax. The main syntactic categories in the logic are: 1) Terms, denoted t, which are symbolic representations of agents, data, attributes, roles, purposes, etc and over which variables $\boldsymbol{x}$ may range, 2) Predicates, denoted p, that represent relations between terms (e.g., Alice is a physician on 09/15/2011), and 3) Formulas, denoted φ, that are combinations of predicates using the usual connectives of logic — $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\forall \boldsymbol{x}.\varphi$ (for all instances of variables $\boldsymbol{x}$, φ), $\exists \boldsymbol{x}.\varphi$ (there is some instance of variables $\boldsymbol{x}$ such that φ), $\top$ (truth) and $\bot$ (falsity).

We represent both positive and negative norms (Section 2.1) as formulas, denoted φ^+ and φ^-, respectively. The exceptions of a norm are represented as subformulas of the formula representing the norm and are, therefore, part of the representation of the norm itself. If the positive norms applicable to a given disclosure are $\varphi_1^+, \ldots, \varphi_n^+$ whereas the negative norms applicable to the disclosure are $\varphi_1^-, \ldots, \varphi_m^-$, then the disclosure satisfies the policy if and only if the following formula is true: $(\varphi_1^+ \vee \ldots \vee \varphi_n^+) \wedge (\varphi_1^- \wedge \ldots \wedge \varphi_m^-)$. Following standard mathematics conventions, this formula is often abbreviated to $(\bigvee_i \varphi_i^+) \wedge (\bigwedge_j \varphi_j^-)$. In related work [22], several authors of this paper have formalized all norms from the HIPAA and GLBA Privacy Rules in this form. Although we do not discuss this formalization in detail here, an example representative of the formalization is shown later.

To represent the privacy policy concepts described in Sections 2.2 and 2.3, we stipulate a specific signature within PrivacyLFP that is inspired by prior work on the Logic of Privacy and Utility (LPU) [8]. Attributes of data (Section 2.2) are represented using symbolic terms (e.g., phi for <u>p</u>rotected <u>h</u>ealth <u>i</u>nformation). The hierarchy between data attributes is represented by a predicate $\texttt{attr\_in}(t_1, t_2)$, meaning that attribute t_1 is a subset of attribute t_2, e.g., $\texttt{attr\_in}(\text{medications}, \text{medical-history})$. We assume that each disclosed message is tagged (by its sender) with the attributes of information it carries and the predicate $\texttt{tagged}(m, q, t)$ means that the disclosed message m is tagged as carrying attribute t about agent q (e.g., a message may carry Alice's medical-history).

Similar to data attributes, role names (Section 2.2) are represented as symbolic terms, e.g, physician, covered-entity, etc. The relation between an agent p and its role r at time τ is represented by the formula $\texttt{inrole}(p, r, \tau)$. Including time in the relation allows accurate representation of dynamism in roles. For example, by including the time we allow for the possibility that (in our semantic model)

$\texttt{inrole}(\text{Alice}, \text{physician}, 09/15/2011)$ is true but $\texttt{inrole}(\text{Alice}, \text{physician}, 09/16/2011)$ is not (Alice is a physician on 09/15/2011, but not on 09/16/2011).

Events such as use or disclosure of personal records are also represented by predicates called *event predicates*. For example, $\texttt{send}(p_1, p_2, m, \tau)$ means that agent p_1 sends message m to agent p_2 at time τ. A salient feature of PrivacyLFP is that it *requires* that every event predicate include a time argument like τ in $\texttt{send}(p_1, p_2, m, \tau)$. This time argument can be used to compare the order of occurence of two events using a relation $\tau \leq \tau'$ between time points. For example, the normative statement "if Alice sends a message to Bob, then Bob must later send a message to Alice" can be represented as: $\forall m, \tau.\ (\texttt{send}(\text{Alice}, \text{Bob}, m, \tau) \supset (\exists m', \tau'.\ ((\tau \leq \tau') \wedge \texttt{send}(\text{Bob}, \text{Alice}, m', \tau'))))$. We use this representation of time to encode both provisions and obligations in privacy policies (Section 2.2). Although PrivacyLFP does not include any explicit temporal operators (e.g., $\Diamond$ and $\Box$ [37]), it is more expressive than two common temporal logics — linear-time temporal logic (LTL) and timed propositional temporal logic (TPTL), as shown in related work [22].

Like data attributes, purposes of use and disclosure (Section 2.3) are represented by symbolic terms (e.g., "treatment", "healthcare", etc.). The predicate $\texttt{purp}(m, u)$ means that the purpose of disclosure of message m is u. Unlike all other predicates listed above, this predicate is *uninterpreted* — in any enforcement system only a human expert may decide whether or not the predicate holds for given m and u. (Later, we explain what uninterpreted means in our semantic model.)

Similar to disclosure purposes, agents' beliefs (Section 2.3) are also represented with uninterpreted predicates. To distinguish such predicates, their names begin with the prefix `believes-`, e.g., $\texttt{believes-cause-of-death-is-crime}(p, q)$ may mean that agent p believes that agent q died due to a criminal act. Like the predicate $\texttt{purp}(m, u)$, predicates beginning with the prefix `believes-` are also uninterpreted.

Example 1. We illustrate representation of privacy policies in PrivacyLFP with the following example that is motivated by similar requirements in HIPAA, but is simpler and serves as a good illustration.

> An entity (e.g., hospital or physician's office) may send an individual's protected health information (phi) to another entity only if the receiving entity is the individual's doctor and the purpose of the transmission is treatment, or the individual has previously consented to the transmission.

Observe that this policy contains two positive norms separated by the word "or" in the above quote. Using the descriptions of predicates presented earlier, this policy can be represented in PrivacyLFP as follows:

$$\begin{aligned}\varphi_{pol} = \forall p_1, p_2, m, q, t, \tau.\ & (\texttt{send}(p_1, p_2, m, \tau) \wedge \texttt{tagged}(m, q, t)) \\ & \supset \overline{\texttt{attr\_in}}(t, \text{phi}) \\ & \vee (\texttt{inrole}(p_2, \texttt{doctorOf}(q), \tau) \wedge \texttt{purp}(m, \text{treatment}))\end{aligned}$$

$$\vee\ \exists\tau'.\ (\tau' < \tau \wedge \texttt{consents}(q, \texttt{sendaction}$$
$$(p_1, p_2, (q, t)), \tau'))$$

The horizontal line over `attr_in` indicates negation: $\overline{\texttt{attr\_in}}(t, \text{phi})$ means that attribute t is *not* a subset of attribute phi. In words, the above formula φ_{pol} means that if entity p_1 sends to entity p_2 a message m at time τ and m is tagged as carrying attribute t of individual q, then either the attribute t is not a form of protected health information (so the policy does not apply) or the recipient p_2 is a doctor of q at time τ (atom $\texttt{inrole}(p_2, \texttt{doctorOf}(q), \tau)$) and the purpose of the disclosure is treatment, or q has consented to this transmission in the past (last line of φ_{pol}).

3.2 Syntax of the Logic of Privacy

Although we have discussed the syntax of PrivacyLFP and also illustrated it in the previous section, we summarize it below. The syntax deviates slightly from first-order logic because it includes a distinct category of formulas called restrictions (denoted c) and requires that all quantifiers contain these restrictions. The inclusion of restrictions is motivated by practical requirements: In the enforcement algorithm of Section 4, a quantifier's restriction allow us to finitely compute all *relevant* instances of the quantifier, which may otherwise be an infinite set. We also omit negation for technical convenience and assume that each predicate p has a dual $\overline{p}$ that behaves exactly like the negation of p. The negation $\overline{\varphi}$ of a formula φ can then be defined using De Morgan's laws, as usual.

$$\begin{array}{lll}
\text{Terms} & t & ::= \ldots \\
\text{Atoms} & P & ::= p(t_1, \ldots, t_n) \\
\text{Formulas} & \varphi & ::= P \mid \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \forall \boldsymbol{x}.(c \supset \varphi) \mid \exists \boldsymbol{x}.(c \wedge \varphi) \\
\text{Restrictions} & c & ::= P \mid \top \mid \bot \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \mid \exists x.c
\end{array}$$

3.3 Partial Structures and Semantics

Next, we describe the mathematical structures over which we interpret PrivacyLFP and its formal semantics. What mathematical structures are appropriate for interpreting PrivacyLFP? Like structures of first-order logic, we may expect the structures of PrivacyLFP to be abstractions of information about the truth and falsity of the logic's relations. Thus, a structure could be an abstract data type encompassing all relevant event logs (to define predicates like `send`), roles' databases (to define predicates like `inrole`), and other auxiliary information about attribute hierarchies, etc.

However, unlike first-order logic where a structure maps each variable-free atom (relation) to either true or false, in PrivacyLFP we must also allow for the possibility that, in some cases, the information about the truth or falsity of a relation may be unavailable. This is primarily for three reasons. First, the logic includes subjective concepts like $\texttt{purp}(m, u)$ (message m is disclosed for purpose u), whose interpretation is unlikely to be available to any mechanized system of policy audit. We call such of lack of information *subjective incompleteness.* Second, the norms of a privacy policy may contain obligations that are to be satisfied in future; whether such obligations will hold or not cannot be determined during enforcement. We call this *future incompleteness.* Third, certain logs or databases may be unavailable at the time of policy enforcement, perhaps because they could not be integrated with the enforcement software. We call this *spatial incompleteness.*

To take into account all such potential incompleteness of information and to *force* ourselves to design enforcement mechanisms that take into account incompleteness, we interpret PrivacyLFP over *three-valued* structures that map each variable-free atom in the logic to one of three values: true (abbrev. `tt`), false (`ff`) or unknown (`uu`). Formally, a structure for PrivacyLFP (also called a *partial* structure) is a total map $\mathcal{L}$ from variable-free atoms of the logic to the set $\{\mathtt{tt}, \mathtt{ff}, \mathtt{uu}\}$.

– Subjective incompleteness may be modeled in a partial structure $\mathcal{L}$ by mapping every predicate that describes a subjective relation to `uu`. For example, we may have $\mathcal{L}(\mathtt{purp}(m, u)) = \mathtt{uu}$ for every m and u. Predicates like `purp` may also be called uninterpreted.
– Future incompleteness may be modeled in a partial structure $\mathcal{L}$ using the time argument in every event predicate. For example, we may force $\mathcal{L}(\mathtt{send}\ (p_1, p_2, m, \tau)) = \mathtt{uu}$ whenever τ exceeds the time of audit.
– Spatial incompleteness may be modeled in a partial structure by mapping each predicate that is unavailable to `uu`. For instance, if the roles database is not available, then $\mathcal{L}(\mathtt{inrole}(p, r, \tau)) = \mathtt{uu}$ for every p, r, and τ.

In Section 4, we describe an audit-based method for enforcement of privacy policies using three-valued structures for interpreting policies. The method uniformly accounts for all these forms of incompleteness.

Semantics. We formalize the semantics of logical formulas as the relation $\mathcal{L} \models \varphi$, read "$\varphi$ is true in the partial structure $\mathcal{L}$". Restrictions c are a subsyntax of formulas φ, so we do not define the relation separately for them. $\Xi[\boldsymbol{t}/\boldsymbol{x}]$ denotes substitution of terms $\boldsymbol{t}$ for variables $\boldsymbol{x}$ in the entity Ξ.

- $\mathcal{L} \models P$ iff $\mathcal{L}(P) = \mathtt{tt}$
- $\mathcal{L} \models \top$
- $\mathcal{L} \models \varphi \wedge \psi$ iff $\mathcal{L} \models \varphi$ and $\mathcal{L} \models \psi$
- $\mathcal{L} \models \varphi \vee \psi$ iff $\mathcal{L} \models \varphi$ or $\mathcal{L} \models \psi$
- $\mathcal{L} \models \forall \boldsymbol{x}.(c \supset \varphi)$ iff for all $\boldsymbol{t}$ either $\mathcal{L} \models \overline{c}[\boldsymbol{t}/\boldsymbol{x}]$ or $\mathcal{L} \models \varphi[\boldsymbol{t}/\boldsymbol{x}]$
- $\mathcal{L} \models \exists \boldsymbol{x}.(c \wedge \varphi)$ iff there exists $\boldsymbol{t}$ such that $\mathcal{L} \models c[\boldsymbol{t}/\boldsymbol{x}]$ and $\mathcal{L} \models \varphi[\boldsymbol{t}/\boldsymbol{x}]$

For dual atoms, we define $\mathcal{L}(\overline{P}) = \overline{\mathcal{L}(P)}$, where $\overline{\mathtt{tt}} = \mathtt{ff}$, $\overline{\mathtt{ff}} = \mathtt{tt}$, and $\overline{\mathtt{uu}} = \mathtt{uu}$. We say that a formula φ is *false* on the structure $\mathcal{L}$ if $\mathcal{L} \models \overline{\varphi}$. The following two properties hold:

1. Consistency: A formula φ cannot be simultaneously true and false in the structure $\mathcal{L}$, i.e., either $\mathcal{L} \not\models \varphi$ or $\mathcal{L} \not\models \overline{\varphi}$

2. Incompleteness: A formula φ may be neither true nor false in a structure $\mathcal{L}$, i.e., $\mathcal{L} \not\models \varphi$ and $\mathcal{L} \not\models \overline{\varphi}$ may both hold.

Consistency means that a policy φ cannot be simultaneously violated and satisfied at the same time. Incompleteness means that there is a policy φ and a structure $\mathcal{L}$ such that it cannot be determined whether φ has been violated in $\mathcal{L}$ or not.

Structure Extension. In practice, event logs and roles' databases evolve over time by gathering more information. This leads to a partial order, $\mathcal{L}_1 \leq \mathcal{L}_2$ on structures ($\mathcal{L}_2$ *extends* $\mathcal{L}_1$), meaning that $\mathcal{L}_2$ has more information than $\mathcal{L}_1$. Formally, $\mathcal{L}_1 \leq \mathcal{L}_2$ if for all variable-free atoms P, $\mathcal{L}_1(P) \in \{\texttt{tt}, \texttt{ff}\}$ implies $\mathcal{L}_2(P) = \mathcal{L}_1(P)$. Thus, as structures extend, the valuation of an atom may change from `uu` to either `tt` or `ff`, but cannot change once it is either `tt` or `ff`. The following property holds:

– Monotonicity: $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_1 \models \varphi$ imply $\mathcal{L}_2 \models \varphi$.

Replacing φ with $\overline{\varphi}$, we also obtain that $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_1 \models \overline{\varphi}$ imply $\mathcal{L}_2 \models \overline{\varphi}$. Hence, if $\mathcal{L}_1 \leq \mathcal{L}_2$ then $\mathcal{L}_2$ preserves both the $\mathcal{L}_1$-truth and $\mathcal{L}_1$-falsity of every formula φ.

4 Policy Audits over Incomplete Logs

In this section, we summarize an interactive algorithm for auditing system logs for privacy policy violation. To keep the presentation accessible, we present only the key ideas of our algorithm here and refer the reader to a technical paper [25] for details.

Ideally, we want our algorithm to answer the following question: Has a policy formula φ been violated in a (partial) structure $\mathcal{L}$? However, because we allow the structure to not have conclusive information about every atom, it is, in general, impossible to answer this question. Consequently, we take an *reduction-based approach*: Our algorithm implements a computable function reduce that takes as input a policy φ and a partial structure $\mathcal{L}$, and outputs a residual policy ψ that contains exactly the parts of φ that could not be verified due to lack of information in $\mathcal{L}$. Such an iteration is written $\mathsf{reduce}(\mathcal{L}, \varphi) = \psi$. If and when more information becomes available, extending $\mathcal{L}$ to $\mathcal{L}'$ ($\mathcal{L} \leq \mathcal{L}'$), another iteration of the algorithm can be used with inputs ψ and $\mathcal{L}'$ to obtain a new formula ψ'. This process can be continued until the output is either $\top$ (no violation) or $\bot$ (violation). A human auditor may augment the iterations by providing input about the truth or falsity of relevant uninterpreted atoms. By design, our algorithm satisfies three important properties:

– Termination: Each iteration terminates.
– Correctness: If $\mathsf{reduce}(\mathcal{L}, \varphi) = \psi$, then for all extensions $\mathcal{L}'$ of $\mathcal{L}$, $\mathcal{L}' \models \varphi$ iff $\mathcal{L}' \models \psi$.
– Minimality: If $\mathsf{reduce}(\mathcal{L}, \varphi) = \psi$, then an atom occurs in ψ only if it occurs in φ and its valuation on $\mathcal{L}$ is `uu`.

Technical details. The technically difficult part of reduce is its treatment of quantifiers over infinite domains. Consider, for instance, the behavior of an algorithm satisfying the above three properties on input $\forall x.\varphi$. Because the output must be minimal, in order to reduce $\forall x.\varphi$, a naive algorithm will instantiate x with each possible term and check the truth or falsity of φ for that instance on $\mathcal{L}$. This immediately leads to non-termination if the set of terms is infinite, which does happen for real policies (e.g., in Example 1, we quantify over messages m and time points τ, both of which are infinite sets).

Given the need for infinite domains, something intrinsic in quantification must limit the number of *relevant instances* of x that need to be checked to a finite number. To this end, we rely on the restrictions c in quantifiers, $\forall \boldsymbol{x}.(c \supset \varphi)$ and $\exists \boldsymbol{x}.(c \wedge \varphi)$, and use the technique of *mode analysis* from logic programming [2] to ensure that the restriction c has only a finite number of satisfying instances in any structure and that these instances are *computable.*

Briefly, mode analysis requires the policy designer to specify which argument positions of a predicate can be *computed finitely* from others. For instance, in Section 3.1 we assumed that the attributes of a message are written on it in machine-readable format and, hence, can be computed from the message. Denoting required inputs by $+$ and computable outputs by $-$, we may give the predicate $\texttt{tagged}(m, q, t)$ the *mode* $\texttt{purp}(+, -, -)$, meaning that from the input m, the outputs q, t can be computed. The mode $\texttt{purp}(-, +, +)$ is incorrect because given a fixed second and third arguments (attribute), there may be an infinite number of first arguments (messages) annotated with that attribute, so the latter set cannot be finitely computed. Similarly, if the predicate $\texttt{mult}(x, y, z)$ means that $x = yz$, where x, y, z are integers, then any of the modes $\texttt{mult}(+, +, -)$, $\texttt{mult}(-, +, +)$, and $\texttt{mult}(+, -, +)$ are okay, but $\texttt{mult}(-, -, +)$ is not. Given the mode information of all predicates in a policy, a static, linear-time check of the policy, called a *mode check*, ensures that there are only a finite number of instances of free variables that can satisfy a restriction c in the policy.

To actually *compute* the satisfying instances of a restriction, we define a function $\widehat{\texttt{sat}}(\mathcal{L}, c)$ that returns all substitutions σ for free variables of c such that $\mathcal{L} \models c\sigma$. This definition assumes a function $\texttt{sat}(\mathcal{L}, P)$ that returns all substitutions σ for free variables of P such that $\mathcal{L} \models P\sigma$ if all input positions in P are ground, which itself is implemented by looking up event logs or other databases, depending on the predicate in P.

Finally, the main audit function $\mathsf{reduce}(\mathcal{L}, \varphi)$ is defined by induction on φ, using $\widehat{\texttt{sat}}(\mathcal{L}, c)$ as a helper function when φ contains a top-level quantifier. The problematic case of the quantifiers is now easily dealt with: To reduce $\forall \boldsymbol{x}.(c \supset \varphi)$, we first invoke $\widehat{\texttt{sat}}(\mathcal{L}, c)$ to find all substitutions for $\boldsymbol{x}$ that satisfy c. Then, we recursively reduce φ after applying each such substitution and the output of reduce is the conjunction of these reducts. The reduction of $\exists \boldsymbol{x}.(c \wedge \varphi)$ is identical except that the output is the disjunction of the recursively obtained reducts.

Formal Properties. We formally prove that the $\mathsf{reduce}(\mathcal{L}, \varphi)$ is total for policies φ that pass our mode check, it is correct and minimal in the sense mentioned at the beginning of this section and that it uses space polynomial in the size of φ and runs in time polynomial in the size of $\mathcal{L}$.

Use for Action-Guidance. Besides audit for policy violations, we expect that reduce can be used as an action-guidance tool, to inform an agent whether or not an action she is about to perform would violate the policy. To do this, reduce can be run on a *hypothetical* structure that includes all the audit information in the system and the action to be performed, and a formula that contains relevant norms from the privacy policy. The formula output by reduce would then be one of: (1) $\top$ (action will not violate the policy), (2) $\bot$ (action will violate the policy), or (3) another formula ψ which lists exactly those undischarged conditions that must hold for the policy to not be violated by the potential action; the agent may check those conditions manually before performing the action.

4.1 Related Work

Runtime Monitoring with Temporal Logic. A lot of prior work addresses the problem of *runtime monitoring* of policies expressed in Linear Temporal Logic (LTL) [5, 7, 12, 44, 46, 47] and its extensions [7, 45, 46]. Although similar in the spirit of enforcing policies, the intended deployment of our work is different: We assume that system logs are accumulated independently and given to our algorithm, whereas an integral component of runtime monitoring is accumulation of system logs on the fly. Our assumption about the availability of system logs fits practical situations like health organizations, which collect transmission, disclosure and other logs to comply with regulations such as HIPAA even if no computerized policy enforcement mechanism is in place.

Comparing only the expressiveness of the logic, our work is more advanced than all existing work on policy enforcement. First, LTL can be encoded in our logic easily [22]. Second, we allow expressive quantification in our logic, whereas prior work is either limited to propositional logic [5, 44, 47], or, when quantifiers are considered, they are severely restricted [7, 45, 46]. A recent exception to such syntactic restrictions is the work of Basin et al. [12], to which we compare in detail below. Third, no prior work considers the possibility of incompleteness in structures, which our reduce algorithm takes into account.

Recent work by Basin et al. [12] considers runtime monitoring over an expressive fragment of Metric First-order Temporal Logic. Similar to our work, Basin et al. allow quantification over infinite domains, and use a form of mode analysis (called a safe-range analysis) to ensure finiteness during enforcement. However, Basin et al.'s mode analysis is weaker than ours; in particular, it cannot relate the same variable in the input and output positions of two different conjuncts of a restriction and requires that each free variable appear in at least one predicate with a finite model. As a consequence, many practical policies (including examples from the HIPAA Privacy Rule) cannot be enforced in their framework, but can be enforced in ours (see [25] for additional details).

Formal Frameworks for Policy Audit. Cederquist et al. [18] present a proof-based system for a-posteriori audit, where policy obligations are discharged by constructing formal proofs. The leaves of proofs are established from logs, but the audit process only checks that an obligation has been satisfied somewhere in the past. Further, there is no systematic mechanism to instantiate quantifiers in proofs. However, using connectives of linear logic, the mechanism admits policies that rely on use-once permissions.

Iterative Enforcement. The idea of iteratively rewriting the policy over evolving logs has been considered previously [44, 47], but only for propositional logic where the absence of quantifiers simplifies the problem considerably. Bauer et al. [5] use a different approach for iterative enforcement: they convert an LTL formula with limited first-order quantification to a Büchi automaton and check whether the automaton accepts the input log. Further, they also use a three-valued semantic model similar to ours, but assume that logs record all information about past events (past-completeness). Three-valued structures have also been considered in work on generalized model checking [17, 27]. However, the problems addressed in that line of work are different; the objective there is to check whether there exist extensions of a given structure in which a formula is satisfied (or falsified).

Compliance Checking. Barth et al. [8] present two formal definitions of compliance of an action with a policy, called strong and weak compliance. An action is *strongly compliant* with a policy given a trace if there exists an extension of the trace that contains the action and satisfies the policy. We do not consider strong compliance in this paper. An action is *weakly compliant* with a policy in Propositional LTL (PLTL) given a trace if the trace augmented with the action satisfies the present requirements of the policy. However, a weakly compliant action might incur unsatisfiable future requirements. The technical definition is stated in terms of a standard tableau construction for PLTL [37] that syntactically separates present and future requirements. Our correctness property for reduce generalizes weak compliance to a richer class of policies and structures: PLTL can be encoded in our policy logic, the residual formula generalizes future requirements, and past-complete traces are a special case of our partial structures.

In a related paper, Barth et al. [9] present an algorithm that examines audit logs to detect policy violations and identify agents to blame for policy violations. While our audit algorithm can be used to detect violations of a much richer class of policies than the propositional logics considered by Barth et al., it does not identify agents to be blamed for violations.

Lam et al. [33] represent policy requirements of a part of the HIPAA Privacy Rule in an extension of Prolog with stratified negation, called pLogic, and use it to implement a compliance checker for a medical messaging system. The compliance checker makes decisions about legitimacy of messages entering the system based on eight attributes attached to each message (such as its sender, intended recipient, subject, type of information and purpose). The prototype tool has a usable front-end and provides a useful interface for understanding what types of disclosures and uses of personal health information are permitted and forbidden by the

HIPAA Privacy Rule. However, as recognized by the authors, the approach has certain limitations in demonstrating compliance with the HIPAA Privacy Rule. First, it does not support temporal conditions. While pLogic uses specialized predicates to capture that certain events happened in the past, it cannot represent future obligations needed to formalize many clauses in HIPAA. In contrast, our policy logic and the `reduce` algorithm handle temporal conditions, including real-time conditions. Second, reasoning in pLogic proceeds assuming that all asserted beliefs, purposes and types of information associated with messages are correct. In contrast, since `reduce` mines logs to determine truth values of atoms, it does not assume facts unless there is evidence in logs to back them up. Typically, a purpose or belief will be taken as true only if a human auditor (or some other oracle) supplies evidence to that effect. Finally, our prototype implementation was evaluated with a formalization of the entire HIPAA Privacy Rule, whereas Lam et al. formalize only §§164.502, 164.506 and 164.510.

Policy Specification and Analysis. Several variants of LTL have been used to *specify* the properties of programs, business processes and security and privacy policies [8, 11, 22, 26, 36]. The logic we use as well as the formalization of HIPAA used in our experiments are adapted from our prior work on the logic PrivacyLFP [22]. PrivacyLFP, in turn, draws inspiration from earlier work on the logic LPU [8]. However, PrivacyLFP is more expressive than LPU because it allows first-order quantification over infinite domains.

Further, several access-control models have extensions for specifying usage control and future obligations [13, 23, 28, 30, 39, 41, 42]. Some of these models assume a pre-defined notion of obligations [30, 39]. For instance, Irwin et al. [30] model obligations as tuples containing the subject of the obligation, the actions to be performed, the objects that are targets of the actions and the time frames of the obligations. Other models leave specifications for obligations abstract [13, 28, 42]. Such specific models and the ensuing policies can be encoded in our logic using quantifiers.

There also has been much work on analyzing the properties of policies represented in formal models. For instance, Ni et al. study the interaction between obligation and authorization [39], Irwin et al. have analyzed accountability problems with obligations [30], and Dougherty et al. have modeled the interaction between obligations and programs [23]. These methods are orthogonal to our objective of policy enforcement.

Finally, privacy languages such as EPAL [6] and privacyAPI [38] do not include obligations or temporal modalities as primitives, and are less expressive than our framework.

5 Periodic Audits with Imperfect Information

Since privacy policies constrain flows of personal information based on subjective conditions (such as purposes and beliefs) that may not be mechanically checkable, `reduce` will output such conditions in the final residual policy leaving them to be checked by other means (e.g., by human auditors). Recent studies have revealed that such subjective conditions are often violated in the real world in the healthcare domain; violations occur as employees access medical records of celebrities, family members, and neighbors motivated by general curiosity, financial gain, child custody lawsuits and other considerations that are not appropriate purposes for accessing patient records [29, 49]. In practice, organizations like hospitals conduct *ad hoc* audits in which the audit log, which records accesses and disclosures of personal information, is examined to determine whether personal information was appropriately handled.

In this section, we summarize an audit model and algorithm that can provide guidance to human auditors in this activity [14]. This work presents the first principled learning-theoretic foundation for audits of this form. Our first contribution is a *repeated game model* that captures the interaction between the defender (e.g., hospital auditors) and the adversary (e.g., hospital employees). The model includes a budget that constrains the number of actions that the defender can inspect thus reflecting the imperfect nature of audit-based enforcement, and a loss function that captures the economic impact of detected and missed violations on the organization. We assume that the adversary is worst-case as is standard in other areas of computer security. We also formulate a desirable property of the audit mechanism in this model based on the concept of *regret* in learning theory [16]. Our second contribution is a novel *audit mechanism* that provably minimizes regret for the defender. The mechanism learns from experience and provides operational guidance to the human auditor about which accesses to inspect and how many of the accesses to inspect. The regret bound is significantly better than prior results in the learning literature.

Mirroring the periodic nature of audits in practice, we use a repeated game model [24] that proceeds in rounds. A round represents an audit cycle and, depending on the application scenario, could be a day, a week or even a quarter.

Adversary Model. In each round, the adversary performs a set of actions (e.g., accesses patient records) of which a subset violates policy. Actions are classified into types. For example, accessing celebrity records could be a different type of action from accessing non-celebrity records. The adversary capabilities are defined by parameters that impose upper bounds on the number of actions of each type that she can perform in any round. We place no additional restrictions on the adversary's behavior. In particular, we do not assume that the adversary violates policy following a fixed probability distribution; nor do we assume that she is rational. Furthermore, we assume that the adversary knows the defender's strategy (audit mechanism) and can adapt her strategy accordingly.

Defender Model. In each round, the defender inspects a subset of actions of each type performed by the adversary. The defender has to take two competing factors into account. First, inspections incur cost. The defender has an audit budget that imposes upper bounds on how many actions of each type she can inspect. We assume that the cost of inspection increases linearly with the number of inspections. So, if the defender inspects fewer actions, she incurs lower cost. Note that, because the defender cannot know with certainty whether the actions not inspected were malicious or benign, this is a game of imperfect information [3]. Second, the defender suffers a loss in reputation for detected violations. The loss is higher for violations that are detected externally (e.g., by an Health and Human Services audit, or because information leaked as a result of the violation is publicized by the media) than those that are caught by the defender's audit mechanism, thus incentivizing the defender to inspect more actions.

In addition, the loss incurred from a detected violation depends on the type of violation. For example, inappropriate access of celebrities' patient records might cause higher loss to a hospital than inappropriate access of other patients' records. Also, to account for the evolution of public memory, we assume that violations detected in recent rounds cause greater loss than those detected in rounds farther in the past. The defender's audit mechanism has to take all these considerations into account in prescribing the number of actions of each type that should be inspected in a given round, keeping in mind that the defender is playing against the powerful strategic adversary described earlier.

Note that for adequate privacy protection, the economic and legal structure has to ensure that it is in the best interests of the organization to invest significant effort into auditing. Our abstraction of the reputation loss from policy violations that incentivizes organizations to audit can, in practice, be achieved through penalties imposed by government audits as well as through market forces, such as brand name erosion and lawsuits.

Regret Property. We formulate a desirable property for the audit mechanism by adopting the concept of regret from online learning theory. The idea is to compare the loss incurred when the real defender plays according to the strategy prescribed by the audit mechanism to the loss incurred by a hypothetical defender with perfect knowledge of the number of violations of each type in each round. The hypothetical defender is allowed to pick a fixed strategy to play in each round that prescribes how many actions of each type to inspect. The *regret* of the real defender in hindsight is the difference between the loss of the hypothetical defender and the actual loss of the real defender averaged over all rounds of game play. We require that the regret of the audit mechanism quickly converge to a small value and, in particular, that it tends to zero as the number of rounds tends to infinity.

Intuitively, this definition captures the idea that although the defender does not know in advance how to allocate her audit budget to inspect different types of accesses (e.g., celebrity record accesses vs. non-celebrity record accesses), the recommendations from the audit mechanism should have the desirable property that over time the budget allocation comes close to the optimal fixed allocation. For example, if the best strategy is to allocate 40% of the budget to inspect celebrity accesses and 60% to non-celebrity accesses, then the algorithm should quickly converge towards these values.

Audit Mechanism. We develop a new audit mechanism that provably minimizes regret for the defender. The algorithm, which we name Regret Minimizing Audits

(RMA), is efficient and can be used in practice. In each round of the game, the algorithm prescribes how many actions of each type the defender should inspect. It does so by maintaining weights for each possible defender action and picking an action with probability proportional to the weight of that action. The weights are updated based on a loss estimation function, which is computed from the observed loss in each round. Intuitively, the algorithm learns the optimal distribution over actions by increasing the weights of actions that yielded better payoff than the expected payoff of the current distribution and decreasing the weight of actions that yielded worse payoff.

Our main technical result is that the exact bound on regret for RMA is approximately $2\sqrt{2\frac{\ln N}{T}}$ where N is the number of possible defender actions and T is the number of rounds (audit cycles). This bound improves the best known bounds of $O\left(\frac{N^{1/3}\log N}{\sqrt[3]{T}}\right)$ for regret minimization over games of imperfect information. The main novelty is in the way we use a loss estimation function and characterize its properties to achieve the significantly better bounds. Specifically, RMA follows the structure of a regret minimization algorithm for perfect information games, but uses the estimated loss instead of the true loss to update the weights in each round. We define two properties of the loss estimation function—*accuracy* (capturing the idea that the expected error in loss estimation in each round is zero) and *independence* (capturing the idea that errors in loss estimation in each round are independent of the errors in other rounds)—and prove that any loss estimation function that satisfies these properties results in regret that is close to the regret from using an actual loss function. Thus, our bounds are of the same order as regret bounds for perfect information games. The better bounds are important from a practical standpoint because they imply that the algorithm converges to the optimal fixed strategy much faster.

5.1 Related Work

Zhao et al. [53] recognize that rigid access control can cause loss in productivity in certain types of organizations. They propose an access control regime that allows all access requests, but marks accesses not permitted by the policy for posthoc audit coupled with punishments for violating policy. They assume that the utility function for the organization and the employees are known and use a single shot game to analyze the optimal behavior of the players. Our approach of using a permissive access control policy coupled with audits is a similar idea. However, we consider a worst-case adversary (employee) because we believe that it is difficult to identify the exact incentives of the employee. We further recognize that the repeated nature of interaction in audits is naturally modeled as a repeated game rather than a one-shot game. Finally, we restrict the amount of audit inspections because of budgetary constraints. Thus, our game model is significantly more realistic than the model of Zhao et al. [53].

Cheng et al. [19, 20] also start from the observation that rigid access control is not desirable in many contexts. They propose a risk-based access control approach. Specifically, they allocate a risk budget to each agent, estimate the risk of allowing an access request, and permit an agent to access a resource if she can pay for the estimated risk of access from her budget. Further, they use metaheuristics such as genetic programming to dynamically change the security policy, i.e. change the risk associated with accesses dynamically. We believe that the above mechanism mitigates the problem of rigid access control in settings such as IT security risk management, but is not directly applicable for privacy protection in settings such as hospitals where denying access based on privacy risks could have negative consequences on the quality of care. Our approach to the problem is fundamentally different: we use a form of risk-based auditing instead of risk-based access control. Also, genetic programming is a metaheuristic, which is known to perform well empirically, but does not have theoretical guarantees [50]. In contrast, we provide mechanisms with provable guarantees. Indeed an interesting topic for future work is to investigate the use of learning-theoretic techniques to dynamically adjust the risk associated with accesses in a principled manner.

Regret Minimization. A regret minimization algorithm is a randomized algorithm for playing in a repeated game. Our algorithm RMA is based on the weighted majority algorithm [35] for regret minimization. The weighted majority maintains weights $w_{\boldsymbol{s}}$ for each of the N fixed actions of the defender. $w^t_{\boldsymbol{s}}$ is the weight of the expert before round t has been played. The weights determine a probability distribution over actions, $p^t_{\boldsymbol{s}}$ denotes the probability of playing $\boldsymbol{s}$ at time t. In any given round the algorithm attempts to learn the optimal distribution over actions by increasing the weights of experts that performed better than its current distribution and decreasing the weights of experts that performed worse.

Sleeping Experts. In the setting of [35] all of the actions are available all of the time. However, we are working in the sleeping experts model where actions may not be available every round due to budget constraints. Informally, in the sleeping experts setting the regret of RMA with respect to a fixed action $\boldsymbol{s}$ in hindsight is the expected decrease in our total loss had we played $\boldsymbol{s}$ in each of the $T_{\boldsymbol{s}}$ rounds when $\boldsymbol{s}$ was available.

There are variations of the weighted majority algorithm that achieve low regret in the sleeping experts setting [15, 16]. These algorithms achieve average regret bounds:

$$\forall \boldsymbol{s}, \frac{\mathbf{Regret}\left(\mathsf{Alg}, \boldsymbol{s}\right)}{T_{\boldsymbol{s}}} = O\left(\frac{\sqrt{T}\log N}{T_{\boldsymbol{s}}}\right) .$$

In fact RMA is very similar to these algorithms. However, we are interested in finding exact (not asymptotic) bounds. We also have to deal with the imperfect information in our game.

Imperfect Information. In order to update its weight after round t, the weighted majority algorithm needs to know the loss of every available defender action $\boldsymbol{s}$. Formally, the algorithm needs to know $\mathbf{L}^t(\boldsymbol{s})$ for each $\boldsymbol{s} \in \mathsf{AWAKE}^t$. However, we only observe an outcome $\mathbf{O}^t$, which allows us to compute

$$\mathbf{L}^t(\boldsymbol{s}^t) = \mathbf{R}(\mathbf{O}^t) - \boldsymbol{C} \cdot \boldsymbol{s}^t,$$

the loss for the particular action $\boldsymbol{s}^t$ played by the defender at time t. There are several existing algorithms for regret minimization in games with imperfect information [3, 4, 21, 54]. For example, [3] provides an average regret bound of

$$\forall \boldsymbol{s}, \frac{\mathbf{Regret}(\mathsf{Alg}, \boldsymbol{s})}{T} = O\left(\frac{N^{1/3}\log N}{\sqrt[3]{T}}\right) .$$

It is acceptable to have $\log N$ in the numerator, but the $N^{1/3}$ term will make the algorithm impractical in our setting. The average regret still does tend to 0 as $T \to \infty$, but the rate of convergence is much slower compared to the case when only $\log N$ is present in the numerator. Other algorithms [4, 21, 54] improve this bound slightly, but we still have the $N^{1/3}$ term in the numerator. Furthermore, [3] assumes that each action $\boldsymbol{s}$ is available in every round. There are algorithms that deal with sleeping experts in repeated games with imperfect information, but the convergence bounds get even worse.

Regret minimization techniques have previously been applied in computer security by Barth et al. [10]. However, that paper addresses a different problem. They show that reactive security is not worse than proactive security in the long run. They propose a regret minimizing algorithm (reactive security) for allocation of budget in each round so that the attacker's "return on attack" does not differ much from the case when a fixed allocation (proactive security) is chosen. Their algorithm is not suitable for our audit setting due to imperfect information and sleeping experts. In their work, the defender learns the attack path played by the adversary after each round, and by extension has perfect knowledge of the loss function for that round. By contrast, RMA must work in the imperfect information setting. Also, their model considers unknown attack paths that get discovered over time. This is a special subcase of the sleeping experts setting, where an expert is awake in every round after she wakes up. They extend the multiplicative weight update algorithm [35] to handle the special case. In our setting experts may be available in one round and unavailable in next. RMA was designed to work in this more general setting.

6 Research Directions

We describe below directions for further research in this area, including support for policy composition and evolution, formalizing seemingly subjective conditions (such as purposes and beliefs), and remaining challenges in the design of audit mechanisms for detecting policy violations, accountability mechanisms for appropriately assigning blame when violations are detected, and incentive mechanisms to deter adversaries from committing violations.

While our work so far has focused on studying a single policy in one context (e.g., HIPAA for healthcare), it would be interesting to study situations where multiple policies from possibly different contexts may be relevant to the disclosure and use of personal information. For example, in the US, transmission of personal health information is governed not only by the HIPAA Privacy Rule, but also by state privacy laws. In Europe, in addition to the EU Directive, member states have their own privacy laws. A natural set of research questions arises in this setting: How should multiple policies from possibly different contexts be composed? How should conflicts be resolved? Is it possible to develop specification and enforcement techniques that are compositional? Is it possible to deal with policies (e.g., laws) that evolve over time in an incremental manner rather than requiring a significant rewrite?

As noted earlier, privacy policies often contain obligations based on beliefs or professional judgment of agents (Section 2.3). Such obligations are subjective and, in general, cannot be verified automatically by a computer system without human input. One natural question is how to provide human input about a relevant belief to a computerized audit mechanism. One possibility, already developed in our implementation of the `reduce` algorithm of Section 4, is to simply allow a user to mark a subjective concept as either true or false. Another possibility, which we plan to investigate in future, is to use logical rules to define that a belief is justified if certain principals support certain statements. We plan to use the connective A `says` φ (principal A supports statement φ) from authorization logics to represent statements made by principals and signed certificates to evidence such statements [1]. For example, the rule $(P\ \texttt{says}\ \texttt{injured-by-crime}(P)) \supset \texttt{believes-injured-by-crime}(Q, P)$ may mean that if principal P says that she was injured by a crime then it is justified for another individual Q to believe that this is the case, and a certificate signed by Alice's private key and containing the statement `injured-by-crime`(Alice) could be evidence for the formula (Alice `says` `injured-by-crime`(Alice)). Certificates necessary to justify relevant beliefs may be collected by an audit system and used to discharge obligations about beliefs automatically.

An approach to semantics and enforcement of privacy policies that place requirements on the *purposes* for which a governed entity may use personal information is outlined in a recent article by the first author and colleagues [48]. The paper presents a semantics for purpose requirements using a formal model based on planning. Specifically, the model is used to formalize when a sequence of actions is *only for* or *not for* a purpose. This semantics serves as a basis for an algorithm for automated auditing of purpose requirements in privacy policies. The algorithm takes as input an audit log and a description of an environment model that the auditee uses in the planning process.

In addition to audit mechanisms that detect violations of policy, an important research direction is developing a rigorous foundation for *accountability* and associated mechanisms that correctly blame agents responsible for violations. While the importance of accountability has been recognized in the literature[34, 52], there has not been much technical work on accountability (see [9, 31, 32] for some exceptions).

Also, while our work on regret minimizing audits makes no assumptions about the *incentives* of adversaries, a worthwhile research direction is to design mechanisms that use partial knowledge of the incentives of adversaries to deter them from committing violations. We expect that a combination of techniques from game theory and learning theory could be leveraged to make progress on this problem.

Finally, while our work has focused on the application domain of healthcare privacy, an exploration of other domains in which these enforcement mechanisms could be used would be interesting. Specifically, it would be worthwhile to investigate whether privacy protection policies adopted by financial institutions, web services providers (e.g., Google, Microsoft, Amazon) and online social networks (e.g., Facebook) can be enforced by using and adapting the kinds of mechanisms being developed in this work.

References

[1] Abadi, M., Burrows, M., Lampson, B.W., Plotkin, G.D.: A calculus for access control in distributed systems. ACM Trans. Program. Lang. Syst. 15(4), 706–734 (1993)

[2] Apt, K.R., Marchiori, E.: Reasoning about Prolog programs: From modes through types to assertions. Formal Aspects of Computing 6(6), 743–765 (1994)

[3] Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.: The nonstochastic multi-armed bandit problem. SIAM Journal on Computing 32(1), 48–77 (2003)

[4] Awerbuch, B., Kleinberg, R.: Online linear optimization and adaptive routing. Journal of Computer and System Sciences 74(1), 97–114 (2008)

[5] Baader, F., Bauer, A., Lippmann, M.: Runtime Verification Using a Temporal Description Logic. In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS, vol. 5749, pp. 149–164. Springer, Heidelberg (2009)

[6] Backes, M., Pfitzmann, B., Schunter, M.: A Toolkit for Managing Enterprise Privacy Policies. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 162–180. Springer, Heidelberg (2003)

[7] Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-Based Runtime Verification. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 44–57. Springer, Heidelberg (2004)

[8] Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and contextual integrity: Framework and applications. In: Proceedings of the 27th IEEE Symposium on Security and Privacy, Oakland, pp. 184–198 (2006)

[9] Barth, A., Datta, A., Mitchell, J.C., Sundaram, S.: Privacy and utility in business processes. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF), pp. 279–294 (2007)

[10] Barth, A., Rubinstein, B.I.P., Sundararajan, M., Mitchell, J.C., Song, D., Bartlett, P.L.: A Learning-Based Approach to Reactive Security. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 192–206. Springer, Heidelberg (2010)

[11] Basin, D., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: Proceeding of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 23–34 (2010)

[12] Basin, D., Klaedtke, F., Müller, S.: Policy Monitoring in First-Order Temporal Logic. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 1–18. Springer, Heidelberg (2010)

[13] Bettini, C., Jajodia, S., Wang, X.S., Wijesekera, D.: Provisions and obligations in policy rule management. Journal of Network and Systems Management 11, 351–372 (2003)

[14] Blocki, J., Christin, N., Datta, A., Sinha, A.: Regret minimizing audits: A learning-theoretic basis for privacy protection. In: Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF), pp. 312–327 (2011)

[15] Blum, A., Mansour, Y.: From External to Internal Regret. In: Auer, P., Meir, R. (eds.) COLT 2005. LNCS (LNAI), vol. 3559, pp. 621–636. Springer, Heidelberg (2005)

[16] Blum, A., Mansour, Y.: Learning, regret minimization, and equilibria. Algorithmic Game Theory, 79–102 (2007)

[17] Bruns, G., Godefroid, P.: Generalized Model Checking: Reasoning About Partial State Spaces. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 168–182. Springer, Heidelberg (2000)

[18] Cederquist, J.G., Corin, R., Dekker, M.A.C., Etalle, S., den Hartog, J.I., Lenzini, G.: Audit-based compliance control. International Journal of Information Security 6(2), 133–151 (2007)

[19] Cheng, P.-C., Rohatgi, P.: IT Security as Risk Management: A Research Perspective. IBM Research Report RC24529 (April 2008)

[20] Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S.: Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. In: Proceedings of the IEEE Symposium on Security and Privacy (2007)

[21] Dani, V., Hayes, T.: Robbing the bandit: Less regret in online geometric optimization against an adaptive adversary. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete algorithm, p. 943. ACM (2006)

[22] De Young, H., Garg, D., Jia, L., Kaynar, D., Datta, A.: Experiences in the logical specification of the HIPAA and GLBA privacy laws. In: Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society (WPES) (2010), Full version: Carnegie Mellon University Technical Report CMU-CyLab-10-007

[23] Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Obligations and their Interaction with Programs. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 375–389. Springer, Heidelberg (2007)

[24] Fudenberg, D., Tirole, J.: Game theory. MIT Press (1991)

[25] Garg, D., Jia, L., Datta, A.: Policy auditing over incomplete logs: Theory, implementation and applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS (2011)

[26] Giblin, C., Liu, A.Y., Müller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models (REALM). In: Proceeding of the 18th Annual Conference on Legal Knowledge and Information Systems (JURIX), pp. 37–48 (2005)

[27] Godefroid, P., Huth, M.: Model checking vs. generalized model checking: Semantic minimizations for temporal logics. In: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 158–167 (2005)

[28] Hilty, M., Basin, D., Pretschner, A.: On Obligations. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 98–117. Springer, Heidelberg (2005)

[29] Hulme, G.: Steady Bleed: State of HealthCare Data Breaches. Information Week (September 2010), http://www.informationweek.com/blog/healthcare/229200720

[30] Irwin, K., Yu, T., Winsborough, W.H.: On the modeling and analysis of obligations. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS), pp. 134–143 (2006)

[31] Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a Theory of Accountability and Audit. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 152–167. Springer, Heidelberg (2009)

[32] Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: ACM Conference on Computer and Communications Security, pp. 526–535 (2010)

[33] Lam, P.E., Mitchell, J.C., Sundaram, S.: A Formalization of HIPAA for a Medical Messaging System. In: Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G. (eds.) TrustBus 2009. LNCS, vol. 5695, pp. 73–85. Springer, Heidelberg (2009)

[34] Lampson, B.W.: Computer security in the real world. IEEE Computer 37(6), 37–46 (2004)

[35] Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comput. 108(2), 212–261 (1994)

[36] Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal 46, 335–361 (2007)

[37] Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)

[38] May, M.J., Gunter, C.A., Lee, I.: Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In: Proceedings of the 19th IEEE Workshop on Computer Security Foundations (CSFW), pp. 85–97 (2006)

[39] Ni, Q., Bertino, E., Lobo, J.: An obligation model bridging access control policies and privacy policies. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 133–142 (2008)

[40] Nissenbaum, H.: Privacy in Context: Technology, Policy, and the Integrity of Social Life. Stanford University Press (2010)

[41] OASIS XACML Committee. Extensible access control markup language (XACML) v2.0 (2004), http://www.oasis-open.org/specs/#xacmlv2.0

[42] Park, J., Sandhu, R.: Towards usage control models: beyond traditional access control. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 57–64 (2002)

[43] Robertson, J.: New data spill shows risk of online health records. Yahoo News (August 2011), http://news.yahoo.com/data-spill-shows-risk-online-health-records-120743449.html

[44] Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Automated Software Engineering 12, 151–197 (2005)

[45] Roger, M., Goubault-Larrecq, J.: Log auditing through model-checking. In: Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSF), pp. 220–236 (2001)

[46] Sokolsky, O., Sammapun, U., Lee, I., Kim, J.: Run-time checking of dynamic properties. Electronic Notes in Theoretical Computer Science 144, 91–108 (2006)

[47] Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science 113, 145–162 (2005)

[48] Tschantz, M. C., Datta, A., Wing, J.: On the semantics of purpose requirements in privacy policies. Tech. Rep. CMU-CS-11-102, Carnegie Mellon University (2010)

[49] US Health and Human Services. HIPAA enforcement, http://www.hhs.gov/ocr/privacy/hipaa/enforcement/index.html (accessed November 19, 2010)

[50] Vose, M.D., Wright, A.H., Rowe, J.E.: Implicit Parallelism. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1505–1517. Springer, Heidelberg (2003)

[51] Wall Street Journal. What they know, http://online.wsj.com/public/page/what-they-know-digital-privacy.html (accessed on September 8, 2011)

[52] Weitzner, D.J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J.A., Sussman, G.J.: Information accountability. Commun. ACM 51(6), 82–87 (2008)

[53] Zhao, X., Johnson, M.E.: Access governance: Flexibility with escalation and audit. In: HICSS, pp. 1–13 (2010)

[54] Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. Advances in Neural Information Processing Systems 20, 1729–1736 (2008)

Efficient Secure Computation with Garbled Circuits

Yan Huang[1], Chih-hao Shen[1], David Evans[1], Jonathan Katz[2], and Abhi Shelat[1]

[1] University of Virginia, Department of Computer Science
[2] University of Maryland, Department of Computer Science
`http://www.SecureComputation.org`

Abstract. Secure two-party computation enables applications in which participants compute the output of a function that depends on their private inputs, without revealing those inputs or relying on any trusted third party. In this paper, we show the potential of building privacy-preserving applications using garbled circuits, a generic technique that until recently was believed to be too inefficient to scale to realistic problems. We present a Java-based framework that uses pipelining and circuit-level optimizations to build efficient and scalable privacy-preserving applications. Although the standard garbled circuit protocol assumes a very week, honest-but-curious adversary, techniques are available for converting such protocols to resist stronger adversaries, including fully malicious adversaries. We summarize approaches to producing malicious-resistant secure computations that reduce the costs of transforming a protocol to be secure against stronger adversaries. In addition, we summarize results on ensuring *fairness*, the property that either both parties receive the result or neither party does. Several open problems remain, but as theory and pragmatism advance, secure computation is approaching the point where it offers practical solutions for a wide variety of important problems.

1 Introduction

Data gains value when it can be used in computations with data from other sources. For example, my genetic information becomes much more valuable when I can use it in a computation to measure kinship with another individual's genetic data, contribute it to a scientific genome association study, or use it to analyze drug effectiveness by comparing it to the genomes of participants in a pharmaceutical study. All of those uses, however, seem to require exposing my private data to other parties (or the other parties being willing to provide their private data to me). This leaves individuals with a dilemma: either maintain privacy, but lose much of the potential value of their data; or give up on privacy and expose ones data to malicious uses.

Secure computation provides an attractive alternative. It enables data owners to keep their data private, while allowing it to be used in computations. Secure two-party computation allows two parties to cooperatively compute the output of a function, $f(a,b)$, without revealing any information about their private inputs, a and b respectively (other than what can be inferred from the function output). The idea of secure function evaluation was introduced by Andrew Yao in the 1980s [52, 53], but it has only recently become realistic to imagine large-scale, important problems being solved by practical secure computations. Realizing such secure computations would enable many real world applications. For example, government agencies could use it to implement biometric security checks [29] (e.g., the no-fly list) and video criminal identification using street cameras [28, 47], without compromising the privacy of innocent citizens. If secure computation protocols can be inexpensive enough to execute on mobile devices, it could also enable applications using smartphones such as proximity-based voting, common interest and contacts matching, and real-time marketing [27].

In this paper, we focus on a generic approach to secure two-party computation known as *garbled circuits* or *Yao circuits* [52]. A garbled circuit protocol allows two semi-honest parties, a circuit *generator* and a circuit *evaluator*, to compute an arbitrary function $f(a,b)$, where a and b are private inputs from each party, without leaking any information about their respective secret inputs beyond what is revealed by the function output itself. We provide background on the garbled circuit protocol in Section 2. Although garbled circuit protocols have a reputation for being inefficient and requiring excessive amounts of memory, there are ways to implement garbled circuit protocols that take advantage of pipelining and circuit-level optimizations to enable much faster execution. Section 3 describes our framework for efficient garbled circuit protocols and reports on results using it for several applications.

An important parameter of any secure computation protocol is the threat model. The weakest commonly used threat model is the *semi-honest* threat model, where both parties are assumed to follow the protocol as specified but attempt to learn additional information about the other party's private inputs from the protocol transcript. This is the easiest model in which to build scalable applications, and the model most frequently used by implemented systems [9, 26, 32, 37, 47, 51]). Although this model may be appropriate for some realistic situations in which both parties have limited ability to interfere with the execution or a vested interest in the correctness of the results, it assumes a very weak adversary so is insufficient for many important use scenarios.

The strongest model is called *malicious* adversary model, where an attacker can deviate arbitrarily from the protocol specification to pry on other parties privacy. Several techniques have been proposed for converting a protocol that is secure under the semi-honest model into a protocol that is secure against a malicious adversary, which we discuss in Section 4, along with our work on making these conversions less expensive.

A second important facet of secure computation is *fairness*, which requires the participating parties obtaining the results of the computation simultaneously. Although fairness seems impossible to achieve since one party could just abort the protocol after obtaining the result, surprisingly, it turns out that fairness is achievable for some functions, and that for other functions a relaxed definition of partial fairness may be useful. We discuss our results on ensuring *fairness* in Section 5.

2 Garbled Circuits Background

Garbled circuits were introduced by Yao [53] as a generic mechanism for secure computation. A standard garbled circuit protocol involves two parties who wish to cooperatively compute the output of a function that depends on private data from both parties without revealing that private data. One party is known as the *generator*, who produces a garbled circuit for computing the function. The other party, the *evaluator*, evaluates that circuit to produce the (encrypted) result, which can then be revealed to either or both parties.

Any binary gate, f, which has two input wires W_0, W_1 and one output wire W_2, can be realized as a garbled gate. First, generate random nonces w_i^0 and w_i^1 to represent signal 0 and signal 1 on each wire W_i. Then, generate a truth table of four entries,

$$\mathrm{Enc}_{w_0^{s_0}, w_1^{s_1}}(w_2^{f(s_0,s_1)})$$

where s_0, s_1 denote the 1-bit plain signal on wire W_0, W_1, respectively. The table entries are then randomly permuted so no information is revealed by the encrypted table. We call this encrypted and permuted truth table a *garbled table*.

Next, the garbled table and the wire labels, $w_0^{s_0}$, representing the generator's secret input, are sent to the evaluator. To obtain the appropriate wire label for her own input (without revealing the input), $w_1^{s_1}$, the evaluator and generator execute an *oblivious transfer* protocol (see Section 2.1). Thus, the evaluator can decrypt one and only one entry that corresponds exactly to their inputs. Following this construction strategy, an arbitrary number of binary gates can be assembled to accomplish general purpose computation using the output wire labels of one gate as the input labels of the next gate.

In summary, a garbled circuit protocol involves three main steps: (1) the circuit generator garbles the circuit's truth tables; (2) the circuit generator directly transfers the circuit and garbled truth tables, and obliviously transfers the appropriate input labels to the evaluator; and (3) the circuit evaluator evaluates the circuit by successively decrypting the entry of each garbled table corresponding to the available input wires to learn the output wire labels necessary to fully evaluate a single path through the circuit.

Garbled circuits can compute any function as a two-party secure computation, and can be easily extended to support multi-party computation. The protocol provides security in the *semi-honest* threat model, where each party is assumed to follow the protocol as specified but attempts to learn additional information about the other party's inputs by observing the protocol execution. In Section 4, we consider approaches to making garbled circuits protocols resilient to stronger adversaries.

Next, we provide background on the oblivious transfer protocols needed to exchange inputs at the beginning of a garbled circuit protocol. Section 2.2 describes some important improvements to the basic garbled circuit protocol that enable more efficient execution. Section 2.3 describes previous frameworks designed to make it easier to build garbled circuit protocols.

2.1 Oblivious Transfer

An oblivious transfer protocol allows a *sender* to send one of a possible set of values to a *receiver*. The receiver selects and learns only one of the values, and the sender cannot learn which value the receiver selected. For example, a 1-out-of-2 oblivious transfer protocol (denoted OT_1^2) allows the sender, who has two bits b_0 and b_1, to transfer b_σ to the receiver, where $\sigma \in \{0,1\}$ is kept secret to the receiver throughout the protocol. OT_1^2 was first proposed by Even, Goldreich, and Lempel [12]. Naor and Pinkas developed an efficient OT_1^2 protocol based on Decisional Diffie-Hellman (DDH) hardness assumption [44]. We use this technique in our implementation. Based on the random oracle assumption, Ishai et al. devised a novel technique to reduce the cost of doing m OT_1^2 transfers to k OT_1^2, where k, ($k \ll m$), serves as a configurable security parameter [30].

2.2 Improvements

Several techniques have been developed to improve garbled circuit protocols. This section describes a few of the most important enhancements.

The *point-and-permute* technique allows the circuit evaluator to identify the "right" entry in a garbled truth table to decrypt [41], saving the evaluator from decrypting more than one truth table entry. The basic idea is to assign an n-bit random string (say $p = p_1 p_2 \dots p_n$) for every garbled truth table with 2^n entries. The random string determines how the generator swaps the entries in the table (swapping every other 2^j consecutive entries if $p_j = 1$). For the i-th entry in the truth table, the generator sends an n-bit string $p' = p'_1 p'_2 \dots p'_n$ (where $p'_k = p_k \oplus b_k$ and $b_1 b_2 \dots b_n$ is the binary representation of the number i), which indexes the "right" entry for the evaluator to decrypt. Considering the full truth table, revealing p' does not leak anything about p, so the evaluator cannot learn any extra information about p.

The *free-XOR* technique [35, 36] realizes all XOR gates by just XOR-ing the input wire labels, without needing any encryption operations. Its security was originally proven in the random oracle model [36]. The idea is to select wire labels where $w_i^1 = w_i^0 \oplus R$ where R is a random nonce. This allows the XOR of two wire labels to be computed by simply XOR-ing the wire labels, without requiring any encryption or communication. Choi et al. proved that it is secure under a notion of *circular* correlation robustness (which is weaker than the random oracle assumption), but is not secure under a standard variant of the correlation robustness assumption [10].

The *Garbled Row Reduction* (GRR) technique reduces the size of a garbled table of binary gates to three entries (saving 25% of network bandwidth) for all non-free gates. This is achieved by simply assigning

$$w_{out}^0 = \mathrm{Enc}_{w_{in_1}^0, w_{in_2}^0}(0)$$

where w_{out}^0 denotes the wire label representing 0 on the output wire while w_{in_1} and w_{in_2} denote the two input wires. This eliminates the necessity to transmit the encryption of one particular entry in the table. This technique is composable with both the free-XOR and point-and-permute techniques.

2.3 Frameworks

Without appropriate programming tools, programmers would have to spend a great deal of tedious effort to build a privacy-preserving application using garbled circuits. Various programming tools have been developed to automate parts of building secure computations. The most widely used garbled circuits framework is *Fairplay* [41], developed by Malkhi et al. and extended by several subsequent papers. Fairplay is a compile-and-interpret framework that automates the production of secure two-party computation applications. The main interface Fairplay exposes to programmers is a simple Algol-like programming language called *Secure Function Description Language* (SFDL) that supports very limited primitive data types (`boolean`, sized `int` and `enumerate`), expressions (addition, subtraction, comparison, and Boolean logical operations), and statements (non-recursive functions, branches, and constant number iterative loops). SFDL programs are compiled to a monolithic digital circuit which is interpreted by the server/-client runtime environments for protocol execution.

Several subsequent frameworks have built upon Fairplay. FairplayMP [8] extended the SFDL language to describe secure multi-party computations, using a circuit-based technique for multi-party computation [5]. TASTY [26] extended Fairplay's SFDL to allow the programmer to specify where in the digital circuit to integrate some arithmetic circuits (limited to addition and constant multiplication) that are realized by additive homomorphic encryption schemes.

Although Fairplay and similar previous secure computation frameworks demonstrated that it is feasible to automate building secure computing protocols, they also led to the false impression that such protocols are unlikely to be useful in practice because of their poor efficiency and scalability. Many researchers (e.g., [32, 47]) concluded that the generic garbled circuit technique is not suitable for solving real world computational problems, and instead developed custom solutions for particular problems [25, 32, 47]. In the next section, we argue that the efficiency and scalability problems attributed to garbled circuits are not inherent in the protocol, but can be overcome by a more efficient framework design.

3 Efficient Garbled Circuits Framework

There are two main reasons why secure computations implemented using Fairplay and similar frameworks tend to be slow and unscalable. The first is that the design of Fairplay requires that the entire circuit is constructed and stored in memory before evaluation can begin. These circuits are huge since each gate requires a garbled table (three encrypted values using the GRR technique) and gates may not be reused since that would leak information. Users of Fairplay have found that the memory required to store the garbled circuit prevents implementations from scaling to large inputs (as an example, Jha et al. failed to compute the edit distance of two 200-character sequences due to memory constraints thus concluded garbled circuit alone is not suitable for large circuits [32]). We address this problem by pipelining the generation and execution of the circuit so there is no need to ever have the entire circuit in memory (Section 3.1).

The other main problem with the Fairplay approach is also its main advantage: computations are represented as high-level, Algol-like programs. The problem with starting from a high-level representation is that it prevents many important optimization opportunities. Although it may one day be possible to automate these optimizations, important optimizations are well beyond the capabilities of current tools. Our approach is to adopt a circuit-level representation that enables both higher-level and lower-level optimizations to greatly improve the efficiency of generated protocols (Section 3.2).

Section 3.3 provides details on our implementation, and Section 3.4 reports on results for several applications.

3.1 Pipelined Circuit Execution

The primary limitation of previous garbled circuit implementations is the memory required to store the entire circuit in memory. For example, Pinkas et al.'s privacy-preserving AES implementation involved 11,500 non-free gates [48], each of which requires a garbled table of encrypted wire values. For problems like Levenshtein distance the size of the circuit scales with the size of the input, so only relatively small inputs can be handled.

There is no need, however, for either the circuit generator or circuit evaluator to ever hold the entire circuit in memory. The circuit generating and evaluating processes of different gates can actually be overlapped in time. In our framework, the processing of the garbled truth tables is *pipelined* to avoid the need to store the entire circuit and to save processing time. At the beginning of the evaluation, both the generator and evaluator instantiate the circuit structure, which is known to both and fairly small since it can reuse components just like a non-garbled circuit. Note that the process of generating and evaluating the circuit does not (indeed, it cannot, because of privacy) depend on the inputs, so there is no overhead required to keep the two parties synchronized.

Our framework automates the pipelined execution, so a user only needs to construct the desired circuit. When the protocol is executed, the generator transmits garbled truth tables over the network as they are produced, in an order defined by the circuit structure. As the client receives the garbled truth tables, it associates them with the corresponding gate. The client determines which gate to evaluate next based on the available output values and tables. Gate evaluation is triggered automatically when all the necessary inputs are ready. Once a gate has been evaluated it is immediately discarded, so the number of garbled tables stored in memory is minimal. Evaluating larger circuits does not substantially increase the memory load on the generator or evaluator, only the network bandwidth needed to transmit the garbled tables.

3.2 Generating Efficient Circuits

Since pipelined execution eliminates the memory bottleneck, the cost of evaluating a garbled circuit protocol scales linearly with the number of garbled gates. One way to reduce the number of gates is to identify parts of the computation that only require private inputs from one party. These components can be computed directly by that party so do not require any garbled circuits. By designing circuits at the circuit-level rather than using a high-level language like SFDL, users of our framework can take advantage of these opportunities (for example, by computing the key schedule for AES locally and transmitting it obliviously).

For the parts of the computation that involve private data from both parties so must be done cooperatively, we exploit several opportunities enabled by our framework for minimizing the number of non-free gates in our circuits.

Circuit Library. A programmer can create circuits using a library of basic circuits (e.g., comparator, adder, muxer, min) designed to make the best use of free-XOR techniques. This serves as one solution to the more general goal of replacing expensive AND and OR gates with XOR gates, which are free. Our goals are distinguished from conventional hardware circuit design in that the latter aims to optimize circuits under a completely different set of criteria, such as total number of gates, area, and power consumption. Also, since each garbled gate can only be evaluated once, the reuse goals of hardware circuit design do not apply to garbled circuits.

Minimizing Width. In garbled circuits, every bit of computation requires very expensive operations. To improve performance, our circuits are constructed with the minimal width required for the correctness of the programs. For example, if one wants to count the number of 1's in a 900-bit number, as is encountered in a face recognition application, SFDL's simplicity encourages programmers to write code that leads to a circuit that uses 10-bit accumulators throughout the computation. However, narrower accumulators are sufficient for early stages. The Hamming distance, Levenshtein distance, and Smith-Waterman applications all take advantage of this technique. For example, this technique reduces the cost for our Levenshtein distance protocol by about 20%.

Propagating Known Values. Our framework automatically propagates known wire signals when the circuit is built. For example, given a circuit designed for Hamming distance of 1024×1024 vectors, we can immediately obtain a 900×900 Hamming distance circuit by fixing 248 of the 2048 input wires to 0. Because of the value propagation, this does not incur any significant evaluation cost. As another example, all the initial states (i.e., entries in the first row and the first column of the state matrix) in both the Levenshtein and Smith-Waterman protocols are filled with known values agreed upon by both parties. Hence, our circuit computes on known signals without any needing any encryption.

Exploiting Wire Labels. The wire labels obtained during a garbled circuit evaluation are normally treated as a worthless by-product of the evaluation, but can be used in subsequent computations. In the garbled circuit evaluator's perspective, the set of wire labels computed are meaningless numbers, conveying no semantic information until the last step. This property is bound to the rigorous definition of security for garbled circuit technique. We exploit this fact to avoid garbled execution for many binary gates, in two main ways:

– **Backtracking.** We used the wire labels in conjunction with permuted data structures to perform additional computation without leaking any information. For example, the wire exiting each comparison sub-circuit in a tree-structured circuit for determining the minimum value of a large set encodes the information about each pairwise comparison. Thus, the wire labels obtained by the evaluator can be used to very efficiently evaluate a *backtracking tree* created by the generator to obliviously retrieve profile information associated with the minimum value found. In essence, the labels are used as encryption keys that are combined to reveal profile information. We use this technique in our fingerprint matching protocol [29] to obtain the identity record associated with the closest matching fingerprint.
– **Symbolic Execution.** Recall that plain signals, occasionally or frequently, can appear in our hybrid circuit (especially when circuit executions of plain and garbled signals are combined). In addition, since wire labels are unique, we can treat them as ordinary distinct symbols. This insight allows us to do binary gate-level *symbolic execution* which is *free* compared to garbled execution. When combined in a large circuit, these gate-level simplfications can collapse many binary gates to

simple wire connections. For example, the initialization in Levenshtein and Smith-Waterman algorithms specifies that a fraction of wire ports (corresponding to the values of the first row and the first column) are bound to known signals, a fact that can be exploited to eliminate many gates. Zahur et al. describes this technique in more detail, as well as further opportunities for using higher-level symbolic execution techniques to speedup privacy-preserving applications when some data can be revealed [54].

3.3 Implementation

Our framework is designed to enable programmers to define secure computations using a high-level language while providing enough control over the circuit design to enable efficient implementation. Users of our framework write a combination of high-level (Java) code and code for constructing circuits. Users need not be cryptographic experts, but are expected to be familiar with digital circuit design. Our framework and applications are available under an open source license from `http://www.MightBeEvil.com`.

Our framework core is about 1500 lines of Java code. The utility circuits comprise an additional 700 lines for efficient implementations of common circuits (adders, muxers, comparators, etc.). The small size of the framework enables it to be reasonably verified to provide users with good confidence in the integrity of the protocol software implementation (although we have not yet attempted any formal verification of properties of the framework implementation). Since both parties involved in a privacy-preserving computation need to fully trust their software for protecting their secrets, this small code base is essential in practice.

Figure 1 depicts a UML class diagram of the core classes of our framework. Concrete circuits are constructed using their build() method. The hierarchy of circuits is organized following the *Composite* design pattern [14] with respect to the build() method. Circuits are constructed in a highly modularized way, using Wire objects to connect them all together. The relation between Wire and Circuit follows a variation of the *Observer* pattern (a kind of publish-subscribe) [14]. The main difference is that, when a wire w is *connected* to a circuit on a *port* p (represented as a position index to the inputWires array of the circuit), all the observers of the input port wire p are automatically become observers of w. Moreover, the wire-to-wire propagation is done once at circuit construction time (instead of circuit execution time), which yields about 15% speedup in our experiments.

Subclasses of the SimpleCircuit abstract class provide the functions commonly required by any binary gates such as 2-to-1 AND, OR, and XOR. The AND and OR gates are implemented using the standard garbled circuit technique, whereas the XOR gate is implemented using the free-XOR optimization [36]. Our framework automatically propagates known signals which saves the protocol run-time cost whenever any internal wires can be fixed to a known value. The binary circuits form the core of our framework. To build a new application, users only need to create new subclasses of CompositeCircuit.

To simplify building new composite circuits, the build() method of CompositeCircuit abstract class is designed with the *Factory Method* pattern [14]. The code below shows the general structure of the build() method:

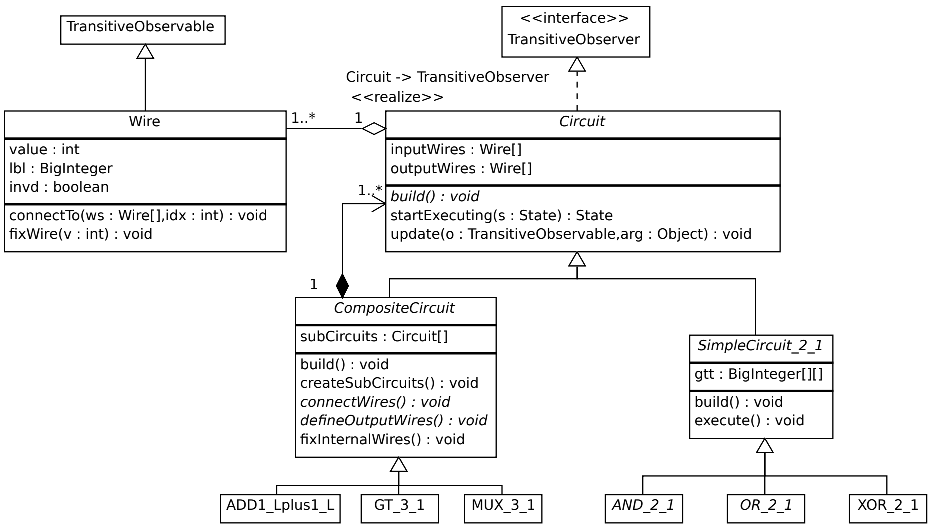


Fig. 1. Core Classes in Framework

```
public void build() throws Exception {
    createInputWires();
    createSubCircuits();
    connectWires();
    defineOutputWires();
    fixInternalWires();
}
```

To define a new circuit, a user creates a new subclass of CompositeCircuit that overrides the createSubCircuits(), connectWires(), and defineOutputWires() methods to define a new circuit. In cases where internal wires have known values (e.g., the carry-in port of an adder is fixed to 0), better performance is obtained if the user also overrides the fixInternalWires() method.

3.4 Applications

We built several target applications using our garbled circuits framework to evaluate its scalability and efficiency. Table 1 summarizes the results.

In *privacy-preserving fingerprint matching* [29], a client has a scanned candidate fingerprint and the server has a database of fingerprint images with associated profile information. The system does not reveal any information about the candidate fingerprint to the server, or about the database to the client, except the identity of the closest match if there is a match within some threshold distance (or the non-existence of any close match). We designed a bit-width minimizing circuit for finding the minimum difference of the Euclidean distances (removing the random vector added in the homomorphic encryption phase), and used a backtracking strategy to obliviously obtain the associated profile information. Our fingerprint matching protocol combines an additive homomorphic encryption phase used to compute Euclidean distances between

Table 1. Performance results for privacy-preserving applications

Application	Best Previous	Measurement	Results	Our Results	Speedup
Fingerprint Matching	Barni et al. [2]	Closest Threshold Match[a]	16s	1.5s	10.6[a]
Face Recognition	SCiFI [47]	900-bit Hamming, Online	0.310s	0.019s	16.3
		900-bit Hamming, Total	213s	0.05s	4176
Levenshtein Distance	Jha et al. [32]	100×100^b	92.4s	4.1s	22.4
		200×200^c	534s	18.4s	29.0
Smith-Waterman	Jha et al. [32]	60×60	*d*	447s	*d*
AES Encryption	Henecka et al. [26]	Online Time (per block)	0.4s	0.008s	50
		Total Time (per block)	3.3s	0.2s	16.5

All results are for 80-bit wire labels and the security parameters for the extended OT protocol [30] are $(80, 80)$. Our results are the average of 100 trials with the client and server each running on a Intel Core Duo E8400 3GHz; the comparisons are with results reported in the cited papers, using similar, but not identical, machines. *a*. Unlike our work, Barni et al. [2] cannot find the best match but instead identifies all entries within some threshold. *b*. Protocol 1, a garbled-circuit only implementation that is faster than Protocol 3, but does not scale to 200×200. *c*. Protocol 3, a hybrid protocol (the flagship protocol of [32]). *d*. No meaningful comparison is possible here, although our protocol is about twice as fast, since [32] implemented a simplified Smith-Waterman protocol [28].

fingerprint vectors, and a garbled circuit phase for finding the closest match within ε. For the other applications, we use only garbled circuit techniques.

The main operation in the *privacy-preserving face recognition* application is computing the Hamming distance between bit vectors representing face characteristics. Osadchy et al.'s results which use a combination of homomorphic encryption and 1-out-of-n oblivious transfer, but are far slower than our generic garbled circuit approach. The *Levenshtein Distance* (edit distance) and *Smith-Waterman* (genome alignment) applications use a dynamic programming algorithm. In the privacy-preserving setting, each party has one of the input strings and they wish to compute the distance between the two strings without revealing anything else about the input strings. In considering these problems, Jha et al. concluded that garbled circuits could not because of the memory blowup as the circuit size increases [32]. Our implementation is able to complete a $2000 \times 10{,}000$ Levenshtein distance problem on commodity PCs, evaluating more than 1.29 billion non-free gates in 223 minutes.

We also demonstrated generic garbled circuit based privacy-preserving applications are even possible for mobile platforms, where memory and computing resources are much more constrained [27].

4 Stronger Adversaries

The standard garbled circuits protocol, as implemented by the framework described in the previous section, is secure against semi-honest adversaries who are required to follow the protocol as specified but attempt to learn additional information about private inputs from the protocol execution. Although the semi-honest model is very weak, it is appropriate for some realistic scenarios such as when the result (including knowledge of whether or not the protocol completed successfully) is only made visible to the circuit evaluator and is not revealed to the circuit generator. In these scenarios, garbled circuit protocols can provide strong privacy (but not correctness) guarantees even against an arbitrary adversary so long as an oblivious transfer protocol that resists malicious adversaries is used. When correctness guarantees are also needed, or when confidentiality must be maintained even though the generator received the final output, the standard garbled circuit protocol is not sufficient. Instead, a protocol that tolerates malicious adversaries is required.

Section 4.1 discusses the possible attacks in the malicious model, and Section 4.2 surveys work on transforming semi-honest protocols to resist malicious adversaries. We describe our approach to constructing garbled circuit protocols that provably tolerate malicious adversaries in Section 4.3, and we compare the asymptotic complexity of our work and previous solutions in Section 4.4. Implementing protocols that can resist stronger adversaries remains much more expensive than protocols in the semi-honest model, but with improvements in both the underlying protocol implementations and approaches for strengthening semi-honest protocols, secure computations that are secure against strong adversaries are now feasible in practice for some problems.

4.1 Threats

The malicious model allows a protocol participant to deviate from the agreed protocol in arbitrary ways. Such an adversary may compromise the privacy of the other participant's data, or tamper with the correctness of the result.

A malicious generator might construct a *faulty circuit* that discloses the evaluator's private input. For example, instead of producing a circuit that computes the agreed upon function $f(a,b)$, a malicious generator could secretly construct a garbled circuit that computes $f'(a,b) \mapsto b$, thus learning the second participant's input directly.

A more subtle attack is *selective failure* [34, 42]. In this attack, a malicious generator uses inconsistent labels to construct the garbled gate and OT so that the evaluator's input can be inferred from whether or not the protocol completes. For example, a cheating generator may assign (w^0, w^1) to an input wire in the garbled circuit while using $(w^0, \hat{w}^1)$ instead in the corresponding OT where $w^1 \neq \hat{w}^1$. Consequently, if the evaluator's input is 0, she will get w^0 from OT and complete the evaluation without noticing any anomaly. In contrast, if her input is 1, she will get $\hat{w}^1$ and be unable to complete the evaluation properly. If the protocol expects the evaluator to share the result with the generator at the end, the generator learns if the evaluation failed and information about the evaluator's input is leaked.

4.2 Previous Work

Many approaches have been proposed to transform a garbled circuit protocol that provides security in the semi-honest model into a protocol that provides security guarantees in the malicious model. The two main approaches are *cut-and-choose* and *commit-and-prove*.

Cut-and-Choose. One approach is to enforce honest behavior from malicious adversaries is for the generator to prepare multiple copies of the garbled circuit with independent randomness, and the evaluator randomly chooses a fraction of the circuits, whose randomness is then revealed. The evaluator aborts if any of the chosen circuits (called *check-circuits*) are inconsistent with the revealed randomness. Otherwise, she evaluates the remaining circuits (called *evaluation-circuits*) and takes the majority of the output from evaluation-circuits as the final output.

The intuition is that in order to pass the check, a malicious generator will need to keep the number of faulty circuits low, and the minority faulty circuits will be fixed by the majority operation in the end. In other words, if a malicious generator wants to manipulate the final output, she needs to construct faulty majority among evaluation-circuits, and then the chance that none of the faulty circuits is checked will be negligible.

The cut-and-choose technique requires that the evaluator has a way to ensure that the generator provides the same input for each evaluation circuit. Recall that the generator also sends multiple copies of her input labels to the evaluator. A malicious generator may provide altered inputs to different evaluation-circuits, and it has been shown that for some functions, there are simple ways for the generator to extract information about the evaluator's input [38]. For example, suppose both parties agree to compute the inner-product of their input, that is, $f(a,b) \mapsto \sum_{i=1}^{n} a_i b_i$, where a_i and b_i is the generator's and evaluator's i-th input bit, respectively. Instead of providing $[a_1,\ldots,a_n]$ to all evaluation-circuits, the generator might send $[d_1^j,\ldots,d_n^j]$ to the j-th copy of the evaluation-circuits where $d_j^j = 1$, and $d_i^j = 0$ if $i \neq j$. The malicious generator then learns the majority bit in the evaluator's input, which is not what the evaluator agreed to reveal in advance. As a result, we must ensure *the generator's input consistency.*

Mohassel and Franklin [42] proposed the *equality-checker* scheme, which needs $O(ns^2)$ commitments to be computed and exchanged to ensure the generator's input consistency, where n is the input size and s is a statistical security parameter that is the number of copies of the garbled circuit. Lindell and Pinkas [38] develop an elegant cut-and-choose based construction that enjoys the simulation-based security against malicious players. This approach requires $O(ns^2)$ commitments to be computed and exchanged between the participants. Although these commitments can be implemented using lightweight primitives such as collision-resistant hash functions, communication complexity is still an issue. Nielsen and Orlandi [45] proposed an approach with Lego-like garbled gates. Although it is also based on the cut-and-choose method, via an alignment technique only a single copy of the generator's input keys is needed for all the evaluation-circuits. However, each gate needs several group elements as commitments resulting both computational and communicational overhead. Lindell and Pinkas propose a Diffie-Hellman pseudorandom synthesizer technique [39]. Their approach relies on finding efficient zero-knowledge proofs for specifically chosen complexity assumptions, which has complexity $O(ns)$.

In summary, to enforce honest behavior from malicious adversaries based on the cut-and-choose technique, we need to deter the faulty circuit, selective failure, and the generator's input inconsistency attacks.

Commit-and-Prove. Another well-known category to enforce honest behavior is called *commit-and-prove*. This approach is first suggested by Goldreich, Micali, and Widgerson [16], and only requires the weak general assumption of zero-knowledge proofs of knowledge. However, it has never been implemented since it requires costly NP-reductions.

Following the same idea, Jarecki and Shmatikov [31] presented an approach, in which the generator is asked to prove the correctness of the garbled circuit in zero knowledge before the evaluation starts. Although only one copy of the garbled circuit is constructed, their protocol requires hundreds of heavy cryptographic operations *per gate*, whereas approaches based on the cut-and-choose method require only such expensive operations for the *input gates*.

Recently, Nielsen et al. proposed a solution based on a variation of the commit-and-prove approach [46]. They extended the GMW [16] protocol with a technique called *authenticated bits* that have the property that only if the participants follow the agreed protocol do the results remain authenticated. In other words, if a malicious participant deviates from the agreed protocol, the other party will notice and then abort. Therefore, if the final result remains authenticated, it constitutes a proof that both parties behaved honestly. Although the GMW protocol requires many expensive OTs, Neilson et al. manage to conquer this issue with OT extensions, and thus, their solution has good amortized efficiency.

4.3 Our Approach

We tackle the selective failure attack by using a stronger notion of OT called *committing OT* [34]. This notion requires that in addition to getting exactly one message of her choice, the receiver of the OT (the evaluator of the garbled circuit protocol) also gets the commitments to *both* of the sender's messages. Later, the sender of the OT can *post-facto* prove that she ran the OT correctly by revealing the randomness used in the OT only for those OT instances corresponding to circuits that are opened for verification.

We solve the input consistency problem in an efficient manner by designing a way to use weak witness indistinguishable proofs instead of zero-knowledge protocols (or Σ-protocols). A witness-indistinguishable proof only makes the guarantee that the verifier cannot learn which witness the prover used during a proof. We design a special witness-indistinguishable proof for an operation concerning *claw-free functions* that have a weak malleability property to generate efficient instantiations of input consistency proofs. Shelat and Shen provide details on the full protocol and its security proof [50].

We note that both the committed-input scheme [42] and Diffie-Hellman pseudorandom synthesizer technique [39] are special cases of our approach, and thus, have similar complexity. However, the committed-input scheme is not known to enjoy simulation-based security, and the pseudorandom synthesizer technique requires zero-knowledge proofs that are unnecessary in this case. Our approach is faster than these works by a constant factor.

4.4 Communication Complexity

To understand the costs of hardening a garbled circuit protocol against malicious adversaries, we compare the communication efficiency between protocols that use a mix of light cryptographic primitives (such as commitments instantiated with collision-resistant hash functions) and heavy ones (such as group operations that rely on algebraic assumptions like discrete logarithm). We consider asymptotic complexity under reasonable assumptions about the growth of various primitives with respect to the security parameter k:

1. light cryptographic primitives have size $\Theta(k)$;
2. heavy cryptographic operations, like elliptic curve operations, have size $\tilde{o}(k^2)$; and
3. heavy cryptographic operations, like RSA or group operations over $\mathbb{Z}$, have size $\tilde{o}(k^3)$.

We make the assumption since in certain elliptic curve groups, known methods for computing discrete logarithms of size n run in time $L_n(1, \frac{1}{2})$. Thus, to achieve security of 2^k, it suffices to use operands of size $\tilde{o}(k^2)$, by which we mean a value that is asymptotically smaller than k^2 by factors of $\log(k)$.

Table 2 summarizes our asymptotic analysis. Let k be a security parameter and s be a statistical security parameter, and let $|C|$ be the number of gates in the base circuit. The other protocols are:

– Jarecki and Shmatikov [31]: This is a commit-and-prove approach, so it does not need to ensure input consistency. However, it requires hundreds of group operations per gate in order to defend faulty circuit and selective failure attacks (with ZK proofs). Since this protocol assumes the decisional composite residuosity problem in an RSA group, each group element is of size $\tilde{o}(k^3)$.
– Kiraz [33]: This approach is based on the cut-and-choose technique. So, it uses s copies of the garbled circuit, each circuit has $|C|$ gates, and each gate needs $O(k)$ for garbled truth table. Also, they use an equality-checker framework that requires $O(ns^2)$ commitments to enforce the generator's input consistency. As with our approach, they thwart the selective failure attack by using committing OTs.
– Lindell and Pinkas [38]: This is also a cut-and-choose-based approach. Each of the generator's input bits requires $O(s^2)$ light commitment for the consistency check. To defend against the selective failure attack, it requires $\max(4n, 8s)$ OT's.

Table 2. Analysis of two-party secure computation against malicious adversaries

	Communication		
	Base Circuit	Generator's Input	Evaluator's Input
JS07 [31]	$\|C\| \cdot \tilde{o}(k^3)$	–	n (committed) OT's
Ki08 [33]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns^2k)$	n (committing) OT's
LP07 [38]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns^2k)$	$\max(4n, 8s)$ OT's
LP10 [39]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	n OT's
Our work [50]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	n (committing) OT's

– Lindell and Pinkas [39]: This approach is similar to ours in finding a good balance between approaches using many but lightweight primitives and approaches using a few expensive group operations. This uses a technique called cut-and-choose OT to handle the generator's input inconsistency and selective failure attacks. However, they rely on more specific cryptographic assumptions, and the unnecessary zero-knowledge proofs incur constant factor overhead.

5 Fairness

Fairness is the property that either *all parties* receive the output, or *no one* does.[1] None of the protocols we have described so far provide any fairness properties. Fairness is desirable in many circumstances:

– *Coin-tossing* protocols can be used to generate an unbiased coin. This can be viewed as secure computation of a probabilistic functionality taking no inputs. If fairness is not guaranteed, then one party might learn the value of the coin first and then decide whether to abort the protocol based on the coin's value.
– In a protocol for *exchanging digital goods* (such as using digital currency to purchase a song) it would be unacceptable for the buyer to obtain the digital good without also making the payment (or for the seller to receive the payment without providing the digital good). Similarly, in *exchange of digital signatures* it is considered undesirable for one party to obtain the second party's signature on a contract without the second party simultaneously getting a copy of the first party's signature on the same contract.

More generally, we can imagine any scenario where learning the output — while preventing the other party from learning the output — provides a competitive advantage. Without a guarantee of fairness, parties in such a situation may be unwilling to even participate in the protocol.

In fact, fairness *can* be ensured in the multi-party setting in the case when a majority of the parties are guaranteed to remain honest [3, 16, 49]. (This assumes a broadcast channel, or a mechanism such as a PKI that allows broadcast to be implemented.) In case no honest majority can be assumed, it may seem obvious that fairness cannot be ensured by the following argument (specialized to the setting of two-party computation): as the parties alternate sending messages of the protocol, surely one party must learn their output first. If that party aborts immediately upon learning its output, then the other party clearly does not learn its output.

Cleve [11] formalized this intuition, and showed that fair coin tossing is impossible in both the two-party and multi-party settings (when an honest majority is not assumed). This implies an analogous impossibility result for fair computation of the 1-bit XOR function (since fair computation of XOR would immediately imply fair coin tossing).

Thus, secure computation without an honest majority (for malicious adversaries) is typically defined relative to an ideal world in which fairness is not ensured at all.

[1] We assume for simplicity in our discussion that all parties are *supposed* to receive the same output, but everything we say generalizes to the case where different parties are supposed to receive different outputs.

Specifically, the ideal world is taken to be one in which the parties send their inputs to a trusted entity who computes the result *and sends it back to the adversary only*; the adversary then sends either abort or continue to the trusted party. If she sends abort, the honest parties receive nothing from the trusted party, while in the second case the trusted party sends the honest parties the correct result.

Faced with Cleve's impossibility result, researchers have explored several ways to obtain some form of fairness:

1. Cleve's impossibility result rules out fairness for one specific function, but does not rule out fairness for *every* function. Might there be any non-trivial functions for which fairness is possible? For over twenty years after Cleve's result the answer was assumed to be "no," especially given how convincing the informal argument against fairness seemed to be. It therefore came as somewhat of a surprise when it was recently shown that there *do* exist non-trivial functions for which fairness is possible. Section 5.1 surveys this work.
2. The standard definition of secure computation without an honest majority gives up on fairness entirely. Instead, it seems preferable to define some notion of *partial fairness* and design protocols achieving at least that. Section 5.2 discusses several different notions of partial fairness.
3. Cleve's impossibility result holds in the usual cryptographic model where the adversary may behave in an arbitrarily malicious way. In some settings, however, it may be reasonable to assume a *rational* adversary whose cheating is motivated by some explicit utility function that the adversary is trying to maximize. Section 5.3 describes work on protocols that are designed to provide fairness against a rational, but not malicious, adversary.

5.1 Complete Fairness for Specific Functions

Cleve showed one function for which fair secure two-party computation is impossible without an honest majority, but did not show that fairness is impossible for *all* functions. Indeed, functions that depend on only one of the parties' inputs can be computed with complete fairness. This class includes some interesting functionalities — *zero-knowledge* among them — but still seems to miss the main difficulty of fairness in the first place. Thus, the question becomes whether there are any functions that can be computed with complete fairness that depend on more than one party's inputs. The answer, surprisingly, turns out to be *yes*. This has been shown in both the two-party [19] and multi-party [21] settings.

Instead of presenting the technical details of the protocols here (see [19, 21]), we explain informally how the two types of protocols shown by Gordon et al. [19] in the two-party setting manage to circumvent the convincing intuition that "one party must learn its output first, and can abort immediately after doing so to prevent the other party from learning its output". In the first type of protocol presented in [19], *the round in which a party learns its output depends on that party's input*. (This is in contrast to standard protocols for secure computation where the output is always learned, by both parties, in some fixed round.) In particular, simplifying slightly, if we number the parties' possible inputs $x_1, \ldots, x_\ell$ then a party holding input x_i learns the output in round i. Thus, depending on the parties' respective inputs, either party might learn the output first. Moreover, on an intuitive level, an abort by a party (say, P_1) in round i can be viewed as a "signal" to the other party P_2 that P_1's input was in fact x_i; thus, even after an abort by P_1, party P_2 can still compute the function using x_i as P_1's input. That this works is not immediate: for one thing, a malicious P_1 can try to "fool" P_2 by aborting in round $i+1$, say. Nevertheless, it can be shown that this protocol does achieve complete fairness for certain carefully constructed functions.

In the second type of protocol, parties also do not learn their outputs in some fixed round. Instead, the round in which the output is revealed is chosen according to a geometric distribution; moreover, the parties do not learn definitively in which round the output is revealed until the end of the protocol. (Slightly more formally, by the end of the protocol they are guaranteed that, with all but negligible probability, they hold the correct output.) Probabilities are balanced in such a way that even if one party aborts early, and thus "knows more information" about the correct output than the other party does, it doesn't *know* how to use this extra knowledge to violate fairness. (The formal proof of security shows that an aborting adversary can "learn the same information" in an ideal world where early abort is not possible, possibly by changing its input.)

What is especially interesting about both protocols described above, is that the proofs of fairness in each case boil down to information-theoretic arguments that do not rely on cryptography. It is thus natural to conjecture that development of the right information-theoretic tools will enable better analysis of fairness, and might help to resolve the main open question that remains: to characterize those functions for which complete fairness is possible.

5.2 Partial Fairness

Even given the work described in the previous section, we know that for certain functions complete fairness is simply not possible. The standard approach is to give up on fairness altogether. An better alternative is to instead define some notion of *partial* fairness and attempt to design protocols that achieve that weaker notion.

There are at least two general approaches to partial fairness that date back to the early 1980s (see [18, 22] for more detailed discussion). In one approach, a protocol is constructed such that at every round both parties can recover their output using a "similar" amount of work [12, 13, 15]. An unsatisfying feature of this approach is that the decision of whether an honest party should invest the necessary work to recover the output is *not* specified as part of the protocol but is instead decided "externally". Such protocols raise the risk of denial-of-service attacks by an adversary who aborts the protocol early (if that would cause the honest party to then invest a significant amount of work to recover the answer). Protocols of this sort also seem to require very strong cryptographic assumptions.

A second approach [4, 17] can be viewed as designing a protocol in which both parties gradually increase their "confidence" in the output (for example, by learning an independent noisy version of the output in each round). It seems difficult to extend such protocols to multi-bit outputs, or the case where parties are supposed to learn different outputs. More problematic is that such protocols allow the adversary to significantly bias the output of the honest party, thus violating correctness.

More recently, Gordon and Katz suggested another definitional approach that has the advantage of remaining within the simulation-based framework of standard security definitions [22]. The idea is to define partial fairness using the *same* ideal-world model used to define complete fairness. However, rather than require that the real world and ideal world be completely (computationally) indistinguishable, partial fairness is instead defined by allowing the real and ideal worlds to be distinguishable by at most $1/p$, for an arbitrary polynomial p. Such a protocol is called $1/p$*-secure*. The way to think about this is that a $1/p$-secure protocol is secure up to a (possible) $1/p$ "defect". In particular, fairness is guaranteed to hold except with probability (at most) $1/p$.

Moran et al. showed how to construct a protocol for $1/p$-secure coin tossing [43]. Gordon and Katz [22] showed how to construct $1/p$-secure protocols in the two-party setting for any function with polynomial-size domain or range. They also proved a general impossibility result for functions without one of these requirements. Both of these works have since been extended to the multi-party setting [6, 7].

5.3 Fairness with Rational Parties

Another approach for dealing with fairness is to explicitly model the adversary as *rational*, rather than arbitrary malicious as in most work in cryptography. In the context of fairness, it is most natural to consider an adversary with the following utilities (specialized to the two-party case):

– The adversary prefers to learn the (correct) output of the function above all else.
– Assuming it learns the output of the function, the adversary prefers that the other party does *not* learn the output.

Protocols that remain fair in the presence of adversaries with the above utilities were first considered in the context of *rational secret sharing* [20, 24, 40]. More recently, the model has been extended to fair secure two-party computation of general functionalities. Asharov et al. [1] propose several equivalent definitions of the problem and show a negative result, giving a function that cannot be computed fairly even if a rational adversary is assumed. Subsequently, Groce and Katz [23] showed broad *positive* results for this setting along with a partial characterization of when rational fair computation is possible.

6 Conclusion

General secure computation techniques offer the promise of strong privacy guarantees without the the need for a trusted third party. Until recently, however, such techniques were largely viewed as a theoretical curiosity because of the high cost of implementing them for real applications. Recent advances in both the theory and implementation of generic garbled circuit protocols, however, make large-scale privacy-preserving applications a realistic possibility. Many challenges remain, especially to provide efficient solutions against stronger adversaries and to provide fairness guarantees when needed, but the promise secure computation offers to perform computation with private data without compromising that data appears to be in reach.

Acknowledgments. The work described in this paper was partly supported by grants from the National Science Foundation, DARPA, and a MURI award from the Air Force Office of Scientific Research. The contents of this paper do not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred. The authors thank Peter Chapman, Jiamin Chen, Yikan Chen, Michael Hicks, Sang Koo, Benjamin Kreuter, Aaron Mackey, Steven Myers, Mona Sergi, and Samee Zahur for their contributions to this project.

References

1. Asharov, G., Canetti, R., Hazay, C.: Towards a Game Theoretic View of Secure Computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 426–445. Springer, Heidelberg (2011)
2. Barni, M., Bianchi, T., Catalano, D., Raimondo, M.D., Labati, R.D., Faillia, P., Fiore, D., Lazzeretti, R., Piuri, V., Scotti, F., Piva, A.: Privacy-preserving Fingercode Authentication. In: ACM Multimedia and Security Workshop (2010)
3. Beaver, D.: Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. Journal of Cryptology (1991)
4. Beaver, D., Goldwasser, S.: Multiparty Computation with Faulty Majority. In: 30th Symposium on Foundations of Computer Science (1989)
5. Beaver, D., Micali, S., Rogaway, P.: The Round Complexity of Secure Protocols. In: ACM Symposium on Theory of Computing (1990)
6. Beimel, A., Lindell, Y., Omri, E., Orlov, I.: 1/*p*-secure Multiparty Computation Without Honest Majority and the Best of Both Worlds. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 277–296. Springer, Heidelberg (2011)
7. Beimel, A., Omri, E., Orlov, I.: Protocols for Multiparty Coin Toss with Dishonest Majority. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 538–557. Springer, Heidelberg (2010)
8. Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: A System for Secure Multi-party Computation. In: ACM Conference on Computer and Communications Security (2008)
9. Brickell, J., Shmatikov, V.: Privacy-Preserving Graph Algorithms in the Semi-Honest Model. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 236–252. Springer, Heidelberg (2005)
10. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.S.: On the Security of the "Free-XOR" Technique (2011), http://eprint.iacr.org/2011/510
11. Cleve, R.: Limits on the Security of Coin Flips when Half the Processors Are Faulty. In: 18th Symposium on Theory of Computing (1986)
12. Even, S., Goldreich, O., Lempel, A.: A Randomized Protocol for Signing Contracts. Communications of the ACM (1985)
13. Galil, Z., Haber, S., Yung, M.: Cryptographic Computation: Secure Fault Tolerant Protocols and the Public-Key Model. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 135–155. Springer, Heidelberg (1988)
14. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns — Elements of Reusable Object-Oriented Software. Addison-Wesley (March 1995)
15. Garay, J.A., MacKenzie, P.D., Prabhakaran, M., Yang, K.: Resource Fairness and Composability of Cryptographic Protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 404–428. Springer, Heidelberg (2006)
16. Goldreich, O., Micali, S., Wigderson, A.: How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority. In: 19th Symposium on Theory of Computing (1987)

17. Goldwasser, S., Levin, L.A.: Fair Computation of General Functions in Presence of Immoral Majority. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 77–93. Springer, Heidelberg (1991)
18. Gordon, S.D.: Fairness in Secure Computation. Ph.D. thesis, University of Maryland (2010)
19. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete Fairness in Secure Two-Party Computation. In: 40th Symposium on Theory of Computing (2008)
20. Gordon, S.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)
21. Gordon, S.D., Katz, J.: Complete Fairness in Multi-Party Computation Without an Honest Majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 19–35. Springer, Heidelberg (2009)
22. Gordon, S.D., Katz, J.: Partial Fairness in Secure Two-Party Computation. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 157–176. Springer, Heidelberg (2010)
23. Groce, A., Katz, J.: Fair Computation with Rational Players (2011), http://eprint.iacr.org/2011/396
24. Halpern, J., Teague, V.: Rational Secret Sharing and Multiparty Computation. In: 36th Symposium on Theory of Computing (2004)
25. Hazay, C., Lindell, Y.: Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
26. Henecka, W., Kogl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-partY computations. In: ACM Conference on Computer and Communications Security (2010)
27. Huang, Y., Chapman, P., Evans, D.: Privacy-Preserving Applications on Smartphones. In: USENIX Workshop on Hot Topics in Security (2011)
28. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster Secure Two-Party Computation Using Garbled Circuits. In: USENIX Security Symposium (2011)
29. Huang, Y., Malka, L., Evans, D., Katz, J.: Efficient Privacy-Preserving Biometric Identification. In: Network and Distributed System Security Symposium (2011)
30. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
31. Jarecki, S., Shmatikov, V.: Efficient Two-Party Secure Computation on Committed Inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)
32. Jha, S., Kruger, L., Shmatikov, V.: Towards Practical Privacy for Genomic Computation. In: IEEE Symposium on Security and Privacy (2008)
33. Kiraz, M.: Secure and Fair Two-Party Computation. Ph.D. thesis, Technische Universiteit Eindhoven (2008)
34. Kiraz, M., Schoenmakers, B.: A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction. In: 27th Symposium on Information Theory in the Benelux (2006)
35. Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009)
36. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
37. Lindell, Y., Pinkas, B.: Privacy Preserving Data Mining. Journal of Cryptology 15(3) (2002)
38. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)

39. Lindell, Y., Pinkas, B.: Secure Two-Party Computation Via Cut-and-Choose Oblivious Transfer. Crypto ePrint Archive (2010), http://eprint.iacr.org/2010/284
40. Lysyanskaya, A., Triandopoulos, N.: Rationality and Adversarial Behavior in Multi-Party Computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 180–197. Springer, Heidelberg (2006)
41. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — A Secure Two-Party Computation System. In: USENIX Security Symposium (2004)
42. Mohassel, P., Franklin, M.: Efficiency Tradeoffs for Malicious Two-Party Computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
43. Moran, T., Naor, M., Segev, G.: An Optimally Fair Coin Toss. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 1–18. Springer, Heidelberg (2009)
44. Naor, M., Pinkas, B.: Efficient Oblivious Transfer Protocols. In: ACM-SIAM Symposium on Discrete Algorithms (2001)
45. Nielsen, J.B., Orlandi, C.: LEGO for Two-Party Secure Computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009)
46. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A New Approach to Practical Active-Secure Two-Party Computation. Crypto ePrint Archive (2011), http://eprint.iacr.org/2011/091
47. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: SCiFI: A System for Secure Face Identification. In: IEEE Symposium on Security and Privacy (2010)
48. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
49. Rabin, T., Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In: 21st Symposium on Theory of Computing (1989)
50. Shelat, A., Shen, C.-H.: Two-Output Secure Computation with Malicious Adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011)
51. Yang, Z., Zhong, S., Wright, R.: Privacy-preserving Classification of Customer Data without Loss of Accuracy. In: SIAM International Conference on Data Mining (2005)
52. Yao, A.C.: Protocols for Secure Computations. In: Symposium on Foundations of Computer Science (1982)
53. Yao, A.C.: How to Generate and Exchange Secrets. In: Symposium on Foundations of Computer Science (1986)
54. Zahur, S., Huang, Y., Evans, D.: Efficient Secure Computation over Partially-Secret Inputs (2011), http://www.mightbeevil.com

Defending Users against Smartphone Apps: Techniques and Future Directions

William Enck

North Carolina State University
enck@cs.ncsu.edu

Abstract. Smartphone security research has become very popular in response to the rapid, worldwide adoption of new platforms such as Android and iOS. Smartphones are characterized by their ability to run third-party applications, and Android and iOS take this concept to the extreme, offering hundreds of thousands of "apps" through application markets. In response, smartphone security research has focused on protecting users from apps. In this paper, we discuss the current state of smartphone research, including efforts in designing new OS protection mechanisms, as well as performing security analysis of real apps. We offer insight into what works, what has clear limitations, and promising directions for future research.

Keywords: Smartphone security.

1 Introduction

Smartphones are a widely popular and growing space of computing technology. In Q2 2011, over 107 million smartphones were sold worldwide, accounting for 25% of mobile devices [34]. Smartphones provide an ultra-portable interface to the Internet and the computational abilities to make it meaningful. Using environment sensors such as GPS, cameras, and accelerometers, they enhance everyday tasks with the wealth of information available from the Internet.

Fundamental to smartphones are applications, colloquially known as "apps." There were not many apps available for early smartphones, and hence adoption was slow. In 2008, a perfect storm emerged: 3G connectivity finally became widespread, handset technology provided "large" touch-screens and useful sensors such as GPS and accelerometers, and the first application market, Apple's App Store, was created. While all of these factors were crucial, the application market played potentially the most important role. There is a strong correlation, if not causation, between the number of applications in Apple's App Store and Google's Android Market and the rising dominance of iOS and Android.

Warnings of smartphone malware were early and succinct. In 2004, long before smartphones gained widespread popularity, Dagon et al. [19] and Guo et al. [37] discussed the dangers of enhancing cellular phones with network and computational power. These dangers derive from the very concept of a "smart" phone. Users have come to trust their cellular phones, carrying them day and night, and using them for personal and intimate conversations. Increasing code functionality and diversifying its origin results in misplaced trust. It enables eavesdropping and privacy violations. As we place more information and reliance on smartphones, they become targets for information and identify theft, as well as denial of service attacks (e.g., battery exhaustion). Furthermore, their connection to telecommunications networks opens potential for emergency call center DDoS, voice spam, and other attacks on the network.

The initial smartphone security threats still exist, but smartphone malware surveys [56,30] have reported trends that help focus attention. Smartphone malware is comprised primarily of Trojans, often designed to exfiltrate user information or use premium rate cellular services (e.g., SMS). That is, smartphone malware targets the user. Hence, this paper discusses available and proposed defenses for the user against apps they choose to install. We focus on both *malware* and *grayware* (i.e., dangerous functionality without provable malicious intent).

Current smartphone platforms have two promising characteristics not yet common on PCs. First, protection policies isolated or sandbox applications by default. Second, applications are frequently distributed via application markets, providing centralized software management. To date, security certification has only played a small role [43]; however, so called "kill switches" have proved to be a valuable means of cleaning up affected devices. Regardless of current implementations, opportunities exist to enhance the security of future markets.

In this paper, we survey research proposals for enhancing smartphone security. We classify existing research into two categories: *protection systems*, and *application analysis*. We overview proposals to enhance the existing smartphone protection systems, discussing their benefits and limitations. We then consider techniques for application analysis and their potential use in market-based security certification. In both areas, our goal is to highlight promising techniques and help direct future research.

Much of this survey focuses on the Android platform, which has been the platform of choice for researchers. This likely results because: 1) Android is open source and widely popular, allowing researchers to build prototypes to validate their ideas for real applications; and 2) Android is the only platform that allows (and encourages) flexible communication between applications, which introduces interesting security problems for study.

We begin our survey by discussing protections already in place by current smartphone platforms. Next, we introduce and contrast proposals to enhance these models. We then discuss approaches for analyzing applications to identify dangerous behavior. We conclude by highlighting promising research directions.

2 Background

Shown in Figure 1, smartphones retrieve apps from application markets and run them within a middleware environment. Existing smartphone platforms rely on application markets and platform protection mechanisms for security. We now overview protections currently implemented in popular platforms.

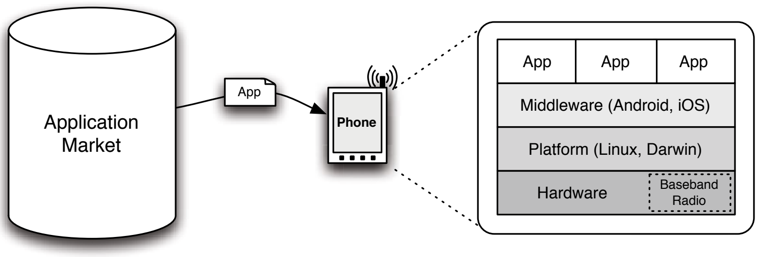


Fig. 1. Smartphone architecture

2.1 Application Markets

Finding and installing applications proved to be a major hurdle for users of early smartphone platforms such as Symbian OS, RIM BlackBerry OS, and Microsoft Windows Mobile, which required manual app installation. Using a PC Web browser, the user navigates to a search engine or app aggregation website to find and download an app, and then must connect a USB cable between the PC and the phone to install the application.

Apple's release of the App Store in 2008 triggered a surge in smartphone popularity. Markets benefit developers by simplifying app discovery, sales, and distribution. More importantly, markets benefit users by simplifying app discovery, purchase, and installation. In fact, the simplicity and ease of use of this one-click installation model has led to over 10 billion downloads in only a few years [3], and was quickly adopted by all other major smartphone platforms.

Application markets can provide several types of security utility. First, they can implement a walled-garden, where the market maintainers have exclusive control over what applications users can install. Second, they can provide a choke point for application security certification. Finally, they can provide remote software management. We compare and contrast Apple's App Store and Google's Android Market to demonstrate these features.

Apple currently implements the walled-garden model for iOS devices. In contrast, Android allows users to install applications from any source, including additional application markets (e.g., the Amazon AppStore). This is often cited as both a feature and security drawback. However, to install a non-Android Market application, the user must change default settings. Most users leave default settings, and therefore are restricted to the applications available in the Android Market. Furthermore, the Android Market restricts what applications are available based on the cellular provider and handset model. Initially, AT&T disabled the ability for its Android devices to install applications from non-Android Market sources, and effectively implementing a walled-garden.

Markets can also provide a choke point for security certification. A walled-garden ensures this, but it is not necessary. If Android users use default settings, they can also benefit. The level of security tests currently implemented is unclear. Apple performs software tests, but they are not forthcoming to the extent of which are for security. Google performs no testing of applications acceptance into the Android Market; however, they have quickly removed malware when identified by researchers [14]. Given recent discoveries of Android malware [30], they likely perform some "unofficial" security analysis after acceptance.

Finally, markets can remotely manage software on handsets. Software management is a historically difficult challenge in desktop environments. Application markets provide a balance of remote administration that allows users to feel like they are in control, but can intervene when necessary. Google recently demonstrated the value of this model when it not only remotely uninstalled malware from handsets, but also pushed a security patch application that repaired changes to the OS made by the malware [2]. By placing this ability in the market, it is unclear whether users actually need antivirus software.

2.2 Platform Protection

In traditional desktop systems, OS protection policy is based on the user: applications run as the user and can access all the user's files. In contrast, smartphone OS protection policy is based on applications. By default, each smartphone application is isolated, e.g., sandbox policies in iOS, and `uid`s in Android.

Permissions. An isolated and unprivileged application has very limited functionality. Therefore, smartphone platforms allow access to individual sensitive resources (e.g., address book, GPS) using permissions. A permission is a form of capability. However, unlike capabilities, they do not always support delegation. Each platform uses permissions in slightly different ways. Au et al. [4] compare the differences between the most prominent platforms.

There are two general types of permissions: *time-of-use* and *install-time*. A time-of-use permission is approved by the user when the application executes a sensitive operation, e.g., iOS's prompt to allow an application access to location. An install-time permission is approved by the user when the application is installed. For Android, this is the user's only opportunity to deny access; the user must accept all permission requests or not install the application.

Install-time permissions serve multiple purposes. They provide [31]: *a*) user consent, *b*) defense-in-depth, and *c*) review triaging. Install-time permissions provide defense-in-depth by defining a maximum privilege level, requiring an attack on an application to additionally exploit a platform vulnerability to perform tasks outside of the application's scope. Studies have also found that applications do not simply request every permission [6,31], making them valuable attributes for security review triaging. For example, if an application does not have the permission to access location, it cannot possibly leak location information [24]. Felt et al. [31] further discuss the effectiveness of install-time permissions.

Android permissions have two additional important characteristics. First, *permission levels* restrict install-time approval; there are four levels: normal, dangerous, signature, and signature-or-system. Only dangerous permissions are presented to the user. Normal permissions are always granted and provide defense-in-depth and review triage. Signature permissions allow application developers to control permissions that provide access to exported interfaces. They are only granted to applications signed with the same developer key. Finally, signature-or-system permissions are also granted to applications signed with the firmware key (or installed in Android's "`/system`" partition). Signature permissions are primarily used to prevent third-party apps from using core system functionality.

The second characteristic is Android's limited ability for permission delegation. Permissions protecting exported database interfaces can be delegated to other applications with row-level granularity (if allowed by the database, which is not default). This allows, for example, an Email application to give an image viewer application access to a specific attachment, but not all attachments.

Application Interaction. Currently, Android is the only platform that allows flexible application communication. While Android is based on Linux, it has few similarities to a traditional UNIX-based OS. The Android middleware bases execution on *components*, not processes. By standardizing programing interfaces between components, application developers can seamlessly transfer execution between applications, and automatically find the best component and application for a task. Several articles [28,13,17] overview component interactions and security concerns, therefore, we restrict ourselves to the highlights.

Applications consist of collections of components. There are four component types: *activity*, *broadcast receiver*, *content provider*, and *service*. Android forces developers to structure applications based on the component types. Activity components define the application's user interface; each "screen" shown to the user is a different activity component. Broadcast receiver components are mailboxes to system and third-party application events, often acting as long-term callback methods. Content provider components are databases and are the primary way to share persistent data between applications. Finally, service components are daemons that define custom RPC interfaces. Each component type has standardized interfaces for interaction; one can, start an activity, broadcast a message to listening receivers, and bind to a service. This interaction is based on a primitive called an *intent message*. An important feature of intent messages is the ability to address them to implicit destinations, called *action strings*. Similar to MIME types, the Android middleware uses action strings to automatically determine which component or components should receive the intent message.

Android's application model requires developers to participate in the phone's security. They must specify (or at least influence) the security policy that protects component interfaces. This security policy is based on permissions. The Android platform defines permissions to protect itself, but developers may define new permissions. As discussed above, Android permissions are requested by and granted to applications at install time. At runtime, components can interact only if the caller application has the permission specified on the callee component. Enck et al. [28] describe additional Android security framework subtleties.

Because Android relies on developers to specify security policy, applications may introduce vulnerabilities for core system resources. Davi et al. [20] were the first to discuss privilege escalation attacks on permissions (not to be confused with attacks resulting in root privilege). They describe an attack on the Android Scripting Environment (ASE) application. The ASE application is granted the

SEND_SMS permission at install, and a malicious application is able to use the Tcl scripting interface to send SMS messages to premium-rate numbers. This scenario has also been discussed as a confused deputy attack, where a privileged application cannot (or does not) check if a caller is authorized to indirectly invoke a security sensitive operation [32,22].

3 Protection Mechanisms

Each smartphone platform defines a protection system to defend users against dangerous functionality in applications. In the previous section, we discussed permission-based protection policy. In this section, we discuss research proposals for enhancing existing smartphone protection systems, as well as their limitations, which often restrict practical deployment.

3.1 Rule Driven Policy Approach

The often-cited limitation of smartphone protection systems is insufficient policy expressibility. To address this, researchers have proposed new policy languages supporting their requirements and demonstrated how to integrate the new policy language into their target operating system. However, to make full use of these policy languages, system developers, application providers, and users need to define an appropriate policy rule-set.

Ion et al. [39] were among the first to define an extended security policy framework for mobile phones. They propose xJ2ME as an extension for J2ME based mobile devices that provides fine-grained runtime enforcement. At the time, unlimited data service plans were rare, and their policies focused on limiting the consumption of network services (e.g., data, SMS, etc). While network service use is still a security concern, unlimited (or practically unlimited, multi-GB) data service plans reduce the need for such policies. Furthermore, determining appropriate quotas for individual applications is not always straightforward, and frequently must be defined by the end user.

Similar to this work, Desmet et al. [21] propose Security-by-Contract (SxC) for the .NET platform to enhance Windows CE based phones. Conceptually, SxC allows the user or application distributor to define a policy specifying how an application should operate when it is run. The contract provides a distinct advantage over simply signing "certified" applications, as the contract can be customized for the target environment. These contracts are similar to the install-time permission model later used by Android, but provide greater expressibility. The contract policies specify allowed security related events, including access and usage quotas for the file system, network, and screen. Similar to xJ2ME, their motivating policies are difficult to define per-application.

The Kirin install-time certification system, proposed by Enck et al. [27], was the first security policy extension for Android. Enck et al. observed that while Android's install-time permissions inform the user what an application can access, they do not abstract the risk associated with specific combinations of permissions. Kirin uses both permissions and action strings listed in the application's package manifest to infer an upper bound on its functionality. Kirin modifies Android's application installer and can be used to prevent application installation, or to display statements of risk (rather than permissions) at install-time. Kirin is only as good as it's rules, therefore, Enck et al. proposed and followed a methodology based on security requirements engineering to define rules to prevent different types of dangerous functionality. Unfortunately, Kirin rules are limited by Android's permission granularity, and therefore cannot express certain policies, e.g., differentiate network destinations. Furthermore, some policies simply cannot be expressed at install-time, e.g., when an application conditionally accesses a sensitive resource such as location.

Shortly after Kirin, Ongtang et al. [52] proposed Saint. Whereas Kirin focuses on preventing malware, Saint focuses on providing more expressive security policy constraints for developers. Saint policies allow application developers to declaratively specify incoming and outgoing interactions from the point of view of their applications. It defines both install-time and runtime policies. Install-time policy rules place dependency constraints on permissions requested by applications, e.g., based on other permissions, application names, signatures, and versions. More valuable are runtime policies, for which Saint places reference monitor hooks within Android's middleware. The runtime policies specify both caller and callee constraints based on permissions, signatures, configuration, and context (e.g., location, time, etc). Providing both caller and callee policies allows an application to protect who can use its interfaces, as well as declaratively (as opposed to programmatically) restrict on who it can interface with. Like other rule-based policy frameworks, Saint's usefulness is limited by desirable policies. Ongtang et al. motivate Saint with a hypothetical shopping application that utilizes Android's ability to modularize functionality into separate applications. In follow on work [53], the authors demonstrate Saint's value by defining policies for several real applications from the OpenIntents project.

Ongtang et al. [51] also proposed Porscha to enforce digital rights management (DRM) policies for content. Porscha is specifically designed for Email, SMS, and MMS, and allows content owners to specify access control policies that restrict which applications can access the content, and under what conditions, e.g., location and maximum number of views. To do this, Porscha creates a shim in Android's SMS and network communication processing to: 1) intercept messages, 2) remove encryption that binds content to a specific device, and 3) place the messages in restricted storage that enforce content policies. Porscha provides valuable utility to enterprises and governments: the content sender can ensure only trusted applications and read and process messages. However, there is limited motivation to use Porscha for general user communication.

Several additional works have proposed fine-grained policies for Android. Conti et al. [18] proposes CRePE, an access control system for Android that enforces fine-grained policies based on context, e.g., location, time, temperature, noise, light, and the presence of other devices. Nauman et al. [47] propose the Android Permission Extension (Apex), which allows users to select which permissions an application is actually granted. Apex also supports dynamic policies, such as SMS sending quotas, and times of day that GPS can be read.

Finally, Bugiel et al. [10] propose XManDroid to mitigate permission privilege escalation attacks in Android. XManDroid seeks to prevent both confused deputy attacks and collusion between to applications (which cannot be detected by Kirin). XManDroid tracks communication between components in different applications as an undirected graph with application `uids` as vertices. System services using the same `uid` are separated using virtual vertices. Policies restrict component interaction based on communication paths and vertex properties. For example, "an application that can obtain location information must not communicate [directly or indirectly with] an application that has network access." The major hurdle for XManDroid is defining useful policies that do not result in excessive false alarms. Not all communication contains sensitive information, and when it does, it may be desired by the user. Therefore, XManDroid needs to define and maintain policy exceptions.

Observations. The obvious limitation of rule driven policy frameworks is the definition and maintenance of the rules. When proposing new frameworks, researchers must *a*) motivate the need for enhanced policy expressibility, and *b*) discuss how new policies can be identified, defined, and maintained. If researchers cannot identify a set of rules that require the full extent of the policy expressibility, they should reconsider the requirements. This aids model clarity and rule specification. For example, the final Kirin policy language [27] is significantly simpler than the original proposal [26].

Motivating policy expressibility is difficult when it is designed to address application-specific needs. In such cases, researchers should survey real applications to motivate several scenarios in which the enhanced policy is needed. Ideally, existing applications will motivate the policy expressibility. However, Android applications have been slow to adopt the platform's "applications without boundaries" mentality, and mostly operate in isolation. Therefore, proposals such as Saint must use mostly hypothetical scenarios. Anecdotally, this trend is changing, thereby allowing better motivating examples.

Policy definition and maintenance is a difficult. New proposals often gloss over the fact that their system will require users to define appropriate policy. Simultaneously useful and usable policy systems are very difficult to create. This is likely the reason Android's existing protection system strikes a balance between security and usability. In general, more specific rules often result in fewer exceptions, but require more upfront work, whereas more general rules require less upfront work, but result in more exceptions.

3.2 High-Level Policy Approach

Traditional OS protection systems such Bell-LaPadula [7] and Biba [9] define security with respect to information flow control. These approaches label processes and resources and define a mathematical specification for label interaction, e.g., "no write down," "no read up." Such approaches allow proofs of high-level security guarantees and policy correctness. In contrast, it is difficult to show that a rule driven policy is complete or correct.

Mulliner et al. [45] propose a process labeling model for Windows CE smartphones. Their goal is to prevent cross-service attacks, e.g., to prevent an exploit of a WiFi application from making phone calls. To do this, they assign labels to sensitive resources, e.g., Internet and telephony. When a process accesses a sensitive resource, the resource label is added to the process label (i.e., high-water mark). The system policy defines sets of incompatible labels based on high-level goals of preventing service interaction. An additional rule-set is required to define exceptions to the label propagation model.

A common high-level security goal for smartphones is *isolation* between business and personal applications. Isolation is achieved by defining two security domains (e.g., personal and business) and not allowing information flows between domains. OS virtualization provides a natural method of achieving this goal, e.g., run one OS instance for personal apps, and one for business apps. VMware provides a mobile solution [59]; however, it runs the business security domain as a hosted VM inside of the personal OS security domain. This slightly skewed threat model is a result of usability requirements: employees take their phone to the corporate IT for business VM installation. In contrast, Motorola is investigating bare metal hypervisors for mobile phones [36], which provide stronger security guarantees. Similarly, Lange et al. [41] propose the open source L4Android project, which uses an L4-based hypervisor.

Isolation between security domains can also be implemented within the OS. Bugiel et al. [11] propose TrustDroid, which provides lightweight domain isolation in Android. TrustDroid is extensible to many security domains, but is motivated with three: *system*, *trusted* (third-party), and *untrusted* (third-party). To allow system operation, TrustDroid allows interaction between the *system* domain and both *trusted* and *untrusted* domains. The policy prevents interaction between *trusted* and *untrusted*. To ensure an untrusted app cannot route through a system app to attack a trusted app, TrustDroid modifies system content provider and service components to enforce the isolation policy. By shifting the isolation mechanism within the OS, TrustDroid reduces the processing and memory overhead of running two separate operating systems. It also allows the user to consolidate common resources such as the address book. In the virtualized OS environment, the user must maintain two copies of such resources.

High-level policies have also been proposed to prevent confused deputy attacks in Android. Felt et al. [32] propose IPC Inspection to determine if an application should indirectly access a sensitive operation. IPC Inspection gets its name from Java Stack Inspection, which inspects the call stack for unprivileged code. However, its runtime logic has similarities to low-water mark Biba [9] in that it reduces the effective permission set on an application based on the permissions of the applications that invokes its interfaces. That is, if app A accesses app B, B's effective permissions will be reduced to the intersection of A and B's permissions. Similar to low-water mark Biba, over time, B's permissions will be reduced to ∅, therefore, IPC Inspection uses poly-instantiation of applications to reset permissions. Unfortunately, IPC Inspection fundamentally changes the semantics of an Android permission, assigning it transitive implications. This change is incompatible with applications that modularize functionality. For example, the Barcode Scanner application has the `CAMERA` permission to take a picture and return the encoded text string. Normally, the application that calls Barcode Scanner does not need the `CAMERA` permission, nor does it need to read directly from the camera. However, IPC inspection requires the caller application to have the `CAMERA` permission, thereby moving away from least privilege.

IPC Inspection assumes application developers do not properly check caller privilege. However, the challenge is determining the context in which the call originated. To address this, Dietz et al. [22] propose Quire, which records the provenance of a chain of IPC invocations. This approach provides an access control primitive for application developers rather than an enforcement model.

Observations. Android's permission-based protection system is rule driven, therefore, one must understand the semantics of individual permissions to understand the global policy. Android permissions are non-comparable and hence cannot be arranged in a lattice, nor are they intended to be transitive. Because of this, high-level policy approaches based entirely on Android permissions will inherently result in many policy exceptions. Permissions can make excellent security hints, if their semantics and limitations are kept in mind. Sensitive information is increasingly application-specific and introduced by third-party applications (e.g., financial). Therefore, application developers must contribute to the global protection policy.

3.3 Platform Hardening

Most smartphone functionality occurs within a middleware layer. This simplifies the underlying platform and allows application of traditional platform hardening technologies. As a result, mandatory access policies can be simpler. For example, Muthukumaran et al. [46] design a custom SELinux policy for OpenMoko to separate trusted and untrusted software. Shabtai et al. [57] describe their experiences porting SELinux to Android, and create a custom SELinux policy. However, they use targeted mode, whereas a strict mode would provide stronger holistic guarantees. Finally, Zhang et al. [60] apply SELinux to a generic Linux phone to provide isolated security domains consistent with the TCG's Trusted Mobile Phone specification.

Integrity measurement and remote attestation have also been applied to smartphones. The Muthukumaran et al. [46] SELinux-based installer was designed to support the policy reduced integrity measurement architecture (PRIMA). Similarly, Zhang et al. [61] discuss an efficient integrity measurement and attestation for the LiMo platform. Finally, Nauman et al. [48] provide integrity measurement of Android applications for enterprises and to prevent malware.

Observations. Device security relies on its trusted computing base (TCB), therefore platform hardening is an important component for smartphone security. However, enterprises and users should keep in mind that while SELinux and remote attestation help security, it is a building block. The most significant challenges lie in defining application-level security policies.

3.4 Multiple Users

Smartphone platform designs assume there is one physical user. This simplifies protection systems and allows them to focus on applications. However, users occasionally lend their phone in social situations. Karlson et al. [40] studied how users of different smartphone platforms lend their phone to other physical users. Their findings motivate a reduced-capability guest profile. Liu et al. [42] report similar findings and propose xShare, a modification of Windows Mobile that creates "normal" and "shared" modes. Finally, Ni et al. [49] propose DiffUser for Android. DiffUser expands a phone from a single user model to one that has three classes: administrative users, normal users, and guest users.

Observations. The studies confirm our intuition: users sometimes share their smartphones with friends for whom "full access" is undesirable. We will likely see many proposals claiming to have "the solution." Fundamentally, this problem requires user participation, unless the phone can reliably predict which applications the owner would like the current physical user to access (e.g., Web browser and games, but not Email, except when the user needs to share an Email). As existing proposals have shown, modifying a platform to provide a "guest mode" is not terribly complex. Therefore, future research must demonstrate usability.

3.5 Faking Sensitive Information

Studies [24,23,25] have identified many smartphone applications leaking phone identifiers and location to servers. In response, Beresford et al. [8] propose providing fake or "mock" information to applications. Their system, MockDroid, returns fake fixed values for location and phone identifiers. MockDroid also fakes Internet connections (by timing out connections), intent broadcasts (by silently dropping them), and SMS/MMS, calendar, and contacts content providers (by return "empty" results). To enable fake data, users must configure "mocked permissions" for each application. TISSA, proposed by Zhou et al. [62], has a similar design with slightly greater flexibility, allowing users to choose from empty, anonymized, or fake results for location, phone identity, contacts, and call logs. Finally, Hornyack et al. [38] propose AppFence. In addition to substituting fake data for phone identifiers and location, AppFence uses TaintDroid [24] (discussed in Section 4) to block network transmissions containing information specified by the user to be used on-device only. AppFence also uses "salted" phone identifiers, which are guaranteed to be unique to a specific application and phone, but different between applications on the phone. This technique allows application developers to track application usage without compromising user privacy.

Observations. Transparently incorporating fake information is an elegant way to get around the Android's limitation of not allowing users to deny specific permissions to applications. While permission selection is trivial to implement, it will likely break many existing applications, and therefore is unlikely to be included in the official Android distribution. A second argument against permission selection is usability. Proposals to insert fake information have the same, if not worse, usability limitations. Nonetheless, there is user demand for more control over privacy sensitive information. Finally, there are hidden consequences to faking sensitive information. Many suspect privacy sensitive values are the basis of an advertisement and analytics economy. Restricting privacy values may in turn increase the monetary cost of applications.

4 Application Analysis

As discussed in Section 2, application markets are the primary means of delivering applications to end users. Hence, they can be used as a security choke-point for identifying malicious and dangerous applications. One difficulty of using markets in this manner is a lack of a common definition for "unwanted" applications. Markets quickly remove malicious applications. However, malicious intent is not always clear. Should a market remove applications meant to monitor (i.e., spy on) children? Should an open market, e.g., the Android Market, remove applications that exploit system vulnerabilities to provide the user desired functionality? Beyond this, there is a class of dangerous functionality that many reputable applications include, specifically disclosing privacy sensitive information such as geographic location and phone identifiers without informed consent by the user.

There are limits to the security protections that can be provided by markets [43]. However, recent advancements in application analysis are moving towards more automated certification. In this section, we discuss several approaches for identifying malware and grayware (i.e., dangerous apps without provable malicious intent).

4.1 Permission Analysis

Permissions articulate protection policy, but they also describe what an application can do once installed. As described in Section 3, Enck et al. [27] were the first to use Android permissions to identify dangerous functionality. Kirin breaks dangerous functionality down into the permissions required to perform it. If an application does not have a requisite permission, the attack cannot occur (without exploiting a vulnerability). Enck et al. used Kirin to study 311 top free applications across different Android Market categories. Their rules flagged 10 applications, 5 of which were questionable after reviewing their purpose.

Following this work, Barrera et al. [6] performed permission analysis of the top 50 free applications of every category of the Android Market (1,100 apps in total). They report an exponential decay in the number of applications requesting individual permissions, i.e., many applications request only a small set of permissions. Barrera et al. also use Self Organizing Maps (SOM) to analyze permission usage. SOM maps the highly dimensional permission space onto a 2-dimensional U-matrix, allowing visual inspection of application permission use. They use heat-maps to show permission frequency in the cells, generating a U-matrix for each permission. By comparing U-matrices for different permissions, one can identify permissions that are frequently requested together. Barrera et al. also labeled cells with categories using a winner-take-all strategy. That is, if most applications mapped to a cell are from the "Multimedia" category, then that cell is marked as "Multimedia." However, their findings do not indicate any correlation between categories and permission requests.

Finally, Felt et al. [31] studied the effectiveness of Android's install-time permissions. They considered 100 paid and 856 free applications from the Android Market. Similar to Barrera et al., they found that most applications request a small number of permissions. They also analyzed the frequency of permission requests, comparing free and paid apps. The `INTERNET` permission is by far the most frequently requested. They also found that developers make obvious errors, e.g., requesting non-existent permissions. In follow on work, Felt et al. [29] create a mapping between Android APIs and permissions and propose the Stowaway tool to detect over-privilege in applications. Note that to do this, Stowaway performs static analysis of applications (discussed below). Felt et al. report the 10 most common unnecessary permissions, the top 2 of which are `ACCESS_NETWORK_STATE` and `READ_PHONE_STATE`.

Observations. Permissions are valuable for performance efficient security analysis, but they do not tell the whole story. The Android platform developers made security and usability trade-offs when defining permissions, and many researchers have noted granularity issues. For example, the `READ_PHONE_STATE` permission is used to protect the APIs for both determining if the phone is ringing *and* for retrieving phone identifiers. This leads to ambiguity during permission analysis. A second culprit of ambiguity is the `INTERNET` permission: most applications do not need access to all network domains. However, unlike `READ_PHONE_STATE`, making `INTERNET` more granular is nontrivial in Android, as enforcement is performed in the kernel based on a `gid` assigned to applications. At this enforcement point, the DNS name is no longer available. That said, to date, Android application developers are not significantly over-requesting permissions, which leaves some potential for identifying dangerous applications by their permissions. However, studies indicate considering permissions alone is limited, and they are likely best used to steer dynamic and static analysis.

4.2 Dynamic Analysis

Researchers began with permission analysis because application source code was not available. The next step in studying applications is dynamic analysis, i.e., watching applications run. Dynamic analysis can help resolve ambiguity in permission granularity. It also resolves configuration dependencies. For example, the

Kirin study identified applications that only send geographic location information to a network server if the user changes a default configuration.

Enck et al. [24] propose TaintDroid to identify when applications send privacy sensitive information to network servers. To do this, TaintDroid uses dynamic taint analysis, also known as taint tracking. This technique marks information at source APIs when its type is unambiguous. Smartphones have many such sources, e.g., location, camera, microphone, and phone identifiers. Next, the taint tracking system automatically propagates the markings on some granularity, e.g., individual instructions: $a = b + c$. Enck et al. modified Android's Dalvik VM to perform instruction-level taint tracking. They also integrate the taint tracking into the broader system using coarser granularities, e.g., files and IPC messages. Finally, at a taint sink, the taint tracking system inspects markings on API parameters and performs a policy action. TaintDroid uses the network APIs as taint sinks and logs the event if a taint marking exists in a data buffer. Enck et al. used TaintDroid to study 30 popular applications from the Android Market and found 15 sharing location with advertisers and 7 sharing phone identifiers with remote servers, all without the users knowledge.

TaintDroid has several limitations, discussed in the paper. First, TaintDroid can only identify that privacy sensitive information has left the phone, and not if the event is a privacy violation. Determining a privacy violations requires knowledge of (a) if the user was aware or intended it to occur (there are many desirable location-aware applications), and (b) what the remote server does with the value. Most researchers and users are only capable of identifying (a), therefore leaking information without the user's knowledge has generally been considered a privacy violation. Second, TaintDroid only tracks explicit flows. Therefore, a malicious developer can use implicit flows within an application to "scrub" taint markings from variables. However, such actions are likely identifiable using static analysis and draw attention to developers for attempting to hide their tracks.

The TaintDroid analysis framework was made open source and subsequently used by several researchers. MockDroid [8] and TISSA [62] (discussed in Section 3.5) use TaintDroid to evaluate their effectiveness. AppFence [38] (also discussed in Section 3.5) adds enforcement policies to TaintDroid. The authors also study additional applications and characterize privacy exposure. Finally, Gilbert et al. [35] extend TaintDroid to track specific types of implicit flows and discuss approaches for automating application analysis. They find that random inputs commonly get "stuck" in parts of applications' UI. Therefore, they use concolic execution, switching between symbolic and concrete execution as necessary.

Observations. Dynamic analysis identifies what actually happens when an application is run. Static analysis (discussed next) cannot capture all runtime configuration and input. For example, the AdMob SDK documentation [1] indicates it will only send location information if a configuration value is set in the application's manifest file. Furthermore, applications can download and execute code, which is not available for static analysis. However, dynamic analysis is limited by scalability. As discussed by Gilbert et al. [35], generating test inputs is hard. Finally, any automated analysis is limited in its ability to understand user intentions. Ideally, automated privacy analysis should only raise alarms for privacy violations. Researchers seeking to scale tools such as TaintDroid must attempt to characterize identified leaks.

4.3 Static Analysis

Static program analysis can be done with or without source code. Egele et al. [23] propose PiOS to perform static taint analysis directly on iOS application binaries. PiOS reconstructs control flow graphs from compiled Objective-C, which is nontrivial because object method invocation is funneled through a single dispatch routine. Interestingly, Egele et al. found that iOS's handling of user interactions disrupts the control flow in the CFG. Therefore, to identify potential privacy violations, PiOS uses control flow analysis on the CFG, followed by data flow analysis to confirm information reached the sink. Egele et al. use PiOS to study 825 free applications form Apple's App Store, and 582 applications from Cydia's BigBoss repository. They find that more than half leak the privacy sensitive device ID without the user's knowledge. They also report a strong penetration of ad and analytics libraries.

Android researchers have also performed static analysis of low-level representations. Chin et al. [17] propose ComDroid, which operates on use disassembled DEX bytecode. ComDroid identifies vulnerabilities in Intent communication between applications, including: broadcast theft, activity hijacking, service hijacking, malicious broadcast injection, malicious activity launch, and malicious service launch. Chin et al. used ComDroid to analyze 50 popular paid and 50 popular free applications, manually inspecting the results of 20. In these 20 applications, they found 34 exploitable vulnerabilities. Other tools developed by this group, including IPC Inspection [32] and Stowaway [29] (discussed above), build upon ComDroid. However, working directly on DEX bytecode is difficult. As noted in the ComDroid paper [17], its control flow analysis follows all branches, which can result in false negatives.

In contrast, Enck et al. [25] propose `ded` to reverse Android applications to their original Java form, for which sophisticated static program analysis tools already exist. Reversing DEX bytecode to Java bytecode is nontrivial: the JVM is stack-based while the DVM is register-based; DEX inserts scalar constants throughout the bytecode, and most importantly, DEX loses the type semantics of scalars in several important situations. Using `ded`, Enck et al. decompile 1,100 popular applications and perform a breadth of security program analysis. They target both dangerous functionality and vulnerabilities using custom rules specified for the Fortify SCA framework and follow the program analysis with substantial manual inspection of results. In doing so, they report many observations that provide insight into how Android applications are developed. Overall, their findings were similar to previous privacy studies, and echo concerns with Intent APIs. Similar to the iOS study [23], Enck et al. also found a strong penetration of ad and analytics libraries.

Finally, researchers modeled Android component interaction using source code analysis. Chaudhuri [15] proposes a formal model for tracking flows between applications using permissions as security types. In follow-on work, Fuchs et al. [33] propose SCanDroid for automated application certification using the WALA Java bytecode analysis framework. However, using permissions as the basis of security type analysis in Android is limited, since most permissions are non-comparable and cannot be partially ordered. SCanDroid was proposed before `ded` was available, and therefore was only evaluated against open source applications. Moving forward, combining SCanDroid's formal model and analysis tools with the motivations of ComDroid [17] and IPC Inspection [32] and applying it to code recovered by `ded` has potential for more accurate results.

Observations. Static code analysis of Android applications is not as simple as one might initially think. While Fortify SCA was useful, Enck et al. [25] found that custom tools are required to overcome analysis hurdles created by the Android middleware. For example, component IPC must be tracked through the middleware, the middleware API has many callbacks that indirectly use IPC, and APIs frequently require depend on variable state (e.g., the address book content provider authority string). Additionally, researchers should continue to look beyond privacy analysis. While static analysis can scale the identification of potential privacy leaks, their existence is well known. The challenge for privacy leak analysis is automatically determining if the leak was desired.

4.4 Cloud-Based Monitoring

Early smartphone security analysis monitored application behavior from the cloud. Cheng et al. [16] propose SmartSiren, which sends logs of device activity, e.g., SMS and Bluetooth, to a server for aggregate analysis to detect virus and worm outbreaks. Oberheide et al. [50] use virtualized in-cloud security services provided by CloudAV for SMS spam filtering, phishing detection, and centralized blacklists for Bluetooth and IP addresses. Schmidt et al. [55] send device features such as free RAM, user activity, process count, CPU usage, and number of sent SMS messages to a central server for intrusion detection analysis. A similar approach is taken by Shabtai et al. [58] in their "Andromaly" proposal for Android. Portokalidis et al. [54] propose "Paranoid Android," which creates a clone of an Android phone in the cloud. A proxy sits in the network so that the network traffic does not need to be uploaded to the server from the phone, and they use "loose synchronization" to only send data when the user is using the device (to safe energy). Finally, Burguera et al. [12], propose Crowdroid, which crowd-sources intrusion detection based on syscalls used by applications.

Observations. Before all of this work, Miettinen et al. [44] discussed the limitations of network based intrusion detection for malicious behavior in smartphones. Their arguments include: (1) administrational boundaries, (2) technical boundaries (e.g., network connection), and (3) conception limitations (e.g., attacks to local storage not in view of network). While sending logs and virtualization address (3), the former to remain valid. Specifically, Miettinen et al. discuss the need to ensure that systems do not expose private data to the cloud services. It is unclear what level of privacy and administrative control users are willing to lose in order to gain security. As mentioned in Section 2, application market kill switches and software management strike a careful balance.

5 Additional Research Directions

In Sections 3 and 4, we discussed existing research proposals, their limitations, and concluded each discussion area with potential enhancements and future directions. In this section, discuss several additional areas with promise. None of these areas are new for computer security, and each has inherent limitations.

Application Discovery. There are hundreds of thousands of applications available for iOS and Android, many of which are practically useless and duplicates of one another. When searching for a new application, the user has to balance *a*) price, *b*) functionality, *c*) aesthetics, and *d*) security (and security is unfortunately often the last consideration). Recommendations are often made via word of mouth, but social search will likely soon emerge. Review services such as Consumer Reports have addressed the first three criteria for decades. As discussed in Section 4, there is no one-size-fits-all criteria for security and privacy. Users have different requirements, particularly when privacy is concerned. One potential model is to use Kirin [27] rules to influence security ratings. To be successful, security reviews need to be integrated into application discovery user interfaces, e.g., application markets. Along these lines, Barrera et al. [5] propose Stratus to consolidate multiple application markets, which can address malware opportunities that arise when bargain shoppers compare prices between markets.

Modularity and Transitivity. Android allows developers to be compartmentalize functionality into multiple applications. This has several advantages: 1) it supports least privilege, 2) it creates boundaries that allow OS mediation, and 3) it simplifies application analysis by defining distinct purposes for applications. Advertisement and analytics functionality is an immediate and real example of where compartmentalization can benefit security. Often, applications only require Internet access to support ads or analytics. Splitting off this functionality reduces the privilege needed by applications and allows certification tools to focus on ad and analytics functionality. However, as noted by several researchers [20,32,22], separating functionality into applications can result in privilege escalation attacks, because Android's permissions are not transitive. Unfortunately, as discussed in Section 3.2, making permissions transitive is not a practical solution. Therefore, a new security primitive may be required.

Security via UI Workflow. Security policies are difficult for users to understand, and there have been many complaints that Android relies on the user to approve install-time permission requests. Security enforcement does not always need to be an explicit permission or policy statement. Consider the two methods of making phone calls in Android. If an application uses the "`CALL`" action string, it requires the `CALL_PHONE` permission, and the call is connected immediately; however, if the application uses the "`DIAL`" action string, no permission is required, and the user is presented the phone's default dialer with the number entered. Realistically, all applications should use the "`DIAL`" action string (unless it replaces the dialer), because the user is naturally involved in the security decision via the workflow. There is no security question, e.g., "allow location," and the user is never aware that a security decision was made. Future research should investigate opportunities to integrate security into the UI workflow.

Developer Tools. Studies [25,17,32] have shown that developers need more oversight when using security sensitive APIs. In particular, these studies have reported vulnerabilities at application interfaces, i.e., Intents. Developer tools should be enhanced with checks that look for Intent forging attacks, unprotected Intent broadcasts, and confused deputy attacks. For confused deputies, the developer may not have sufficient context to prevent an attack, therefore new primitives such as IPC provenance [22] are required. Additionally, research is needed to ensure that the new security enhanced developer tools are usable, and not simply discarded by developers.

6 Conclusion

Smartphone security research is growing in popularity. To help direct future research, we have described existing protections and surveyed research proposals to enhance security, discussing their advantages and limitations. The proposals have discussed enhanced on-phone protection, as well as application analysis that will aid future certification services. Finally, we discussed several additional areas for future smartphone security research.

Acknowledgement. We would like to thank David Barrera, Kevin Butler, Patrick McDaniel, Patrick Traynor, and Paul Van Oorschot, for their comments during the writing of this paper.

References

1. AdMob: AdMob Android SDK: Installation Instructions, `http://www.admob.com/docs/AdMob_Android_SDK_Instructions.pdf` (accessed November 2010)
2. Android Market: March 2011 Security Issue (March 2011), `https://market.android.com/support/bin/answer.py?answer=1207928`
3. Apple Inc.: Apple's App Store Downloads Top 10 Billion (January 2011), `http://www.apple.com/pr/library/2011/01/22appstore.html`
4. Au, K., Zhou, B., Huang, Z., Gill, P., Lie, D.: Short Paper: A Look at SmartPhone Permission Models. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)

5. Barrera, D., Enck, W., van Oorschot, P.C.: Seeding a Security-Enhancing Infrastructure for Multi-market Application Ecosystems. Tech. Rep. TR-11-06, Carleton University, School of Computer Science, Ottawa, ON, Canada (April 2011)
6. Barrera, D., Kayacik, H.G., van Oorshot, P.C., Somayaji, A.: A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In: Proceedings of the ACM Conference on Computer and Communications Security (October 2010)
7. Bell, D.E., LaPadula, L.J.: Secure Computer Systems: Mathematical Foundations. Tech. Rep. MTR-2547, Vol. 1, MITRE Corp., Bedford, MA (1973)
8. Beresford, A.R., Rice, A., Skehin, N., Sohan, R.: MockDroid: Trading Privacy for Application Functionality on Smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, HotMobile (2011)
9. Biba, K.J.: Integrity considerations for secure computer systems. Tech. Rep. MTR-3153, MITRE (April 1977)
10. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.R.: XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. Tech. Rep. TR-2011-04, Technische Universitat Darmstadt, Center for Advanced Security Research Darmstadt, Darmstadt, Germany (April 2011)
11. Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.R., Shastry, B.: Practical and Lightweight Domain Isolation on Android. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)
12. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: Behavior-Based Malware Detection System for Android. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)
13. Burns, J.: Developing Secure Mobile Applications for Android. iSEC Partners (October 2008), http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf
14. Cannings, R.: Exercising Our Remote Application Removal Feature (June 2010), http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html
15. Chaudhuri, A.: Language-Based Security on Android. In: Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS) (June 2009)
16. Cheng, J., Wong, S.H., Yang, H., Lu, S.: SmartSiren: Virus Detection and Alert for Smartphones. In: Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys) (June 2007)
17. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.: Analyzing Inter-Application Communication in Android. In: Proceedings of the 9th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys (2011)
18. Conti, M., Nguyen, V.T.N., Crispo, B.: CRePE: Context-Related Policy Enforcement for Android. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 331–345. Springer, Heidelberg (2011)
19. Dagon, D., Martin, T., Starner, T.: Mobile Phones as Computing Devices: The Viruses are Coming! IEEE Pervasive Computing 3(4), 11–15 (2004)
20. Davi, L., Dmitrienko, A., Sadeghi, A.-R., Winandy, M.: Privilege Escalation Attacks on Android. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 346–360. Springer, Heidelberg (2011)
21. Desmet, L., Joosen, W., Massacci, F., Philippaerts, P., Piessens, F., Siahaan, I., Vanoverberghe, D.: Security-by-contract on the. NET platform. Information Security Technical Report 13(1), 25–32 (2008)

22. Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., Wallach, D.S.: Quire: Lightweight Provenance for Smart Phone Operating Systems. In: Proceedings of the 20th USENIX Security Symposium (August 2011)
23. Egele, M., Kruegel, C., Kirda, E., Vigna, G.: PiOS: Detecting Privacy Leaks in iOS Applications. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS) (February 2011)
24. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (October 2010)
25. Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S.: A Study of Android Application Security. In: Proceedings of the 20th USENIX Security Symposium (August 2011)
26. Enck, W., Ongtang, M., McDaniel, P.: Mitigating Android Software Misuse Before It Happens. Tech. Rep. NAS-TR-0094-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA (September 2008)
27. Enck, W., Ongtang, M., McDaniel, P.: On Lightweight Mobile Phone Application Certification. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS) (November 2009)
28. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android Security. IEEE Security & Privacy Magazine 7(1), 50–57 (2009)
29. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android Permissions Demystified. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2011)
30. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A Survey of Mobile Malware in the Wild. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)
31. Felt, A.P., Greenwood, K., Wagner, D.: The Effectiveness of Application Permissions. In: Proceedings of the USENIX Conference on Web Application Development, WebApps (2011)
32. Felt, A.P., Wang, H.J., Moshchuk, A., Hanna, S., Chin, E.: Permission Re-Delegation: Attacks and Defenses. In: Proceedings of the 20th USENIX Security Symposium (August 2011)
33. Fuchs, A.P., Chaudhuri, A., Foster, J.S.: ScanDroid: Automated Security Certification of Android Applications, `http://www.cs.umd.edu/~avik/projects/scandroidascaa/paper.pdf` (accessed January 11, 2011)
34. Gartner: Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent (August 2011), `http://www.gartner.com/it/page.jsp?id=1764714`
35. Gilbert, P., Chun, B.G., Cox, L.P., Jung, J.: Vision: Automated Security Validation of Mobile Apps at App Markets. In: Proceedings of the International Workshop on Mobile Cloud Computing and Services, MCS (2011)
36. Gudeth, K., Pirretti, M., Hoeper, K., Buskey, R.: Short Paper: Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)
37. Guo, C., Wang, H.J., Zhu, W.: Smart-Phone Attacks and Defenses. In: Proceedings of the 3rd Workshop on Hot Topics in Networks, HotNets (2004)

38. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2011)
39. Ion, I., Dragovic, B., Crispo, B.: Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC) (December 2007)
40. Karlson, A.K., Brush, A.B., Schechter, S.: Can I Borrow Your Phone? Understanding Concerns When Sharing Mobile Phones. In: Proceedings of the Conference on Human Factors in Computing Systems (CHI) (April 2009)
41. Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., Peter, M.: L4Android: A Generic Operating System Framework for Secure Smartphones. In: Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices, SPSM (2011)
42. Liu, Y., Rahmati, A., Huang, Y., Jang, H., Zhong, L., Zhang, Y., Zhang, S.: xShare: Supporting Impromptu Sharing of Mobile Phones. In: Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys) (June 2009)
43. McDaniel, P., Enck, W.: Not So Great Expectations: Why Application Markets Haven't Failed Security. IEEE Security & Privacy Magazine 8(5), 76–78 (2010)
44. Miettinen, M., Halonen, P., Hatonen, K.: Host-Based Intrusion Detection for Advanced Mobile Devices. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA) (April 2006)
45. Mulliner, C., Vigna, G., Dagon, D., Lee, W.: Using Labeling to Prevent Cross-Service Attacks Against Smart Phones. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 91–108. Springer, Heidelberg (2006)
46. Muthukumaran, D., Sawani, A., Schiffman, J., Jung, B.M., Jaeger, T.: Measuring Integrity on Mobile Phone Systems. In: Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 155–164 (June 2008)
47. Nauman, M., Khan, S., Zhang, X.: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In: Proceedings of ASIACCS (2010)
48. Nauman, M., Khan, S., Zhang, X., Seifert, J.-P.: Beyond Kernel-Level Integrity Measurement: Enabling Remote Attestation for the Android Platform. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 1–15. Springer, Heidelberg (2010)
49. Ni, X., Yang, Z., Bai, X., Champion, A.C., Xuan, D.: DiffUser: Differentiated User Access Control on Smartphones. In: Proceedings of the 5th IEEE Workshop on Wireless and Sensor Networks Security (WSNS) (October 2009)
50. Oberheide, J., Veeraraghavan, K., Cooke, E., Flinn, J., Jahanian, F.: Virtualized In-Cloud Security Services for Mobile Devices. In: Proceedings of the 1st Workshop on Virtualization in Mobile Computing (June 2008)
51. Ongtang, M., Butler, K., McDaniel, P.: Porscha: Policy Oriented Secure Content Handling in Android. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC) (December 2010)
52. Ongtang, M., McLaughlin, S., Enck, W., McDaniel, P.: Semantically Rich Application-Centric Security in Android. In: Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC), pp. 340–349 (December 2009)
53. Ongtang, M., McLaughlin, S., Enck, W., McDaniel, P.: Semantically Rich Application-Centric Security in Android. Journal of Security and Communication Networks (2011), (Published online August 2011)

54. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H.: Paranoid Android: Versatile Protection For Smartphones. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC) (December 2010)
55. Schmidt, A.D., Peters, F., Lamour, F., Albayrak, S.: Monitoring Smartphones for Anomaly Detection. In: Proceedings of the 1st International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, MOBILWARE (2008)
56. Schmidt, A.D., Schmidt, H.G., Batyuk, L., Clausen, J.H., Camtepe, S.A., Albayrak, S.: Smartphone Malware Evolution Revisited: Android Next Target? In: Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE) (October 2009)
57. Shabtai, A., Fledel, Y., Elovici, Y.: Securing Android-Powered Mobile Devices Using SELinux. IEEE Security and Privacy Magazine (May/June 2010)
58. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: "Andromaly": A Behavioral Malware Detection Framework for Android Devices. Journal of Intelligent Information Systems (2011), (published online January 2011)
59. VMware, Inc.: VMware Mobile Virtualization Platform, http://www.vmware.com/products/mobile/ (accessed January 2011)
60. Zhang, X., Aciiçmez, O., Seifert, J.P.: A Trusted Mobile Phone Reference Architecture via Secure Kernel. In: Proceedings of the ACM workshop on Scalable Trusted Computing, pp. 7–14 (November 2007)
61. Zhang, X., Acıiçmez, O., Seifert, J.-P.: Building efficient integrity measurement and Attestation for Mobile Phone Platforms. In: Schmidt, A.U., Lian, S. (eds.) MobiSec 2009. LNICST, vol. 17, pp. 71–82. Springer, Heidelberg (2009)
62. Zhou, Y., Zhang, X., Jiang, X., Freeh, V.W.: Taming Information-Stealing Smartphone Applications (on Android). In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.-R., Sasse, A., Beres, Y. (eds.) Trust 2011. LNCS, vol. 6740, pp. 93–107. Springer, Heidelberg (2011)

Secure Composition of Cryptographic Protocols

Vipul Goyal

Microsoft Research, India
vipul@microsoft.com

1 Talk Overview

General positive results for secure computation were obtained more than two decades ago. These results were for the setting where each protocol execution is done in isolation. With the proliferation of the network setting (and especially the internet), an ambitious effort to generalize these results and obtain concurrently secure protocols was started. However it was soon shown that designing secure protocols in the concurrent setting is unfortunately impossible in general. In this talk, we will first describe the so called chosen protocol attack. This is an explicit attack which establishes general impossibility of designing secure protocols in the concurrent setting. The negative results hold for the so called plain model where there is no trusted party, no honest majority, etc.

On the other hand, several *positive* results for protocols composition have been established in various related settings (which are either weaker or incomparable). A few examples are the setting of resettable computation (where the parties may not be able to keep state during the protocol execution and may be run several times with the same random tape), bounded concurrent secure computation (where there is an apriori bound on the total number of concurrent sessions), standalone protocol execution with man-in-the-middle (i.e., the setting of non-malleable protocols), the single input setting (where the honest party uses the same input in all polynomially unbounded concurrent protocol executions), etc.

We will survey known results as well various open problems in each of the above settings. We also given an overview of an emerging technique which has been used to construct secure protocols in several of these settings. We will focus on the plain model throughout the talk.

Flow Based Interpretation of Access Control: Detection of Illegal Information Flows

Mathieu Jaume[1], Valérie Viet Triem Tong[2], and Ludovic Mé[2]

[1] University Pierre & Marie Curie, LIP6, Paris, France
[2] SUPELEC, SSIR Group (EA 4039), Rennes, France

Abstract. In this paper, we introduce a formal property characterizing access control policies for which the interpretations of access control as mechanism over objects and as mechanism over information contained into objects are similar. This leads us to define both a flow based interpretation of access control policies and the information flows generated during the executions of a system implementing an access control mechanism. When these two interpretations are not equivalent, we propose to add a mechanism dedicated to illegal information flow detection to the mechanism of access control over objects. Such a mechanism is parameterized by the access control policy and is proved sound and complete. Finally, we briefly describe two real implementations, at two levels of granularity, of our illegal flow detection mechanism: one for the Linux operating system and one for the Java Virtual Machine. We show that the whole approach is effective in detecting real life computer attacks.

Keywords: Access control models, Information flows.

1 Introduction

The classical approach to protect information in a system consists in defining a security policy and enforcing it with diverse security mechanisms. Among these mechanisms, access control, which allows to grant or revoke the rights for active entities (the subjects) to access some passive entities (the objects), is of particular importance. Unfortunately, classical access control mechanisms implemented nowadays in mainly used computer systems cannot always control how the information is used once it has been accessed. As an example, let us consider an access control policy where granted accesses are specified by the matrix:

	o_1	o_2	o_3	o_4
Alice	read, write		read	
Bob	read	read, write		
Charlie		read, write		write

(1)

In this example, an authorized user can pass information to other users that are not authorized to read or write it. Indeed, Alice can read o_3 and write information contained in o_3 into o_1 on which Bob can make a read access, even if Bob cannot read o_3. Similarly, Bob can write some information into o_2 and then Charlie can read o_2 and write this information into o_4, even if Bob cannot write into o_4. To overcome this difficulty, flow control has been proposed. On the contrary to access control that does not provide a sufficient protection of information as it is not aware of the *can flow* relationship between objects in the system, flow control ensures that data contained in a container cannot flow into other containers readable by users that cannot read the original container. The more noticeable model of that kind is of course the Bell & La Padula model [1] where high level information cannot flow into low level containers. However, flow control has not been widely used until now for everyday computers running Windows or Unix-like OS. We believe nevertheless that flow control is crucial for a system to be secured: many real world attacks imply a series of flows, generated by several subjects, each flow being legal by itself (and hence not stopped by access control), but the composition of the flows resulting in information disclosure or integrity violation. After a presentation of some related works, we present a framework that enables us to formally define information flows induced by an access control policy over objects (section 2). Then we characterize flow policies induced by access control policies over information (section 3) and a mechanism for detecting illegal information flows in an operating system (section 4). Lastly we present an implementation of the monitor of information flows (section 5).

Related Works. Denning & al were the first to propose a static approach to check the legality of information flows in programs [3,4]. The legality of information flows is expressed through a policy which is specified using a lattice of security labels. The static analysis of programs consists in checking the legality of information flows, with regard to the security labels that are associated with data containers of the programs. We follow a similar approach with reading and writing tags, but we aim at a dynamic detection, without a previous static analysis. Many previous works are related to the information flows that might appear during the lifetime of a system where access to information is under the control of an access control policy. In [15], Sandhu presents a synthesis of Lattice-Based Access Control Models (LBAC), allowing to ensure that information only flows in authorized security classes. Nowadays, the system SELinux descends from these works and enforces a Mandatory Access Control (MAC) policy based on the Bell & LaPadula model. Unfortunately, SElinux is notoriously hard-to-configure. We thus propose a system that can be configured directly from an access control policy. Furthermore, in a real system, the use of a strict LBAC model appears quite rigid since applications often need declassification mechanisms [13,12,11]. As systems grow larger and more complex, such rigid models are not often used, users simply preferring an access control model like HRU or RBAC[16] without any control of information flows. Hence, we believe it is important to provide another mechanism to ensure that information flows are consistent with the policy. With the same objective, Osborn proposes in [14] to check if a given access control policy expressed within the RBAC model has an equivalent information flow policy in the LBAC model and if this last model is consistent with the security policy. More precisely, [14] describes an algorithm to map a role-graph of a RBAC model to an information flow graph defining the *can flow* relationship of the given role graph, thus characterizing the information flows that can occur. We follow such an approach to study the *can flow* relationship generated by an arbitrary access control model and we provide a formal framework allowing to express the desired properties at an abstract level. A similar approach can be found in [19], where Zimmermann & al have proposed to complement classical non-transitive access control by a dynamic detection mechanism able to identify illegal composed information flows. In this development a reading tag is associated with each information (the initial contents of each object) and a writing tag to each object. The initial values of these tags comes directly from the access control matrix which specify the security policy enforced on the system. When an information flow occurs between a read object o_1 and a written object o_2, the reading tag of o_2 is updated according to the one of o_1. Thanks to this propagation mechanism, the detector is able to identify accesses that transitively create illegal information flows. Nevertheless, this detection model can only be used with an access control expressed by an matrix of rights, and there is no proof of its soundness and its completeness. Following the same idea, in [10], Ko and Redmond introduce a policy-based intrusion detection system dealing with information flows and allowing to detect data integrity violations by monitoring the users' actions' effects on the contents of data containers (mainly files). The detection process is achieved by controlling noninterference between users and data. They show that although the noninterference model itself is not always an enforceable security policy [17], it can nevertheless be used as a foundation for intrusion detection by taking into account the semantics of individual actions (in their case, file-related Linux system calls). Our approach is similar but allows to express confinment, integrity and confidentiality properties (it is not limited to data integrity) and we are thus able to detect a wider range of attacks, while Ko and Redmond's IDS is limited by construction to race condition-based attacks. Moreover, from a theoretical point of view, they only give a proof of the soundness of their approach, while completeness is not studied.

2 Access Control Models and Induced Information Flows

Our objective is to propose a dynamic monitor of information flows for currently used computers whose aim is to raise an alert when observing an illegal information flow. Hence, this monitor must be able to deduce some information flows from the observation of accesses. An information flow from an object o_1 to an object o_2 can be either *elementary* when a user performs at the same time a read access on o_1 and a write access on o_2 or *composed* when a user performs a read or write access on an object and closes a read/write chain between o_1 and o_2 that allows information to flow from o_1 to o_2. An elementary flow is always legal since it is directly permitted by the access control policy. We consider here that a composed information flow is legal if can be obtained as a (legal) elementary flow. Practically, our monitor tracks elementary information flows and checks dynamically if resulting composed flows are legal or not.

Access Control Policies. We introduce here an abstract specification of access control policies over objects together with operational mechanisms allowing to enforce them. By following the approach introduced in [9], a security policy is a characterization of secure elements of a set according to some security information. Hence, specifying a policy $\mathbb{P}$ first consists of defining a set $\mathbb{A}$ of "things" that the policy aims to control, called the security targets, in order to ensure the desired security properties (these "things" can be the actions simultaneously done in the system or some information about the entities of the system). When dealing with access control policies, targets are sets of accesses simultaneously done in the system and we represent accesses as tuples (s, o, a) expressing that a subject $s \in \mathcal{S}$ has an access over an object $o \in \mathcal{O}$ according to an access mode $a \in \mathcal{A} = \{\mathsf{read}, \mathsf{write}\}$. Hence, $\mathbb{A}$ is the powerset of the cartesien product $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$. Then, a set $\mathcal{C}$ of security configurations is introduced: configurations correspond to the information needed to characterize secure elements of $\mathbb{A}$ according to the policy. For access control policies, a configuration can be a set of granted accesses (for the HRU policy), a lattice of security levels (for a MultiLevel security policy), a hierarchy of roles (for the RBAC policy), etc. Last, a policy is defined by a relation $\Vdash$ allowing to characterize secure targets according to configurations.

Definition 1. *A security policy $\mathbb{P}$ is a tuple $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ where $\mathbb{A}$ is a set of security targets, $\mathcal{C}$ is a set of security configurations and $\Vdash \subseteq \mathcal{C} \times \mathbb{A}$ is a relation specifying secure targets according to configurations. An access control policy is a security policy such that $\mathbb{A}$ is the powerset of the cartesien product $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$.*

Example 1. We consider here the HRU policy $\mathbb{P}_H = (\mathbb{A}, \mathcal{C}_H, \Vdash_{\mathbb{P}_H})$, introduced in [7], which is a discretionary access control policy where $\mathcal{C}_H = \mathbb{A}$ (a configuration $m \in \mathcal{C}_H$ specifies a set of granted accesses). Secure targets are thus defined as sets of accesses which are granted: $m \Vdash_{\mathbb{P}_H} A$ iff $A \subseteq m$. In all this paper, we illustrate some of our definitions by considering the HRU policy applied with the configuration m defined by the access control matrix (1). We do not consider here the administrative part of this model allowing to change authorized accesses.

Information Flows. We formalize here information flows occurring during the lifetime of a system on which an access control policy applies.

Transition systems. We represent a system on which a policy $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ applies by a transition system whose states belongs to a set Σ. A state σ contains both the description of the security configuration of the policy used, written $\Upsilon(\sigma)$, and the set of current accesses done in the system, written $\Lambda(\sigma)$. Given a configuration $c \in \mathcal{C}$, we write

$$\Sigma_{|c} = \{\sigma \in \Sigma \mid \Upsilon(\sigma) = c \wedge c \Vdash \Lambda(\sigma)\}$$

the set of secure states according to c. We consider the language of requests $\mathcal{R} = \{\langle +, s, o, a\rangle, \langle -, s, o, a\rangle\}$, providing to the users a way to access objects: $\langle +, s, o, a\rangle$ (resp. $\langle -, s, o, a\rangle$) means that the subject s asks to get (resp. to release) an access over the object o according to the access mode a. Then, transitions are defined by a transition function $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ (where $\mathcal{D} = \{\mathsf{yes}, \mathsf{no}\}$ are the answers given according to the policy). We assume here that this transition function is correct according to the policy and does not change security configurations, and that when the answer given is no the state is not modified:

$$\begin{array}{l}\forall \sigma_1, \sigma_2 \in \Sigma \; \forall R \in \mathcal{R} \; \forall d \in \mathcal{D} \\ \quad (\Upsilon(\sigma_1) \Vdash \Lambda(\sigma_1) \wedge \tau(R, \sigma_1) = (d, \sigma_2)) \Rightarrow \Upsilon(\sigma_2) \Vdash \Lambda(\sigma_2) \\ \forall \sigma_1, \sigma_2 \in \Sigma \; \forall R \in \mathcal{R} \quad \tau(R, \sigma_1) = (\mathsf{no}, \sigma_2) \Rightarrow \sigma_1 = \sigma_2\end{array}$$

We write $\sigma' = \sigma \oplus (s, o, a)$ (resp. $\sigma' = \sigma \ominus (s, o, a)$) the state obtained from σ by adding (resp. removing) the access (s, o, a) to its current accesses without changing security configuration: $\Lambda(\sigma') = \Lambda(\sigma) \cup \{(s, o, a)\}$ (resp. $\Lambda(\sigma') = \Lambda(\sigma) \backslash \{(s, o, a)\}$) and $\Upsilon(\sigma) = \Upsilon(\sigma')$. We assume that the transition function is correct according to this "semantics":

$$\begin{array}{l}\forall \sigma_1, \sigma_2 \in \Sigma \; \tau(\langle +, s, o, a\rangle, \sigma_1) = (\mathsf{yes}, \sigma_2) \Rightarrow \sigma_2 = \sigma_1 \oplus (s, o, a) \\ \forall \sigma_1, \sigma_2 \in \Sigma \; \tau(\langle -, s, o, a\rangle, \sigma_1) = (\mathsf{yes}, \sigma_2) \Rightarrow \sigma_2 = \sigma_1 \ominus (s, o, a)\end{array}$$

Given a set $\Sigma_I \subseteq \Sigma$ of initial secure states, we define executions of τ as follows:

$$Exec(\tau, \Sigma_I) = \bigcup_{n \in \mathbb{N}} \left\{ \begin{array}{l} (\sigma_1, \cdots, \sigma_n) \mid \sigma_1 \in \Sigma_I \\ \wedge \; \forall i \; (1 \leq i \leq n-1) \; \exists R \in \mathcal{R} \; \tau(R, \sigma_i) = (\mathsf{yes}, \sigma_{i+1}) \end{array} \right\}$$

Flows generated by sequences of states. During the lifetime of a system, transitions generate information flows. We introduce flow relations between entities of the system allowing to define flows generated by sequences of states (corresponding to executions of transition functions). We define subsets of $\overset{\mathsf{oo}}{\hookrightarrow} = \mathcal{O} \times \mathcal{O}$, $\overset{\mathsf{os}}{\hookrightarrow} = \mathcal{O} \times \mathcal{S}$ and $\overset{\mathsf{so}}{\hookrightarrow} = \mathcal{S} \times \mathcal{O}$ characterizing flows occurring in several contexts. An elementary flow of the information contained into an object o_1 to an object o_2 can occur iff there exists a subject s reading o_1 and writing into o_2.

Definition 2. *Elementary flows generated by a set of accesses A are defined by:*

$$\mapsto_A = \left\{ o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_2 \mid \exists s \in \mathcal{S} \; \{(s, o_1, \mathsf{read}), (s, o_2, \mathsf{write})\} \subseteq A \right\}$$

We can now define the relation $\overset{\mathsf{oo}}{\hookrightarrow}_\sigma \subseteq \overset{\mathsf{oo}}{\hookrightarrow}$ allowing to express that when the system is in a state σ, the information contained in the object o_1 flows into the object o_2 (which is written $o_1 \overset{\mathsf{oo}}{\hookrightarrow}_\sigma o_2$). Such a flow can occur iff there exists a chain of read and write accesses starting from a read access over o_1 and ending at a write access over o_2. The flows between objects generated by the set of current accesses of σ are thus defined as the reflexive and transitive closure of $\mapsto_{\Lambda(\sigma)}$.

Definition 3. *The set of information flows generated by a state σ is denoted by $\overset{\mathsf{oo}}{\hookrightarrow}_\sigma$ and is defined by $\overset{\mathsf{oo}}{\hookrightarrow}_\sigma = \mapsto^*_{\Lambda(\sigma)}$.*

Hence, $o_1 \overset{\mathsf{oo}}{\hookrightarrow}_\sigma o_2$ iff $o_1 = o_2$ or $\exists s_1, \cdots, s_k, s_{k+1} \in \mathcal{S},\ \exists o'_1, \cdots, o'_k \in \mathcal{O}$ such that:

$$\left\{ \begin{array}{l} (s_1, o_1, \mathsf{read}), (s_1, o'_1, \mathsf{write}), (s_2, o'_1, \mathsf{read}), (s_2, o'_2, \mathsf{write}), \cdots, \\ (s_i, o'_{i-1}, \mathsf{read}), (s_i, o'_i, \mathsf{write}), \cdots, (s_{k+1}, o'_k, \mathsf{read}), (s_{k+1}, o_2, \mathsf{write}) \end{array} \right\} \subseteq \Lambda(\sigma)$$

We extend this definition for a set $E \subseteq \Sigma$ of states: $\overset{\mathsf{oo}}{\hookrightarrow}_E = \bigcup_{\sigma \in E} \overset{\mathsf{oo}}{\hookrightarrow}_\sigma$.

Example 2. By considering the HRU policy and the configuration m defined by (1), the set of flows generated by secure states is:

$$\overset{\mathsf{oo}}{\hookrightarrow}_{\Sigma_{|m}} = \left\{ \begin{array}{l} o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_1,\ o_2 \overset{\mathsf{oo}}{\hookrightarrow} o_2,\ o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_3,\ o_4 \overset{\mathsf{oo}}{\hookrightarrow} o_4,\ o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_1,\ o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_2, \\ o_2 \overset{\mathsf{oo}}{\hookrightarrow} o_4,\ o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_4,\ o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_2,\ o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_4 \end{array} \right\}$$

Notice that we assume here the "worst" case: we suppose that if there is a potential for information flow then the flow actually occurs. However, it is possible to refine this definition by observing flows at the OS level. From definition 3, we can now define the relations characterizing flows generated by sequences of states.

Definition 4. *Let $E \subseteq \Sigma$, and $F \subseteq E^\star$ be a set of sequences of states in E. A sequence $(\sigma_1, \cdots, \sigma_n) \in F$ generates several flows defined as follows.*

1. *Flows between objects are defined by composition as follows:*

$$\overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_1, \cdots, \sigma_n)} = \left\{ \begin{array}{ll} \overset{\mathsf{oo}}{\hookrightarrow}_{\sigma_1} & \textit{if } n = 1 \\ \overset{\mathsf{oo}}{\hookrightarrow}_{\sigma_{k+1}} \circ \overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_1, \cdots, \sigma_k)} & \textit{if } n = k+1 \end{array} \right.$$

2. *Flows from objects to subjects characterize which subject can read (in a direct or indirect way) which information initially contained into an object:*

$$\overset{\mathsf{os}}{\hookrightarrow}_{(\sigma_1, \cdots, \sigma_n)} = \bigcup_{i=1}^{n} \left\{ o_1 \overset{\mathsf{os}}{\hookrightarrow} s \mid o_1 \overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_1, \cdots, \sigma_i)} o_2 \wedge\ (s, o_2, \mathsf{read}) \in \Lambda(\sigma_i) \right\}$$

3. *Flows from subjects to objects characterize which subject can write (in a direct or indirect way) into which object:*

$$\overset{\mathsf{so}}{\hookrightarrow}_{(\sigma_1, \cdots, \sigma_n)} = \bigcup_{i=1}^{n} \left\{ s \overset{\mathsf{so}}{\hookrightarrow} o_2 \mid (s, o_1, \mathsf{write}) \in \Lambda(\sigma_i) \wedge\ o_1 \overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_i, \sigma_{i+1}, \cdots, \sigma_n)} o_2 \right\}$$

We extend these definitions as follows: $\overset{X}{\hookrightarrow}_Y = \bigcup_{y \in Y} \overset{X}{\hookrightarrow}_y$ where $X \in \{\mathsf{oo}, \mathsf{os}, \mathsf{so}\}$ and $Y \in \{E, F\}$.

Expressing flows in a uniform way. It is possible to express all flows between subjects and objects as flows between objects. To achieve this goal, each subject $s \in \mathcal{S}$ is associated with an object $o_s \in \mathcal{O}_\mathcal{S}$, where $\mathcal{O}_\mathcal{S}$ is the set of objects associated with subjects. Of course, objects occurring in the definition of the access control policy are only those belonging to $\mathcal{O}$. Furthermore, by following such an approach, for all state σ and all subject s, we have:

$$\{(s, o_s, \mathsf{read}), (s, o_s, \mathsf{write})\} \subseteq \Lambda(\sigma) \text{ and } ((s, o, a) \in \Lambda(\sigma) \wedge o \in \mathcal{O}_\mathcal{S}) \Rightarrow o = o_s$$

Of course, for a sequence $(\sigma_1, \dots, \sigma_n)$ of states, we have:

$$\overset{\mathsf{os}}{\hookrightarrow}_{(\sigma_1,\dots,\sigma_n)} = \{o_1 \overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_1,\dots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O} \wedge o_2 \in \mathcal{O}_\mathcal{S}\}$$
$$\overset{\mathsf{so}}{\hookrightarrow}_{(\sigma_1,\dots,\sigma_n)} = \{o_1 \overset{\mathsf{oo}}{\hookrightarrow}_{(\sigma_1,\dots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O}_\mathcal{S} \wedge o_2 \in \mathcal{O}\}$$

3 Flow Policies

We consider now access control over information contained into the objects of the system. This leads to view access control policies as information flows policies. A confidentiality policy defines which information can be accessed per users. This can be expressed in terms of flows from objects to subjects: a confidentiality policy can be expressed by a subset $\overset{\mathsf{os}}{\rightsquigarrow}$ of $\overset{\mathsf{os}}{\hookrightarrow}$. An integrity policy defines who is allowed to modify a container of information and can be expressed by a subset $\overset{\mathsf{so}}{\rightsquigarrow}$ of $\overset{\mathsf{so}}{\hookrightarrow}$. A confinment policy defines which information contained into an object can flow into another object and can be expressed by a subset $\overset{\mathsf{oo}}{\rightsquigarrow}$ of $\overset{\mathsf{oo}}{\hookrightarrow}$.

Access Control over Information. An access control policy can be used to ensure confinment and/or confidentiality and/or integrity in an information system. Such policies, expressed in terms of information flows, can be defined from the access control policy as follows.

Definition 5. *Flow policies over Σ associated with a configuration $c \in \mathcal{C}$ of an access control policy $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ are defined as follows:*

$$\begin{aligned}
\textit{Confidentiality Policy:}\ & \overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}[c]} = \{o \overset{\mathsf{os}}{\hookrightarrow} s \mid \exists \sigma \in \Sigma_{|c}\ (s, o, \mathsf{read}) \in \Lambda(\sigma)\} \\
\textit{Integrity Policy:}\ & \overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}[c]} = \{s \overset{\mathsf{so}}{\hookrightarrow} o \mid \exists \sigma \in \Sigma_{|c}\ (s, o, \mathsf{write}) \in \Lambda(\sigma)\} \\
\textit{Confinment Policy:}\ & \overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[c]} = \left\{ \begin{array}{l} o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_2 \mid o_1 = o_2\ \vee\ \exists \sigma \in \Sigma_{|c}\, \exists s \in S \\ (s, o_1, \mathsf{read}) \in \Lambda(\sigma), (s, o_2, \mathsf{write}) \in \Lambda(\sigma) \end{array} \right\}
\end{aligned}$$

The whole flow policy induced by the access control is composed by the confidentiality, integrity and confinment policies. Using the expression of flows in a uniform way as defined in page 77, it is represented by:

$$\rightsquigarrow_{\mathbb{P}[c]} = \left\{ \begin{array}{ll} o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_2 \mid o_1 = o_2 & \\ \vee(\exists s \in S\ \exists \sigma \in \Sigma_{|c} & o_2 = o_s \in O_s \wedge (s, o_1, \mathsf{read}) \in \Lambda(\sigma)) \\ \vee(\exists s \in S\ \exists \sigma \in \Sigma_{|c} & o_1 = o_s \in O_s \wedge (s, o_2, \mathsf{write}) \in \Lambda(\sigma)) \\ \vee(\exists s \in S\ \exists \sigma \in \Sigma_{|c} & (s, o_1, \mathsf{read}), (s, o_2, \mathsf{write}) \in \Lambda(\sigma)) \end{array} \right\}$$

Example 3. If we consider the configuration m defined in (1) for HRU, we have:

$$\overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]} = \left\{ o_1 \overset{\mathsf{os}}{\hookrightarrow} \text{Alice}, o_3 \overset{\mathsf{os}}{\hookrightarrow} \text{Alice}, o_1 \overset{\mathsf{os}}{\hookrightarrow} \text{Bob}, o_2 \overset{\mathsf{os}}{\hookrightarrow} \text{Bob}, o_2 \overset{\mathsf{os}}{\hookrightarrow} \text{Charlie} \right\}$$
$$\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]} = \left\{ \text{Alice} \overset{\mathsf{so}}{\hookrightarrow} o_1, \text{Bob} \overset{\mathsf{so}}{\hookrightarrow} o_2, \text{Charlie} \overset{\mathsf{so}}{\hookrightarrow} o_2, \text{Charlie} \overset{\mathsf{so}}{\hookrightarrow} o_4 \right\}$$
$$\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} = \left\{ o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_1, o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_1, o_1 \overset{\mathsf{oo}}{\hookrightarrow} o_2, o_2 \overset{\mathsf{oo}}{\hookrightarrow} o_2, o_2 \overset{\mathsf{oo}}{\hookrightarrow} o_4, o_3 \overset{\mathsf{oo}}{\hookrightarrow} o_3, o_4 \overset{\mathsf{oo}}{\hookrightarrow} o_4 \right\}$$

Access Control over Objects *Versus* **over Information.** We can now define in a formal way the properties expressing that a set of flows (occurring for a set of states E or occurring during sequences of states in a set F) is consistent according to the flow policies obtained from the access control policy.

Definition 6. *Let $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ be an access control policy, $c \in \mathcal{C}$, and X be a set of states or a set of sequences of states. X is said to be consistent according to:*

- *the confidentiality policy $\overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\mathsf{os}}{\hookrightarrow}_X \subseteq \overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$,*
- *the integrity policy $\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\mathsf{so}}{\hookrightarrow}_X \subseteq \overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$,*
- *the confinment policy $\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\mathsf{oo}}{\hookrightarrow}_X \subseteq \overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$,*
- *the information flow policy $\rightsquigarrow_{\mathbb{P}[c]}$ iff X is consistent according to $\overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$, $\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$ and $\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$.*

These properties are useful to analyze flows generated by secure states of access control policies ($X = \Sigma_{|m}$) or flows generated by executions ($X = Exec(\tau, \Sigma_I)$).

Example 4. For HRU, there exists configurations m for which the sets of flows occurring during sequences in $Exec(\tau_H, \Sigma_H^I)$ are neither consistent according to $\overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ nor to $\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ nor to $\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$. Indeed, if we consider the configuration m defined in (1), we have:

$$o_3 \overset{\mathsf{os}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)} \text{Bob},\ \text{Bob} \overset{\mathsf{so}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)} o_4,\ o_3 \overset{\mathsf{oo}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)} o_2$$

but we don't have:

$$o_3 \overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]} \text{Bob},\ \text{Bob} \overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]} o_4,\ o_3 \overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} o_2$$

Of course, when m satisfies some "good" properties (transitivity), the consistency properties hold. More precisely, we have proved that:

1. $\overset{\mathsf{os}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)}$ is consistent according to $\overset{\mathsf{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff:

$$\forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O} \quad (o_1 \overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[m]} o_2 \wedge (s, o_2, \mathsf{read}) \in m) \Rightarrow (s, o_1, \mathsf{read}) \in m$$

2. $\overset{\mathsf{so}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)}$ is consistent according to $\overset{\mathsf{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff:

$$\forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O} \quad (o_1 \overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}[m]} o_2 \wedge (s, o_1, \mathsf{write}) \in m) \Rightarrow (s, o_2, \mathsf{write}) \in m$$

3. $\overset{\mathsf{oo}}{\hookrightarrow}_{Exec(\tau_H,\Sigma_H^I)}$ is consistent according to $\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff $\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} = (\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]})^*$ (i.e. $\overset{\mathsf{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ and its reflexive and transitive closure define the same relation)

This framework has been successfully used to analyse flow properties of several classical access control policies: the Bell & LaPadula policy [1] which has been (not surprisingly) proved consistent, the Chinese Wall model [2] which has been proved consistent according to the confidentiality and integrity flow policies, and the role-based access control model (RBAC) [5,16] which is not consistent.

4 Detecting Illegal Information Flows

As we said, access control mechanisms are specified in terms of granted accesses but don't always make explicit the information flows that they are supposed to permit. In fact, the notion of granted flows depends on the interpretation of the access control model. As shown in the above section, the flow based interpretation of an access control policy may forbid flows that can occur when applying a classical access control mechanism. We introduce here a flow detection mechanism allowing to deal with such problems. We define what a flow detection mechanism consists in and what properties we can express over such mechanism. Then, in this setting, we provide theoretical foundations for the intrusion detection systems introduced in [6,8,18,19], by formalizing it as a detection flow mechanism together with proofs of its soundness and completeness. Last, we show how such mechanism can be used in practice for the HRU model.

Specification of Flow Detection Mechanisms. We show here how to specify mechanisms allowing to detect illegal information flows according to a flow policy $\rightsquigarrow$ during the lifetime of an information system. Hence, we introduce the formal specification of flow detection mechanisms allowing to monitor an information system in order to detect flows occurring in a set $\rightsquigarrow_{\mathbb{F}}$ of forbidden flows during the lifetime of the system. Such mechanisms are based on the definition of a predicate $\mho$ over states characterizing states occurring in a sequence of states generating at least a flow in $\rightsquigarrow_{\mathbb{F}}$. A state σ verifying $\mho(\sigma)$ is called an alert state. The set of observable sequences of states can be the set of executions of an operational mechanism of an access control policy.

Definition 7. *A flow detection mechanism parameterized by a set E of observable states, a set $F \subseteq E^\star$ of observable sequences of states and a set $\rightsquigarrow_{\mathbb{F}}$ of flows is denoted by $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ and is defined by $(\Sigma, \mho)$ where Σ is the set of states of the system and $\mho$ is a predicate characterizing alert states.*

We also introduce the properties over flow detection mechanisms allowing to express that alert states are exactly states occurring in a sequence of states generating at least a flow in $\rightsquigarrow_{\mathbb{F}}$. In the following definition, $\rightsquigarrow_{\mathbb{F}}$ can specify a set of flows between objects ($X = \mathsf{oo}$), from subjects to objects ($X = \mathsf{so}$), or from objects to subjects ($X = \mathsf{os}$).

Definition 8. *Let $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ be a flow detection mechanism.*

– *$\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ is sound iff each state σ verifying $\mho(\sigma)$ is coming from a sequence in F generating at least one flow occurring in $\rightsquigarrow_{\mathbb{F}}$:*

$$\forall(\sigma_1, \ldots, \sigma_n) \in F \;\; \mho(\sigma_n) \Rightarrow \;\; (\overset{X}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset)$$

– *$\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ is complete iff each sequence in F generating a flow occurring in $\rightsquigarrow_{\mathbb{F}}$ ends with an alert state.*

$$\forall(\sigma_1, \ldots, \sigma_n) \in F \;\; (\overset{X}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset) \Rightarrow \mho(\sigma_n)$$

Of course, the parameter $\rightsquigarrow_{\mathbb{F}}$ of a flow detection mechanism $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ used to check that a flow policy $\rightsquigarrow$ is satisfied must be such that:

– granted flows according to $\rightsquigarrow$ are not detected by the detection mechanism:

$$\rightsquigarrow_{\mathbb{F}} \cap \rightsquigarrow = \emptyset \tag{2}$$

– flows that may occur and that are not authorized by $\rightsquigarrow$ are detected:

$$(\overset{X}{\hookrightarrow}_F \setminus \rightsquigarrow) \subseteq \rightsquigarrow_{\mathbb{F}} \tag{3}$$

Thanks to these properties, for a sound and complete flow detection mechanism $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$, alert states are exactly states coming from a sequence generating at least one flow that do not respect the flow policy $\rightsquigarrow$.

Definition of a Flow Detection Mechanism. We use the uniform representation of flows introduced page 77 and define a flow detection mechanism allowing to detect flows occurring during sequences of states that do no satisfy a reflexive information flow policy $\rightsquigarrow$. We use the relation $\twoheadrightarrow \subseteq \mathcal{P}(\mathcal{O}) \times \mathcal{O}$ such that $\{o_1, \cdots, o_n\} \twoheadrightarrow o$ expresses that information initially contained in $o_1, \ldots, o_n$ are allowed to flow separately or together into o. The relation $\twoheadrightarrow$ is defined as follow:

$$\twoheadrightarrow = \bigcup_{o \in \mathcal{O}} (\{o_i | o_i \rightsquigarrow o\}, o)$$

Observable sequences of states. The flow detection mechanism we define here is generic: it is parameterised by a set E of observable states and by a set $F \subseteq E^\star$ of observable sequences of states such that for all sequences $(\sigma_1, \cdots, \sigma_n) \in F$, the initial state σ_1 has an empty set of current accesses ($\Lambda(\sigma_1) = \emptyset$), and such that the state σ_{i+1} is obtained from σ_i either by adding or by removing an access:

$$\forall (\sigma_1, \cdots, \sigma_n) \in F \ \forall i \quad \sigma_{i+1} = \sigma_i \oplus (s, o, a) \vee \sigma_{i+1} = \sigma_i \ominus (s, o, a)$$

Definition of the set of forbidden information flows $\rightsquigarrow_{\mathbb{F}}$. Since we want to detect flows that do not satisfy the policy $\rightsquigarrow$, we define the set of forbidden information flows $\rightsquigarrow_{\mathbb{F}}$ as the set of all possible information flows except those in the policy:

$$\rightsquigarrow_{\mathbb{F}} = \overset{\mathsf{OO}}{\hookrightarrow}_F \setminus \rightsquigarrow$$

Definition of alert states. The flow detection mechanism we define here is a tagging system for which we prove that it permits a sound and complete detection of illegal information flows defined in $\rightsquigarrow_{\mathbb{F}}$. For a state σ, and for each object $o \in \mathcal{O}$, we define a reading tag $T_\sigma^R(o)$ and a writing tag $T_\sigma^W(o)$. The reading tag denotes the policy related to the information really contained in the object. The writing tag denotes the policy related to the object viewed as a container of information. These tags are defined as follows:

– For all sequence $(\sigma_1, \cdots, \sigma_n) \in F$ and for all $o \in \mathcal{O}$, the tags are initially defined for σ_1 as follows. The tag $T_{\sigma_1}^R(o)$ attached to o denotes the part of the flow policy $\twoheadrightarrow$ related to the information initially contained in o:

$$T_{\sigma_1}^R(o) = \{(\mathsf{O}, o') \in \twoheadrightarrow | \, o \in \mathsf{O}\}$$

The tag $T^W_{\sigma_1}(o)$ attached to o denotes the part of the information flow policy $\twoheadrightarrow$ related to o where o is viewed as a container of information:

$$T^W_{\sigma_1}(o) = \{(\mathsf{O}, o') \in \twoheadrightarrow | \, o = o'\}$$

– At each transition from σ_i to σ_{i+1}, the writing tags don't change ($T^W_{\sigma_i}(o) = T^W_{\sigma_{i+1}}(o)$) and the reading tags of objects evolve in the following way:

$$T^R_{\sigma_{i+1}}(o) = \bigcap_{\{o_j \in \mathcal{O} | o_j \overset{\circ\circ}{\hookrightarrow}_{\sigma_{i+1}} o\}} T^R_{\sigma_i}(o_j)$$

Notice that if o has not been modified by a flow then $\{o_j \in \mathcal{O} | o_j \overset{\circ\circ}{\hookrightarrow}_{\sigma_{i+1}} o\}$ is simply $\{o\}$ (see definition 3) and the tag $T^R_{\sigma_{i+1}}(o)$ is exactly $T^R_{\sigma_i}(o)$.

The following lemma (proved by induction over n) states that $T^R_\sigma(o)$ exactly denotes the part of the policy $\twoheadrightarrow$ shared by all information contained in o.

Lemma 1. $\forall (\sigma_1, \sigma_2, \ldots, \sigma_n) \in F \ \forall o \in \mathcal{O}$

$$T^R_{\sigma_n}(o) = \{(\mathsf{O}, o') \in \twoheadrightarrow | \, \forall o_i \in \mathcal{O} \ o_i \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} o \Rightarrow o_i \in \mathsf{O}\}$$

We are now in position to define the predicate $\mho$ as follows:

$$\mho(\sigma) \Leftrightarrow \exists o \in \mathcal{O} \ T^R_\sigma(o) \cap T^W_\sigma(o) = \emptyset$$

which allows to characterize alert states and leads to define a sound and complete flow detection mechanism (definition 8) as shown by the following lemma.

Lemma 2. $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ *is both sound and complete.*

Proof. The proof is done by contraposition, we prove that in a sequence of states $(\sigma_1, \cdots, \sigma_n) \in F$ no illegal information flows has occurred if and only if for any object o the intersection between read and write tags is non-empty in σ_n (e.g. σ_n is not an alert state):

$$\forall (\sigma_1, \cdots, \sigma_n) \in F \ \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} = \emptyset \Leftrightarrow \forall o \in \mathcal{O} \quad T^R_{\sigma_n}(o) \cap T^W_{\sigma_n}(o) \neq \emptyset$$

Let $(\sigma_1, \cdots, \sigma_n) \in F$. According to (2) and (3) and the definition of $\rightsquigarrow_{\mathbb{F}}$, no illegal flow has occured during $(\sigma_1, \cdots, \sigma_n)$ iff only allowed flows has occured:

$$\begin{aligned}
& \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} = \emptyset \\
\Leftrightarrow & \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} \subseteq \rightsquigarrow \\
\Leftrightarrow & \forall o \in \mathcal{O} \ \forall o' \in \mathcal{O} \, o' \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} o \Rightarrow o' \rightsquigarrow o \\
\Leftrightarrow & \forall o \in \mathcal{O} \ \forall o' \in \mathcal{O} o' \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} o \Rightarrow \exists (O, o) \in \twoheadrightarrow, \ o' \in O \text{ (by def. of } \twoheadrightarrow) \\
\Leftrightarrow & \forall o \in \mathcal{O} \left\{ (O, o) \in \twoheadrightarrow | \forall o' \in \mathcal{O} \, o' \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} o \Rightarrow o' \in O \right\} \neq \emptyset
\end{aligned}$$

Hence, by lemma 1 and by definition of $T^W_{\sigma_n}(o)$, we have:

$$\forall o \in \mathcal{O} \ T^R_{\sigma_n}(o) \cap T^W_{\sigma_n}(o) = \left\{ (O, o) \in \twoheadrightarrow | \forall o_j \in \mathcal{O} \, o_j \overset{\circ\circ}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} o \Rightarrow o_j \in O \right\}$$

which allows to conclude, by definition. $\triangleleft$

Application. We describe now how to use our flow detection mechanism to detect illegal flows (w.r.t. $\rightsquigarrow_{\mathbb{P}_H[m]}$) generated by executions of $Exec(\tau_H, \Sigma_H^I)$. Parameters are instanciated as follows: E is the set $\Sigma_{|m}$ of secure states and F is the set $Exec(\tau_H, \Sigma_H^I)$ where $\Sigma_H^I \subseteq \{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$ and $\rightsquigarrow$ is the relation $\rightsquigarrow_{\mathbb{P}_H[m]}$. According to lemma 2, alert states are exactly states coming from sequences of states generating flows that do not satisfy $\rightsquigarrow_{\mathbb{P}_H[m]}$.

Example 5. Let us consider again the HRU policy applied with the configuration m defined in (1). This policy induces an information flow policy $\twoheadrightarrow_H$ defined as follows (the objects o_A, o_B, o_C stand for the objects associated with the subjects Alice, Bob and Charlie):

$$\twoheadrightarrow_H = \left\{ \begin{array}{llll} (\{o_1, o_3, o_A\}, o_1), & (\{o_1, o_2, o_B, \}, o_2), & (\{o_2, o_C\}, o_2), & (\{o_3\}, o_3), \\ (\{o_2, o_4, o_c\}, o_4), & (\{o_1, o_3, o_A\}, o_A), & (\{o_1, o_2, o_B\}, o_B), & (\{o_2, o_C\}, o_C) \end{array} \right\}$$

Let $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ be a sequence of states such that σ_1 has an empty set of current accesses, σ_2 is obtained by adding a read access on o_3 to Alice, σ_3 by adding a write access on o_1 to Alice and σ_4 by adding a read access on o_1 to Bob. None of the states σ_1, σ_2 and σ_3 are alert states, while σ_4 is an alert state since $T^R_{\sigma_4}(o_B) \cap T^W_{\sigma_4}(o_B) = \{(\{o_1, o_3, o_3\}, o_1), (\{o_3\}, o_3), \{o_1, o_3, o_A\}, o_A)\} \cap \{(\{o_1, o_2, o_B\}, o_B)\} = \emptyset$.

5 Implementation

Detecting Illegal Information Flows at the OS Level. A first implementation of the tagging system, called Blare[1], has been realised at the OS level. Blare deduces information flows between typical OS containers (files, sockets or IPC) by observing system calls. Blare is implemented in the Linux kernel, with extra userland tools and associated API library. Figure 1 presents the general architecture of Blare. In order to detect accesses to the OS containers, hooks have been inserted in the kernel to avoid system call overloading. These hooks are principally located in the *Virtual File System* (VFS) layer of the Linux kernel. Linux provides access to containers through the file concept and the VFS is an abstraction of various container types in terms of files. In addition, some containers require operations outside of the scope of the VFS, such as socket connections or virtual consoles. Kernel code implementing these functions is thus also modified. The detection of illegal information flows engine features can be classified into four categories:

1. the *core detection engine* implements the detection algorithm and the tag propagation mechanism;
2. the *configuration management component* allows user space processes to manage the detection engine or to check some information such as the total number of alerts (this is done using the Linux sysfs interface and is useful for debugging or alert diagnosis, but can be locked for security reasons);

[1] Freely available at http://www.rennes.supelec.fr/blare/

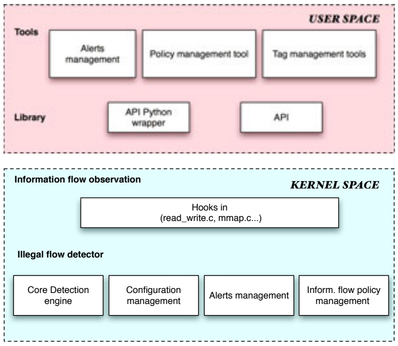


Fig. 1. Architecture of the illegal flow detector at the OS level (Blare)

3. the *information flow management component* allows users to specify a security policy $\twoheadrightarrow$;
4. the *alert management component* increments the alert counter, notifies a userland tool whenever an alert is raised and can optionally and directly log alerts and diagnosis information into a file.

Tags are implemented by a specific structure containing two fixed size bitmap arrays, describing the reading and writing tags. Thanks to this data structure, checking the legality of flows has a small slowdown. In [8], the flow policy $\twoheadrightarrow$ is computed from a free interpretation of access control rights, but we have also proposed to set $\twoheadrightarrow$ manually [18] or to compute it from a MAC policy [6].

Detecting Illegal Information Flows within the JVM. Blare suffers from one major drawback: the semantics of the operations are not taken into account by the detection system. Hence, processes are viewed as black boxes: the reading tags that are modified by a syscall depend on *all* the reading tags of the objects that act as input for this syscall. This approximation is correct, but its coarse-grained nature sometimes leads Blare to trigger false positives. This reason has led to a second implementation, JBlare [8], refining the view of the OS implementation. Information flows are here observed at the language level (Java). The containers of information are variables and object attributes. Flows between these containers are those observed trough method calls or affectation of variables. Java types are either primitive types or reference types and we only consider objects whose types are primitive data types (`int`, `char`, ...). We consider reference data types (such as classes or arrays) as an aggregation of primitive data types. The main goal of JBlare is to refine the computation of reading tags used by Blare. JBlare thus associates a reading tag to any variable whose type is a primitive type. JBlare reading tags are implemented as Java objects. The wrapper class `blareTag` encodes these tags as bitmap structures compatible with the Blare OS implementation. Whenever an OS object is read, corresponding tags are created on the corresponding java object. Tags are updated when a field is accessed or a method is called. Reading or modifying a field involves a flow from the read field to the current method local variables or vice-versa. Method calls are handled according to the method signatures.

A Two-Level Cooperative Detector. Cooperation between JBlare and Blare is achieved by a class on the Java side which detects when a Java program performs some accesses to OS objects such as files or sockets. On the one hand, JBlare language level tags are initialized from Blare OS-level tags. For example when a file is read, the reading tag maintained by Blare is propagated to JBlare through the read security tag of the first instrumented method in the stack frame. On the other hand, JBlare security tags are propagated to Blare tags. For example, a write access to a file implies the propagation from JBlare to Blare of the reading tag related to the method accessing the file.

6 Conclusion

In this paper, we have proposed a formal framework allowing to study different access control and information flow policies. From an access control policy, we have defined three flows policies: a confinment policy which is the set of authorized information flows between objects, a confidentiality policy which is the set of authorized flows from objects to subjects, and an integrity policy which is the set of authorized flows from subjects to objects. Then, we have introduced the consistency property expressing that flows occuring during the life of a system implementing an access control policy are authorized by these flows policies. For example, this property holds for the Bell & LaPadula policy but is not satisfied by the HRU policy. Thus we have elaborated a general mechanism dedicated to illegal information flow detection (security tags propagation and evaluation of a predicate over the tags associated with a state). This mechanism can be used to detect attacks generating illegal information flows in a system (for example a HRU-like system such as Linux), leading actually to an intrusion detection system (IDS) that aims at both completeness and soundness, provided that information flows that violate the policy are detectable and discernible by the IDS. We have presented two implementations of our detection model in two concrete IDSes at two levels of granularity. The first IDS, Blare, is a patch of the Linux kernel and works at the OS level. The second IDS, JBlare, is a Java class that performs bytecode instrumentation on the classes of Java applications; this allows the modified application, once loaded in the JVM, to track the information flow resulting from its own execution. We now plan to extend our formalism to deal with two main features. First, we'd like to add the execute access mode in order to take into account executions of algorithms that can generate flows between its inputs and ouputs. We also aim to consider systems for which the configurations can be modified (under the control of an administrative policy).

References

1. Bell, D., LaPadula, L.: Secure Computer Systems: a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA (May 1973)
2. Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: Proc. IEEE Symposium on Security and Privacy, pp. 206–214 (1989)

3. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19(5), 236–243 (1976)
4. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Communications of the ACM 20(7), 504–513 (1977)
5. Ferraiolo, D.F., Kuhn, D.R.: Role-based access control. In: Proceedings of the 15th National Computer Security Conference (1992)
6. Geller, S., Hauser, C., Tronel, F., Viet Triem Tong, V.: Information flow control for intrusion detection derived from mac policy. In: IEEE International Conference on Communications, ICC 2011 (2011)
7. Harrison, M., Ruzzo, W., Ullman, J.: Protection in operating systems. Communications of the ACM 19, 461–471 (1976)
8. Hiet, G., Viet Triem Tong, V., Mé, L., Morin, B.: Policy-based intrusion detection in web applications by monitoring java information flows. In: 3nd International Conference on Risks and Security of Internet and Systems, CRiSIS 2008 (2008)
9. Jaume, M.: Security Rules *versus* Security Properties. In: Jha, S., Mathuria, A. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 231–245. Springer, Heidelberg (2010)
10. Ko, C., Redmond, T.: Noninterference and intrusion detection. In: IEEE Symposium on Security and Privacy, pp. 177–187 (2002)
11. Myers, A.C.: Jflow: Practical mostly-static information flow control. In: Proceedings of the 26th ACM on Principles of Programming Languages (1999)
12. Myers, A.C., Liskov, B.: Complete safe information flow with decentralized labels. In: IEEE Symposium on Security and Privacy (1998)
13. Myers, A.C., Liskov, B.: A decentralized model for information flow control. SIGOPS Oper. Syst. Rev. 31(5), 129–142 (1997)
14. Osborn, S.L.: Information flow analysis of an RBAC system. In: 7th ACM Symposium on Access Control Models and Technologies SACMAT, pp. 163–168 (2002)
15. Sandhu, R.S.: Lattice-Based Access Control Models. IEEE Computer 26(11), 9–19 (1993)
16. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
17. Schneider, F.B.: Enforceable security policies. Information and System Security 3(1), 30–50 (2000)
18. Viet Triem Tong, V., Clark, A., Mé, L.: Specifying and enforcing a fined-grained information flow policy: Model and experiments. Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications, JOWUA (2010)
19. Zimmermann, J., Mé, L., Bidan, C.: An Improved Reference Flow Control Model for Policy-Based Intrusion Detection. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 291–308. Springer, Heidelberg (2003)

Consistency Policies for Dynamic Information Systems with Declassification Flows

Julien A. Thomas, Frédéric Cuppens, and Nora Cuppens-Boulahia

Télécom Bretagne, LUSSI Department
Université Européenne de Bretagne
Rennes, France

Abstract. Many research work focused on modeling relational database management systems (DBMS) that support multilevel security (MLS) policies. One issue in this context is the inference problem which occurs when it is possible to derive higher classified data from lower classified ones. This corresponds to situations where data is inconsistently classified. Research work that address the inconsistent classification problem generally assume that classification assigned to data is statically defined and does not change over time (the tranquility principle). However, in more recent studies, advanced properties such as secure data declassification were also considered. The main issues addressed in these work are how to extend existing information flow control models, like non interference, to control information flows created by data declassification. But, these work do not consider that dependencies between data may create inconsistent classification problems when data is declassified.

In this paper, we present an approach to consider consistency issues in dynamic information systems with declassifications. Our approach relies on the modeling of explanation graphs associated to both the information system and the declassification flows.

1 Introduction

Consistency is a major property to address when defining a security policy. This property has already been widely studied for access control policies [1,2,3,4,5,6]. Regarding Multilevel Security (MLS) policies, enforcing consistency is also a critical requirement since it may happen that data classified at a low classification level can be used to derive other data classified at a higher one. This is the so-called inference problem in multilevel databases which may be viewed as a special case of inconsistent policy specification. This inference problem has been investigated by many research work before, see for instance [4,5]. In these works, the security policy is said consistent if a user cleared at level L can only infer data classified at most L. The inference system relies on well-known [7] inference principles like deductive and abductive inference channels. However, these previous proposals assume that data is assigned static classifications.

Since [8], several papers investigated declassification requirements [9] using Language-Based security. These work focus on controlling new information flow created by data declassification. In [10], the authors proposed an abstract model to specify declassification operations which consider the *who?*, *what?* and *when?* dimensions suggested in [9]. The authors studied the different declassification operations and distinguished two general types of declassification operations. *Trust based Downgrading* is defined as "a user u is allowed to declassify a piece of information o if u is trusted for the downgrading order on o". Second, *Automatic Downgrading* considers all the operations that do not rely on a downgrading order. This encompasses the time based declassifications and the event based declassifications [11,12]. This work was extended by [13,14] to model declassification rules in dynamic information systems, using Event-Condition-Action (ECA) rules [15]. However, when considering data declassification actions, controlling new information flow created by data declassification is not the only issue we have to address. Another issue previous work do not consider is that dependencies between data may create inconsistent classification problems when some data are declassified. As a consequence, existing proposals related to the inference problem in MLS databases are not robust anymore.

In this paper, we investigate this on-the-fly evaluation of the security policy consistency. In section 2, we first present a scenario to illustrate different consistency problems raised by information declassification. In section 3, we define a multilevel information system with declassification capabilities. In section 4, we present our information flow control model and formalize our consistency property with the notion of dependency graphs. In section 5, we formalize the consistency properties associated with the evaluation of such dependency graphs. We conclude this section with the formalization of consistency policies In section 6, we present related works. Finally, we conclude in section 7.

2 Motivating Scenario

As briefly explained, inconsistent states may be generated by declassification actions. To illustrate this point, consider the scenario presented in figure 1.

We first consider that a secret diplomatic flight denoted *Num* which secretly carries VIP passengers. In the following we shall use logical predicates to represent information. So let us assume that predicate $flight(Num)$ represents the fact that *Num* is a flight and predicate $passenger(Num, Name)$ says that *Name* is a passenger of flight *Num*. Unfortunately, this flight crashed and this information is revealed to the public. Due to this declassification of $crash(Num)$, it is possible to abduciate the existence of secret flight $flight(Num)$ at the public level. So, to maintain classification consistency, we need also to reveal fact $flight(Num)$. Thus, either $flight(Num)$ was declassified before $crash(Num)$ or the declassification of $crash(Num)$ implies the declassification of $flight(Num)$.

Let us then extend this example. We now assume that $crash(Num)$ is actually not declassified and secret predicate $lost(Name)$ is inserted in the database for every passenger of flight *Num*. Let us assume that there may be two possible causes for inserting such a predicate in the database: (1) either *Name* was a passenger of a flight and this flight crashed: $passenger(Num, Name) \wedge crash(Num)$

or (2) *Name* has been kidnapped: *kiddnapped*(*Name*). Depending on the associated security policy, the declassification of *lost*(*Name*) may not trigger the declassification of *crash*(*Num*), if *kiddnapped*(*Name*) is secret, i.e. unknown at the public level. However, if *kiddnapped*(*Name*) is a public predicate and its truth value is false, then one can assume that there is flight which crashed and *Name* was a passenger of this flight. And if *passenger*(*Num*, *Name*) is known at the public level, then *crash*(*Num*) will have to be declassified at the public level.

Finally, let us illustrate a more tricky insecure state which may occur. Let us assume, that it is actually decided to disclose the list of passengers involved in the crash of flight *Num*. Then, rescue operations start and some passengers are rescued but the fact *rescued*(*Name*) is kept secret. After some time, it is decided to end the rescue operations by inserting *endRescue*(*Num*) in the database. Let us now assume that when *endRescue*(*Num*) is inserted, then all the predicates *lost*(*Name*) are declassified to public. If for some passenger *John*, *lost*(*John*) is not declassified at that time, one may infer that this fact has been removed because John was rescued during the rescue operations (if we assume that rescued is the only reason for not being lost). So *rescued*(*John*) should be declassified as a consequence of not declassifying *lost*(*John*).

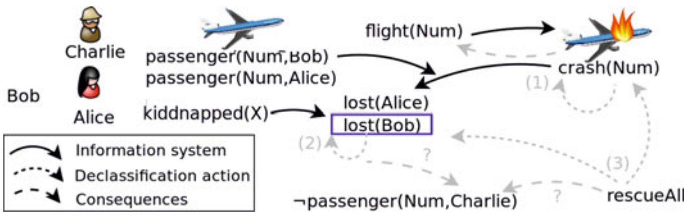


Fig. 1. Declassification Actions and Inconsistency Issues

3 Declassification Flows and Consistency Issues

3.1 Information System Model with Dynamic Behavior

As suggested in section 2, we adopt a logical-based representation of the information system. The information system model consists of active entities (*USERS*), system states (*STATES*) and system operations (*ACTIONS*). Information is modeled using a set of predicates. The system state *state* then corresponds to a set of fully instantiated predicates p_i called facts. We consider a valuation function Π: $(PREDICATES \times STATES) \rightarrow BOOL$. We say that a fact p_i holds in state s if $\Pi(p_i, s) = True$. The basic operations (*ACTIONS*) we consider are the *select*, *delete* and *insert* operations. The *update* operation is modeled as a sequence of a *delete* operation followed by an *insert* operation.

We consider an information system that supports a multilevel security policy. We thus define the *LEVELS* set and the *inf_level*, *equal_level* and *inf_equal_level*

($\sqsubseteq$) security lattices defined on $(LEVELS \times LEVELS) \rightarrow BOOL$. We also define level assignment functions for the database entities. *clearance_level*, defined on $USERS \rightarrow LEVELS$, refers to the clearance level of the active entities while *classification_level* and *integrity_level* relations, defined on $PREDICATES \rightarrow LEVELS$, refer to the classification and the integrity levels of the predicates. These functions respectively assign security levels to the active entities and the objects. Finally, we require the security policy to be secure under the *known policy* constraint, which means that no part of the security policy is secret.

3.2 ECA Rules Security Model

To model dynamic information flows and thus declassification flows, we consider the ECA-rule paradigm defined in [15] and the associated L_{active} language [16] proposed by Baral, Lobo and Trajcevski. As we proposed in [13,14], we more precisely rely on an extended version of the L_{active} language, in order to propose a model able to, on one hand, efficiently model dynamic information systems and, on the other hand, consider declassification policies. This proposal consider both confidentiality (Lc) and integrity (Li) levels.

When considering active rules, the first dimension to consider is the action definition. Action rules are defined as follows, where op_1, ..., op_k is the post condition which corresponds to sequential execution of *select*, *delete* or *insert* operations and $\mathrm{q}_1(\mathrm{X}_1)$, ... $\mathrm{q}_n(\mathrm{X}_n)$ is the conjunctive pre-condition to execute α.

Definition 1 (Multilevel Action). *An action is defined with the do operator on* $ACTIONS \times LEVELS \times LEVELS$ *and is modeled by the following template:* $do(\alpha, L_c, L_i)$ *causes* op_1 ... op_k *if* $q_1(X_1)$ ... $q_n(X_n)$

Second, actions may trigger events. Trigger events are defined as follows, where $\mathrm{r}_1(\mathrm{X}_1)$, ... $\mathrm{r}_n(\mathrm{X}_n)$ is the conjunctive pre-condition for E to occur.

Definition 2 (Multilevel Event). *An event is defined by a name and a header on* $(PREDICATES\times)^{*}LEVELS \times LEVELS$*. It is modeled by the following template:* $event\_name(E, L_1, L_{i1})$ $after$ $do(\alpha, L_2, L_{i2})$ if $r_1(X_1)$ ... $r_m(X_m)$

The occurrence of an event may then in turn initiate the execution of a sequence of new actions. Such rules are defined as follows, where $\mathrm{t}_1(\mathrm{X}_1)$, ... $\mathrm{t}_n(\mathrm{X}_n)$ is the conjunctive pre-condition to execute actions $\alpha_{i=1...j}$, after E.

Definition 3 (Multilevel Active Rule). *An active rule is modeled by the following template:* $rule\_name : event\_name(E, L_0, L_{i0})$ *initiates* $do(\alpha_1, L_1, L_{i1})$ ... $do(\alpha_j, L_j, L_{ij})$ if $t_1(X_1)$ ... $t_p(X_p)$

To consider traditional access control policies which enforce *confidentiality*, we finally assume the information system complies with the three following laws, associated to the first security level of the definitions: the *Access Law* and *Modification Law* respectively control access to and modification of the predicates, the *User Privilege Law* enforces that users send queries at the right security level.

Definition 4 (Access Law). *If p is a predicate and Lc a security level, the Access Law is satisfied w.r.t. (p, Lc) if the classification level of p is lower than or equal to Lc.*

Definition 5 (Modification Law). *If p is a predicate and Lc a security level, the Modification Law is satisfied w.r.t (p, Lc) if Lc is lower than or equal to the classification level of p.*

Definition 6 (User Privilege Law). *If s is a user and Lc is a security level, the User Privilege Law is satisfied w.r.t (s, Lc) if Lc is the clearance level of s.*

3.3 Flow Control Policy and Declassification Policy

Based on these three laws, we must define security properties that respectively control action execution, event occurrence and active rule activation in a multi-level information system. As we showed in [13], the enforcement of these properties is sufficient for confidentiality in dynamic systems without declassification.

Definition 7 (Action Security Property). *The execution of an action by user s at a confidentiality level L satisfies the Action Security Property if (1) for each action condition $q_i(X_i)$, the Access Law is satisfied w.r.t $q_i(X_i)$ and L, and (2) the User Privilege Law is satisfied w.r.t (s, L), (3) the Modification Law w.r.t (p, L) if one op_i operation is an insert of predicate p, (4) the Access Law w.r.t (p, L) if one op_i operation is a select of predicate p and (5) both the Modification and Access Law w.r.t (p, L) if one op_i operation is a delete of predicate p.*

Definition 8 (Event Security Property). *The occurrence of an event at a confidentiality level L satisfies the Event Security Property if (1) for each condition $r_i(X_i)$ of the event, the Access Law is satisfied w.r.t $r_i(X_i)$ and L and (2) the security level L of the triggering action is lower than or equal to L.*

Definition 9 (Active Rule Security Property). *An active rule triggered at level L satisfies the Active Rule Security Property if (1) the security level of the triggering event L is lower than or equal to the execution level of each triggered action and (2) for each condition $t_i(X_i)$ of the active rule, the Access Law is satisfied w.r.t $t_i(X_i)$ and the greatest lower bound $glb(L_1, ..., L_j)$ where $L_1, ..., L_j$ respectively represent the security level of the different triggered actions.*

However, as we shown in [14], flow control policies are no longer sufficient to enforce confidentiality in information systems with declassification flows. Declassification actions must satisfy additional security requirements and in particular integrity requirements. More precisely and in order to assure the trustworthiness of the information system data, as we already mentioned in [10], the security policy must only allow trusted actions. We thus state the following integrity policy which links the trustworthiness of an object to its integrity level.

Definition 10 (Integrity Policy). *Considering a modification (insert, update) request performed with an integrity level li. An object obj may be modified if the integrity Ls of the object satisfies $Ls \sqsubseteq li$*

3.4 Scenario Specification

Let us specify the scenario presented in section 2 using ECA rules. We first specify the *flight_crash* action as follows:

```
do(flight_crash(Num),L,Li) causes insert(crash(Num)) if flight(Num);
do(flight_crash(Num),L,Li) causes insert(lost(Na)) if passenger(Num,Na);
```

First part of this action definition says that the effect of *flight_crash* action will insert predicate $crash(Num)$ if there is a fact $flight(Num)$ in the database and second part says that facts $lost(Name)$ are inserted in the database for every passenger of flight Num. We also define the *kidnapping* action as follows:

```
do(kidnapping(Name),L,Li) causes  insert(lost(Name)) if kidnapped(Name);
```

The declassification action of some predicate to level L is defined as an update from the initial classification level L' of this predicate to level L:

```
do(declassify(P,L),L1,Li) causes
  delete(classification_level(P,L')),insert(classification_level(P,L))
  if classification_level(P,L') ∧ inf_level(L,L');
```

Automatic downgrading is modeled through an active rule which automatically triggers a downgrading operation when some events occur. For this purpose, we first define $declassify\_ev(P, L)$.

```
declassify_rule: declassify_event(P,L,L1,Li1)
  initiates do(declassify(P,L),L2,Li2);
```

Specifying automatic downgrading then consists in specifying when the event $declassify\_event(P, L)$ occurs. For example, in our scenario, we consider that every lost passenger is automatically downgraded by the end of the rescue operations. This requirement is specified by the following *declassify_event* definition:

```
declassify_event(lost(Name),Public,L1,Li1) after
do(insert(endRescue(Flight)),L2,Li2) if passenger(Flight,Name)∧lost(Name)
```

4 Dynamic Information Systems and Consistency

In the previous section, we defined how to model the information systems we consider. However, according to the consistency needs presented in section 2, we must extend existing flow control policy to consider consistency issues associated with declassification actions. In this section, we study how to model consistency.

4.1 Graph Model for the ECA Rules

Modeling Overview. To model the consistency property for dynamic information systems, we rely on graphs to model information flows. In this section, we thus consider the modeling of ECA rules with *multilevel Conditioned Graph*.

Such graphs rely on states which represent the evolution of the information system predicates between the execution of two rules. The possible transitions are determined by the ECA rules and the information states. However, as an impossible transition may be considered as valid by a user who does not know the failing condition value, we rely on the notion of *Valid Conditioned Transition.*

Model. In the ECA paradigm, the information system state evolves due to the execution of actions. The execution of actions may cause the occurrence of events which in turn may trigger the execution of active rules. An active rule then specifies that some sequence of actions must be executed when some event occurs and some associated conditions are satisfied. Based on the execution of actions and active rules, we can thus build a graph of dependencies between pre conditions and post conditions of these executed actions.

First, we assume that the relation between two actions is not reflexive and that no parallel transition is possible. According to the work by Harary and al. [17], the graph of the information system is a directed graph $G = (V, E, f, s)$:

1. V is the subset of predicates which evolve after the triggering of an action,
2. E refers to the allowed transitions between the objects,
3. f (first) links a transition with the source,
4. s (second) links a transition with the destination.

Instead of considering simple transitions, we however rely on a weighted directed graph where the weight w s the associated set of conditions P_S for the transition to occur. Based on L_{active}, we say $P_1 \overset{P_S}{\rightarrow} P_2$ if there exists:

- an action, noted $do(\alpha_1)$, which results in the modification of P_1,
- an event, noted ϵ, associated with action $do(\alpha_1)$,
- a rule, noted *rule*, associated with event ϵ which triggers action $do(\alpha_2)$,
- the action $do(\alpha_2)$ results in the modification of P_2,
- the set P_S refers to the conditions associated with ϵ, *rule* and $do(\alpha_2)$.

Regarding our previous notation, $t_1 = P_1 \overset{P_S}{\rightarrow} P_2$ means that $P_1, P_2 \in V$, $P_S \subseteq V$, $t_1 \in E$, $f(t_1) = P_1$, $s(t_1) = P_2$ and $w(t_1) = P_S$. The figure 2 illustrates the modeling of the ECA rules specified in section 2.

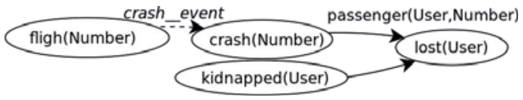


Fig. 2. Example of graph based modeling of a dynamic information system

Second, in multilevel information systems without declassification, data a user may inferred are predetermined as the classification levels are static and no sensitive data is never disclosed. To evaluate the possibility for a transition to be performed, we thus rely on the restricted subset of P_S by L, noted $P_S \lceil L$ and defined as $P \in P_S \lceil L \Leftrightarrow \Pi(classification\_level(P, L_P), s_k) \wedge L_P \sqsubseteq L$, for any state s_k. We then define the concept of valid conditions as follows:

Definition 11 (Valid Conditioned Transition). *Considering the state s_k, a conditioned transition is valid according to L, noted $s_k : P \overset{P_S\lceil L}{\rightarrow} P_2$, if the conditions belonging to the subset $P_S\lceil L$ are valid.*

4.2 Explanation Graphs

Modeling Overview. In the previous section, we formalized the information system rules which a graph G which represent the evolution of the information system predicates and the associated condition for such evolutions. To be able to formalize our consistency property, we need to go further to model explanations that is the set of evolutions which may explain an information state. We thus extend our graph model based analysis which explanation graphs.

Model. To model such explanation graphs, we rely on the graph model. We more precisely define an explanation graphs as a graph G where

1. each node is defined as (i, α_i, s) with i a step number, s the state at step i and α_i the action responsible for the transition to state s. To simplify the notation, we note $\alpha(P)$ the fact that α engenders P
2. E is derived from the transition relation $s_k : P \overset{P_S\lceil L}{\rightarrow} P_2$ as follows: if a transition is defined in E then such a transition is a valid condition transition.

We then distinguish two reasons for an explanation to be inconsistent. First, a state of the graph may be inconsistent as it is possible to infer $p \wedge \neg p$. Second, users may partially know the actions performed on the systems. Thus, the subset of the action visible to a user u must be the same as the one visible to u in the considered explanation G. When an explanation is inconsistent, it must be removed from the set of possible explanations. In figures 3 and 4, we illustrate such an evaluation of explanation graphs, for the mentioned consistency issues. In figure 3, we consider $\neg kidnapped(user)$ as secret. In figure 4, knowing $\neg kidnapped(user)$, the set of possible explanation is reduced.

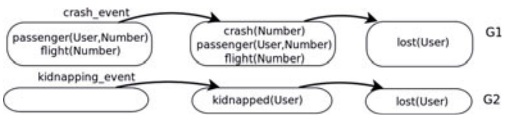


Fig. 3. Explanation graphs (1)

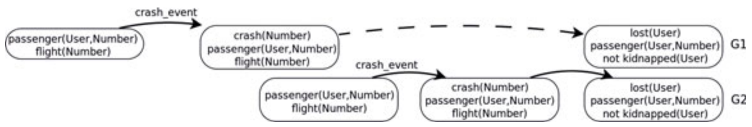


Fig. 4. Explanation graphs (2)

4.3 Consistency Property

In section 2, we showed that sensitive information may be inferred due to information flow based dependencies and declassification actions. Such an issue occurs because the information system security policy does not efficiently manage some information dependencies attackers rely on. More precisely, we illustrated the inconsistency with examples which illustrate the possibility for a user u cleared the security level L_u to infer without doubt information classified L with $L \not\sqsubseteq L_u$.

Based on the Non-Deducibility property [18], we thus say that the system is consistent if for any user u, the state of any information classified L with $L \sqsubseteq L_U$ may be explained by information flows which only rely on information classified L_2 with $L_2 \sqsubseteq L_U$. The evaluation of the consistency property thus consists in building a graph G_{all} that is the intersection of all explanation graphs:

1. each $node_k$ of G_{all} is defined by (k, α_k, s) with k a step number and s the state defined by $v \in s$ iff $\forall(graph\ G).(\exists(\alpha, s_G | (k, \alpha, s_G) \in G \implies v \in s_G))$,
2. the relation E simply links a node (k, α_k, s) to the node $(k+1, \alpha_{k+1}, s')$,
3. if it exists at least a graph G_i such that node $node_k$ is not defined, then $node_k$ is not defined in G_{all},
4. if it exists two graphs G' and G'' such that $\alpha'_k \neq \alpha''_k$, then the value of α_k in G_{all} is undefined.

According to this graph, we evaluate the inferable information as follows: the value of p is inferable in step k if $\exists((\alpha_k, s) | node_k \in G_{all} \wedge node_k = (k, \alpha_k, s) \wedge p \in s)$. In figure 5, we illustrate such an evaluation of the inferable information for the mentioned consistency issues. In the first graph, which is associated to the figure 3, undecidability holds. However, knowing $\neg kidnapped(user)$ (the second graph), it is possible to infer additional information.

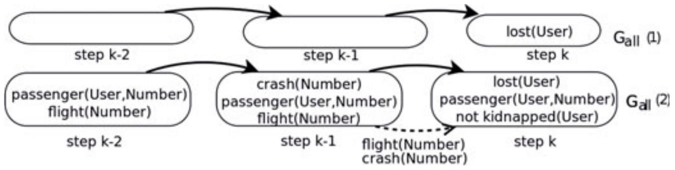


Fig. 5. Union of explanation graphs

4.4 Initial Database Consistency Policy

According to our policies, blind writings are allowed, providing the object integrity is not violated. Consider for instance the following scheme.

```
insert_L2 after do(U,I,obj,L1,Li1)
insert_L2 initiates do(U,I,obj_L2,L1,Li2) if C_L1,LiC
```

Such active rules are allowed according to the policy if $L1 \sqsubseteq L2$ and $Li2 \sqsubseteq lub(Li1, LiC)$ with *lub* the *lowest upper bound*. It is straightforward the policy allows users cleared $L1$ know obj_{L2} Such information flow is permitted as

1. the integrity of obj_{L2} is at most $L1$ ($Li1 \sqsubseteq L1$). Thus any decision taken based on the value of obj_{L2} is trusted at most up to the level $L1$.
2. this does not means a disclosure of obj_{L2} as it may still be modified

When considering declassification flows, this initial consistency policy may be taken into account to prevent some dependencies to be declassified:

Property 1. the declassification of an object O to Ld does not imply the declassification of a child dependency D if $integrity\_level(D) \sqsubseteq Ld$.

5 Dynamic Explanation Graph Evaluations and Consistency Policy

To avoid consistency issues, we proposed in the previous sections graph based models. In this section, we present the derivation rules associated to the construction of the explanation graphs. Due to a lack of space, the proof of the completeness of our proposal is however not given. In the explanation graph based model, we define a state of the graph as (i, α_i, s) with i a step number, s the state of the information system and α_i the action responsible for the transition to s. The step number may however be unknown. We say that a node $node_i$ whose step is unknown is labeled $(\_, \alpha_i, s)$. We consider this issue in section 5.5.

5.1 Invariants on the State Based Knowledge: Initializing a Node N_i

Before presenting our derivation rules, we first state invariants on the revealed sub-state. According to our definition, this sub-state is the information inferable due to either the information level or the information dependencies. In this section, we consider the relation between the system information state (such as the security level) and the revealed state, where $N_i = (i, \alpha_i, s_i)$.

When considering a state s_k, we state that if the classification level of a node is inferior or equal to the considered security level L_D of the graph, then it must be revealed in the state s of the node $N = (k, \alpha_k, s)$ of any explanation. Thus, when creating a new node, we initialize its state based on the classification level of the predicates. Inferences are then possible due to others dependencies dependent of the information system behavior. For instance, when inserting a predicate P, the insertion will occur (and generate events) only if the associated object is non-present in the database. To manage such inference, we state the following invariant for information systems which rely on these assumptions. When considering $N_i = (i, \alpha_i, s_i)$, and α_i being $\alpha_i(P)$, $\neg P$ is added to the state s_j associated to $N_j = (j, \alpha_j, s_j)$ where $j = i + 1$. Finally, considering an action on P, the state of the other predicate is preserved in the parent and child states.

5.2 Explicit Derivation Rules: After a Node N_i Is Created

In the previous subsection, we presented standard inference rules. In this subsection, we present explicit derivation rules.

Explicit Derivation Rule 1: Deductive Inference. When a node $N_i = (i, \alpha_i, s_i)$ is created, the associated action α_i is also revealed. According to the information system, such an action may trigger others, in return. To consider these dependencies and whenever a node is created, the associated child nodes must be added to the graph and the state of each condition is revealed in the parent state. We formalize this inference as follows. For each valid transition $\alpha_i \overset{P_S \lceil L_D}{\rightarrow} \alpha_x(P)$ of the graph model, the current graph is forked. In a first graph, the transition is applied as follows: if a node $N_j = (j, \alpha_j, s_j)$ exists such that $\alpha_j = \alpha_x$ and N_j is a child of N_i ($N_i \rightarrow N_j$), N_j is considered. Otherwise, N_j is created with an unknown index and such that $N_i \rightarrow N_j$. Besides, we say that P and any $P_c \in P_S$ are known in s_i. As the transition may not occur if $P_S \lceil L_D \sqsubset P_S$, nothing is updated in the second graph. Otherwise, the second graph is removed. The other transitions are performed on the remaining graphs.

Explicit Derivation Rule 2: Deductive Inference 2. In subsection 2, we stated that implicit information flows due to failing conditions must also be considered. To model this issue, we need to consider the non-updated graph defined by the Explicit Derivation Rule 1:

1. according to our statements in section 5.1, P may already be set in s_i. So, we create a graph were $P \in s_i$
2. a condition P_c in $P_S - P_S \lceil L_D$ is false. For each possible P_c, we create a graph where $\neg P_c \in s_i$

Explicit Derivation Rule 3: Abductive Inference. Third, when the associated action α_i is revealed, an abductive inference is possible. Such inference consists in considering any possible explanation for α_i. According to our consistency policy, we thus generate a graph for each possible explanation. We formalize this abductive inference based derivation as follows. If a node $N_j = (j, \alpha_j, s_j)$ exists such that $N_j \rightarrow N_i$, we consider the node N_j as the parent action. Otherwise, for each relation $s_{i-1} : \alpha_j \overset{P_S \lceil L_D}{\rightarrow} \alpha_i$, we create a new graph G with a non-indexed node N_j such that $N_j \rightarrow N_i$. Besides, we say that any $P_c \in P_S$ and $\neg P$ are known in s_{i-1}. The external insertion is modeled by the not updated first graph.

Explicit Derivation Rule 4: Abductive Inference 2. Finally, implicit information flows due to failing conditions must also be considered. To model this issue, we need to add another explanation: an external insertion due to failing conditions. When considering each relation $\alpha_j \overset{P_S}{\rightarrow} \alpha_i$, for each condition $P_{fc} \in P_S - P_S \lceil L_D$, we create a graph where P_{fc} is a failing condition.

5.3 Explicit Derivation Rules: After a Predicate P Is Defined in s_i

In the previous subsection, we presented consistency rules associated to the insertion of a new node in the graph. Such rules considered the dynamic dimension of the information systems and relied on both abductive and deductive inference rules. However, when considering the dynamic dimension of the information, we must also considered inference based on the update of a predicate. Indeed, such a predicate as been inserted (or deleted) by a previous action. Thus, when a predicate is updated, the possible explanations for its state must be considered too. We thus state the following rules. Consider $Node_i = (i, \alpha_i, s_i)$. If $\alpha_i(P)$, the predicate is already considered by the previously stated rules. Otherwise,

1. if($P \in s_{i-1}$), the predicate state is explained in a previous state.
2. if($\neg P \in s_{i-1}$), the system state is inconsistent and the graph is thus removed
3. otherwise, if α_i is defined, we add P to s_{i-1}.
4. otherwise, if α_i is not set, the graph is forked and for the first graph, we add P to s_{i-1}. For the second, we set $\alpha_i(P)$

5.4 Implicit Derivation Rules: After a Predicate *P* Is defined in s_i

In section 5.1, we stated that the creation of a node implies the preservation of the state of the predicates not considered by the associated action. We thus also needs to consider this preservation when a predicate is revealed after the node is created. To enforce this preservation, we say that if $\alpha_i \neq \alpha_i(P)$ then P is set in s_{i-1} and if $\alpha_{i+1} \neq \alpha_{i+1}(P)$ then P is set in s_{i+1}.

5.5 Index Evaluation

In the previous subsections, we defined algorithms to build the explanation graphs. Some of the index may yet not be determined when the graph is created.

First, consider any rule which results in the insertion of a predicate P in the state s_l. We consider here two possibilities. If $\exists N_l = (l, \alpha_l, s_l)$ then P is effectively added to s_l of N_l. Otherwise, we create a node $N_l = (l, \_, s_l)$. Second, consider the deductive inference (rule 1 and 2 of section 5.2). According to the active rule ordering process, users may infer node indexes. Considering the first applied transition, the user knows $j = i + 1$. Besides, when no deductive rule is applicable to N_i (a leaf of the graph), it is possible to infer the relation between N_j and N_i as $j = i+1$ if starting from N_{pi} where $N_{pi} \rightarrow N_i$, for each parent node, either the current transition is the last possible transition or the next transition is $N_{pi} \rightarrow N_j$. Third, based on the instantiation of the index of $N_i = (i, \alpha_i, s_i)$ and if a second node N_l already exists where $N_l = (i, \alpha_l, s_l)$, then if $\alpha_l = \_$ or $\alpha_l = \alpha_i$, the states are merged. Otherwise, the explanation is inconsistent.

5.6 Consistency Policy for Dynamic Information Systems

In the previous sections, we showed that declassification actions may engender inconsistent states. We thus defined an algorithm for the evaluation of such states. Based on this evaluation, we may define policy that either allow declassification action and implies the management of the associated inconsistency or deny (pro-active) declassification actions when the inconsistency is undesired.

To define our policies, we consider the following sets. *inconsistent_states* refers to any inferable information as $(O, s_k, L) \in inconsistent\_states$ if O is inferable at step k for a user with the clearance level L and such inference is in conflict with the security policy. $declassification(s_n)$ refers to the objects declassified in s_n. We then define $declassification$ as the subset of $inconsistent\_states$ restricted to s_n. Finally, several consistency policies are possible.

Definition 12 (Strict Non Interference). *The action α performed in the state s_{n-1} is authorized if the state s_n satisfies $inconsistent\_states(s_n) = \varnothing$*

Definition 13 (Intransitive Declassification). *The action α performed in the state s_{n-1} is authorized if the state s_n satisfies $declassification(s_n) = \varnothing$*

Definition 14 (Authoritative Declassifications). *The action α performed in the state s_{n-1} is authorized, no matter the declassifications it engenders*

When the consistency policy authorizes the action α, the system must declassify each object P to L if $(P, L) \in declassification(s_n)$. According to these policies, it is obvious that declassification requests may be rejected due to inconsistent states. However, such inconsistency may rely on sensitive data. To avoid the violation of the confidentiality policy [14], we thus state that:

1. Event based declassifications rely on the authoritative declassification policy.
2. Order based declassifications may rely on any consistency policy if the user who order the declassification of o is at least cleared the initial classification level of o. Otherwise the authoritative declassification policy is used.

6 Related Work

Several work [1,2,3,4,6] already considered consistency issues in security policies. In [4], the authors present consistency issues in role based access control policy. They model the concept of norms and considered consistency issues when a specific user may simultaneously be assigned to several roles, even when the subset of norms associated with each role is conflict-free. To manage such conflicts, the authors define derivation algorithms based on role priorities, which thus solve the consistency issues. In our work, we do not assume that our derivation rules are conflict-free (even under instantiation). Thus, to define a complete inference policy, we must define additional constraints to enforce their composability [2]. Two different instantiations of the derivation rules may infer different implicit classification levels. However, we rely on the concept of priority [4] and assume that instantiated derivation rules are given a priority order such that the lower derived implicit classification level is prioritized [5]. In [5], the authors present an inference controller for multilevel databases. Based on inference rules specified in first order logic, the authors formalized the classification level of derived facts. We extended this concept of implicit classification levels using derivation rules for active rule based dependencies. Conflicts are detected when associated invariants are violated. We also refined the conflict resolution algorithms with the concept of *Strict Non Interference and* Intransitive Declassification, provided they do not violate the declassification policy hypotheses [14].

In [6], the authors present conflict resolution algorithms to prevent sensitive data to be inferred. Based on the concept of minimum support and confidence, they define the concept of unsafe association rules and propose algorithm to evaluate reclassification needs. Thus, contrary to our proposal, the authors (only) proposed a reclassification policy. Besides, the authors focused on simple databases. They did not consider functional dependencies. In [7], the author presents several types of inference channels including deductive, abductive and probabilistic channels. In our formalization, our model relies on the Non-Deducibility policy. So, we can manage the deductive and abductive inference channels but probabilistic inference channels are out of the scope this paper. Regarding consistency issues in declassification policies, the work by Sabelfeld and Sands [9] summarizes existing work and issues related to the concept of semantic consistency. According to the authors, a program satisfies this consistency property if the security is preserved under semantic transformation. Thus, declassification policy only consider consistency issues of semantic dependencies [19,20]. According to our study, such consistency policies are not sufficient.

According to these analyses, we argue that existing work on consistency policies are not able to manage consistency issues in dynamic information systems with declassification. The consistency policy must consider semantic dependencies but also execution related dependencies.

7 Conclusion

In this article, we studied the consistency issues for information systems with dynamic behaviors and declassification capabilities. We formally defined a graph based dependency model to handle these issues and define a robust consistency policy. Associated to this model, when then presented inference rules associated to each dependency created by action executions and concluded with the definition our consistency policy. Finally, we compared our work with related ones and show that the issues addressed in this paper were not solved before.

Due to a lack of space, we however did not present the algorithm which enforces these invariants and showed the completeness of our proposal. This algorithm may be presented as a future work. Besides, the consistency issues only considered inferred information based on ECA rules related dependencies, In future work, we may refine our approach to also consider the information dependencies produced by arithmetic operations.

References

1. Moffett, J.D., Sloman, M.S.: Policy conflict analysis in distributed system management (1993)
2. Dinolt, G., Benzinger, L., Yatabe, M.: Combining components and policies, pp. 22–33 (June 1994)
3. Cuppens, F., Saurel, C.: Specifying a security policy: A case study. In: Proc. of the Computer Security Foundations Workshop, pp. 123–134. Kenmare Press (1996)

4. Cholvy, L., Cuppens, F., Belin, A.E.: Analyzing consistency of security policies. In: 1997 IEEE Symposium on Security and Privacy, pp. 103–112. IEEE (1997)
5. Cuppens-Boulahia, N., Cuppens, F.: Inference controller for multilevel databases. In: International Symposium on Programming and Systems, Algiers (May 2001)
6. Li, C., Shirani-Mehr, H., Yang, X.: Protecting Individual Information Against Inference Attacks in Data Publishing. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 422–433. Springer, Heidelberg (2007)
7. Raman, S.: Detecting Inference Attacks Using Association Rules (December 2001)
8. Goguen, J., Meseguer, J.: Security policies and security models. In: 1982 IEEE Symp. Security and Privacy, pp. 11–20. IEEE (1982)
9. Sabelfel, A., Sands, D.: Dimensions and principles of declassification. In: CSFW 2005: Proceedings of the 18th IEEE workshop on Computer Security Foundations, pp. 255–269 (2005) ISBN ISSN:1063-6900 , 0-7695-2340-4
10. Thomas, J., Cuppens-Boulahia, N., Cuppens, F.: Modeling and Controlling Downgrading Operations in Information Systems. In: 5th International Conference on Signal-Image Technology & Internet-based Systems, SITIS 2009 (December 2009)
11. President of the United States: Executive order 12958, classified national security information. Technical report (March 2003)
12. Secrétariat Général de la Défense Nationale: Instruction générale interministérielle sur la protection du secret de la défense nationale (August 2003)
13. Thomas, J., Cuppens-Boulahia, N., Cuppens, F.: Expression and Enforcement of Confidentiality Policy in Active Databases. In: 5th International ACM Conference on Management of Emergent Digital EcoSystems, MEDES 2010, Bangkok, Thailand, LUSSI - Institut Télécom-Télécom Bretagne, October 26-29 (2010)
14. Thomas, J., Cuppens-Boulahia, N., Cuppens, F.: Declassification Policy Management in Dynamic Information Systems. In: The Sixth International Conference on Availability, Reliability and Security, ARES 2011, Vienna, Austria, LUSSI - Institut Télécom-Télécom Bretagne, August 22-26 (2011)
15. Dayal, U., Buchmann, A.P., McCarthy, D.R.: Rules are Objects Too: A Knowledge Model for an Active, Object-Oriented Databasesystem. In: Dittrich, K.R. (ed.) OODBS 1988. LNCS, vol. 334, pp. 129–143. Springer, Heidelberg (1988)
16. Baral, C., Lobo, J., Trajcevski, G.: Formal Characterizations of Active Databases: Part II. In: Bry, F. (ed.) DOOD 1997. LNCS, vol. 1341, pp. 247–264. Springer, Heidelberg (1997)
17. Harary, F., Norman, R., Cartwright, D.: Structural Models: An Introduction to the Theory of Directed Graphs. Wiley, New York (1966)
18. Sutherland, D.: A model of information. In: Proceedings of the 9th National Computer Security Conference (1986)
19. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: POPL 2004: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 186–197. ACM, New York (2004)
20. Cohen, E.: Information transmission in computational systems. SIGOPS Oper. Syst. Rev. 11(5), 133–139 (1977)

Authorization Policy Specification and Enforcement for Group-Centric Secure Information Sharing

Ram Krishnan[1,2] and Ravi Sandhu[1]

[1] Institute for Cyber Security
[2] Department of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, TX
{ram.krishnan,ravi.sandhu}@utsa.edu

Abstract. In this paper, we propose a methodology for incremental security policy specification at varying levels of abstraction while maintaining strict equivalence with respect to authorization state. We specifically consider the recently proposed group-centric secure information sharing (g-SIS) domain. The current specification for g-SIS authorization policy is *stateless* in the sense that it solely focuses on specifying the precise conditions under which authorization can hold in the system while only considering the history of actions that have occurred. The stateless application policy has been specified using linear temporal logic. In this paper, we develop an enforceable specification that is *stateful* in the sense that it is defined using specific data structures that are maintained in each state so as to make authorization decisions. We show that the stateful specification is authorization equivalent to that of stateless. That is, in any state, authorization will hold in stateful if and only if it also holds in the stateless specification.

Keywords: Authorization, Enforcement, Equivalence, Security Policy.

1 Introduction

A fundamental problem in access control is the consistency of specification and enforcement of authorization policies. A large body of literature focuses on either the specification of authorization policies or its enforcement independent of each other. Our focus in this paper is to bridge these two areas. Our application domain is the recently proposed model for group-centric secure information sharing or g-SIS [4,6]. In g-SIS, users and objects are brought together in a group to promote sharing and collaboration. Users may join and leave and objects may be added and removed from the group. The join, leave, add and remove operations may have different authorization semantics as will be discussed later. A formal set of core properties that are required of all g-SIS specifications have been defined given the basic group operations of join, leave, add and remove. Further, a specification, called the π-system, has been formulated and proven to satisfy the core g-SIS properties.

The π-system specification is defined in a *stateless* manner using first-order linear temporal logic (FOTL). (FOTL differs from the familiar propositional linear temporal logic [7] by incorporating predicates with parameters, constants, variables, and quantifiers.) Specifically, the π-system is not directly enforceable in the way it is specified because it does not define the data structures that need to be maintained in order to make authorization decisions. Instead, the FOTL characterization of the π-system simply specifies the sequence of actions that need to have occurred in the past in order for authorization to hold at any given state. Thus, for example, a stateless specification may specify that a user may access an object in a group in a particular state if and only if the user had joined the group in the past, the object has been added to the group in the past and both the user and object are current members in the group (that is, the user has not left and the object has not been removed). Note that such a characterization using FOTL does not specify how to enforce that policy. A *stateful* specification, on the other hand, specifies the data structures that need to be maintained in the system so that they can be inspected in each state and authorization decisions be made.

In this paper, we develop a stateful specification for the π-system and prove that this specification is *authorization equivalent* to the stateless π-system specification. That is, a user will be authorized to access an object in a group in the stateful π-system specification *if and only if* it is also the case in the stateless π-system specification.

The separation of stateless from the stateful specification has a number of important virtues. A security policy researcher developing the stateless specification is not distracted by the data structures that need to be designed and maintained. Instead, she can focus purely on the precise characterization of the conditions under which authorization should hold in her system. Formal specification using FOTL also allows one to conduct rigorous formal analysis using automated techniques such as model checking as demonstrated in [4]. Once the stateless specification is developed, one can then focus on the data structure design and mechanisms needed to enforce the stateless policy. As will be shown, while the stateless specification may be complex for a non-expert in the field, the stateful specification is understandable and can be implemented by relatively competent programmers. The techniques we use include algorithmic specification of stateful π-system and induction for our proofs. We believe that this can be applied in various other application domains in which new policy specifications are developed.

This line of work is inspired in part by the relationship between the non-interference [2] and the Bell-LaPadula model [1]. The Bell-LaPadula model provides a lattice structure of security labels and the famous simple-security and star-properties to enforce one-directional information flow in the lattice. This is a stateful specification in that it describes data structures and rules that are enforceable. The non-interference specification is stateless and makes reference only to input-output behavior of a secure system. Our goals in this paper are to formalize authorization policy rather than information flow policy. Nonetheless the stateless and stateful distinction has strong similarities and the non-interference work has been inspirational. To the best of our knowledge, this is the first effort towards bridging authorization policy specification and enforcement.

The rest of the paper proceeds as follows. In section 2, we give a brief background on g-SIS and an overview of the stateless π-system specification. In section 3, we present a stateful specification for the π-system. In section 4, we show the equivalence of the stateful and stateless π-system specifications. We discuss future work and conclude in section 5.

2 Background

In this section, we provide a brief overview of g-SIS. A detailed discussion can be found in [4] and [6].

2.1 Overview of g-SIS

In g-SIS, users may join, leave and re-join the group. Similarly, objects may be added, removed and re-added to the group. Authorization may hold in any state depending on the relative membership status of the user and object in question. The group operations join, leave, add and remove can be of different types with various authorization semantics. We use the following shorthand to denote such different semantics of group operations:

$$\begin{aligned}
\mathrm{Join}(u,g) &= (\mathrm{join}_1(u,g) \vee \mathrm{join}_2(u,g) \vee ... \vee \mathrm{join}_m(u,g)) \\
\mathrm{Leave}(u,g) &= (\mathrm{leave}_1(u,g) \vee \mathrm{leave}_2(u,g) \vee ... \vee \mathrm{leave}_n(u,g)) \\
\mathrm{Add}(o,g) &= (\mathrm{add}_1(o,g) \vee \mathrm{add}_2(o,g) \vee ... \vee \mathrm{add}_p(o,g)) \\
\mathrm{Remove}(o,g) &= (\mathrm{remove}_1(o,g) \vee \mathrm{remove}_2(o,g) \vee ... \vee \mathrm{remove}_q(o,g))
\end{aligned}$$

Thus, for instance, $\mathrm{join}_1(u,g)$ could represent a specific type of join operation that is different in authorization semantics from that of $\mathrm{join}_2(u,g)$. However, $\mathrm{Join}(u,g)$ captures the notion that a join operation of some type has occurred for u in g.

Table 1. Intuitive summary of temporal operators used in this paper

Future/Past	Operator	Read as	Explanation
Future	$\bigcirc$	Next	($\bigcirc$ p) means that the formula p holds in the next state.
	$\square$	Henceforth	($\square$ p) means that the formula p will continuously hold in all future states starting from the current state.
	$\mathcal{W}$	Unless	It says that p holds either until the next occurrence of q or if q never occurs, it holds throughout.
Past	$\blacklozenge$	Once	($\blacklozenge$ p) means that formula p held at least once in the past.
	$\mathcal{S}$	Since	(p $\mathcal{S}$ q) means that q happened in the past and p held continuously from the position following the last occurrence of q to the present.

Definition 1 (State in Stateless Specification). *A state in the stateless specification is an interpretation that maps each predicate in the language to a relation over appropriate carriers.*

The predicates in the g-SIS language include action predicates such as Join, Leave, Add and Remove and an authorization predicate Authz. These predicates are specified over appropriate sorts (types). The semantic values over which a variable ranges depend on the variable's sort and are drawn from a set that is called the *carrier* of that sort. We use standard upper-case roman characters such as U (user sort) to denote sorts and calligraphic letters such as $\mathcal{U}$ (user carrier) to denote the corresponding carriers. A detailed discussion of the g-SIS language can be found in [4].

Definition 2 (Stateless Trace). *A trace in the stateless specification is an infinite sequence of states.*

The formulas that we specify below talk about stateless traces.

Well-Formed Traces. We now introduce four formulas that define what we call *well-formed* g-SIS traces. (An intuitive overview of temporal operators used in this paper is provided in table 1.) The formulas we consider treat the authorization a user has to access an object independently of actions involving other users and objects. Thus, from here on it is often convenient to omit the parameters in all of the predicates. We also omit the quantifiers as they can be easily inferred from the context (join and leave are user operations, add and remove are object operations).

A. An object cannot be Added and Removed and a user cannot Join and Leave at the same time.[1]

$$\tau_0 = \Box(\neg(\text{Add} \wedge \text{Remove}) \wedge \neg(\text{Join} \wedge \text{Leave}))$$

B. For any given user or object, two types of operations cannot occur at the same time.

$$\tau_1 = \forall i,j\ \Box((i \neq j) \rightarrow \neg(\text{join}_i \wedge \text{join}_j)) \wedge \forall i,j\ \Box((i \neq j) \rightarrow \neg(\text{leave}_i \wedge \text{leave}_j)) \wedge \forall i,j\ \Box((i \neq j) \rightarrow \neg(\text{add}_i \wedge \text{add}_j)) \wedge \forall i,j\, \Box((i \neq j) \rightarrow \neg(\text{remove}_i \wedge \text{remove}_j))$$

Thus, for example, a user cannot join with 2 different semantics in the same state. Multiple occurrences of the same event in a given state (i.e. when i equals j above) are treated as a single occurrence of that event in FOTL.

C. If a user u joins a group, u cannot join again unless u first leaves the group. Similar rules apply for other operations.

$$\tau_2 = \Box(\text{Join} \rightarrow\!\!\!\!\rightarrow (\neg\text{Join}\ \mathcal{W}\ \text{Leave})) \wedge \Box(\text{Leave} \rightarrow\!\!\!\!\rightarrow (\neg\text{Leave}\ \mathcal{W}\ \text{Join})) \wedge \Box(\text{Add} \rightarrow\!\!\!\!\rightarrow (\neg\text{Add}\ \mathcal{W}\ \text{Remove})) \wedge\ \Box(\text{Remove} \rightarrow\!\!\!\!\rightarrow (\neg\text{Remove}\ \mathcal{W}\ \text{Add}))$$

D. A Leave event cannot occur before Join. Similarly for objects.

$$\tau_3 =\ \Box(\text{Leave} \rightarrow \blacklozenge\text{Join}) \wedge \Box(\text{Remove} \rightarrow \blacklozenge\text{Add})$$

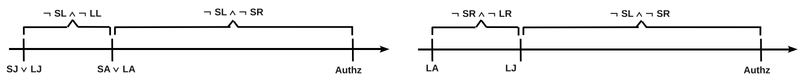
Thus, in any given trace, an object needs to be added before a remove operation may occur in any state.

2.2 The Stateless π-system G-SIS Specification

The π-system specification supports two types of semantics for join, leave, add and remove operations namely: strict and liberal.

A strict join (SJ) allows the joining user to access only those objects added on or after the state in which the user joins. A liberal join (LJ), in addition, allows the joining user to access objects added to the group prior to the join state.

[1] Note that here and below we introduce names of the form τ_j for each of the formulas for later reference. The equality introduces shorthands for the respective formulas.



(a) Formula λ_1 (b) Formula λ_2

Fig. 1. Stateless specification illustration

On strict leave (SL), the user loses access to all objects in the group. On liberal leave (LL), the user retains access to all objects that were authorized in the leaving state.

Similarly, for objects, on strict add (SA), the added object may be accessed only by users who have joined at or prior to the state in which the object is added to the group. Liberal add (LA) does not have such a constraint.

On strict remove (SR), the object cannot be accessed by any user. On liberal remove (LR), the object may be accessed by users who were authorized to access the object in the remove state.

The π-system specification supports the strict and liberal semantics for group operations. Given that different users may join and leave with different semantics and different objects may be added and removed with different semantics, the π-system specifies the precise conditions under which authorization for a user to access an object in a group may hold in the system.

Definition 3 (Stateless π-system). *The stateless π-system specification, $\pi_{stateless}$, accepts traces satisfied by the following formula:*

$$\forall u.\forall o.\forall g.\Box(\mathrm{Authz}(u,o,g,\mathrm{read}) \leftrightarrow \lambda_1 \vee \lambda_2) \wedge \bigwedge_{0\leq j\leq 3} \tau_j$$

where,

$$\lambda_1 = ((\neg\mathrm{SL} \wedge \neg\mathrm{SR})\ \mathcal{S}\ ((\mathrm{SA} \vee \mathrm{LA}) \wedge ((\neg\mathrm{LL} \wedge \neg\mathrm{SL})\ \mathcal{S}\ (\mathrm{SJ} \vee \mathrm{LJ}))))$$
$$\lambda_2 = ((\neg\mathrm{SL} \wedge \neg\mathrm{SR})\ \mathcal{S}\ (\mathrm{LJ} \wedge ((\neg\mathrm{SR} \wedge \neg\mathrm{LR})\ \mathcal{S}\ \mathrm{LA})))$$

and the τ_j's are the well-formedness constraints.

Given a specific user and an object, note that formula λ_1 handles the scenario where an add operation occurs after a join operation (figure 1(a)) and formula λ_2 handles the scenario where an add operation occurs before a join operation (figure 1(b)). (Here, due to the semantics of the strict add and strict join, we do not need to check for their occurrence in formula λ_2 illustrated in figure 1(b)). In [6], we have shown that the specification above is consistent with the semantics of strict and liberal operations discussed earlier. In addition, we have specified a set of core security properties that are required of any g-SIS specification and shown that the stateless π-system specification discussed above satisfies those core properties.

A g-SIS stateless specification with read and write operations (that supports multiple versions of the object) has been specified in [4]. Although we consider a stateless specification for read authorization in this paper, our discussion is not specific to the type of permission.

Table 2. Stateful Specification (Request Handling)

```
main(){
    // Phase 1 and 2 time periods below are allocated such that phase 1 occurs before
    // phase 2 and tasks in perTick step below conclude before the tick interval elapses.
    perTick: During each tick interval i {
        Phase 1:{ // Steps 1.1, 1.2 and 1.3 may execute concurrently.
            1.1. For each user, accept the first request received and
            store that information in variable userReq(u,g).
            // the request received could be one of:
            // SJReq(u,g), LJReq(u,g), SLReq(u,g) and LLReq(u,g).
            1.2. For each object, accept the first request received and
            store that information in variable objectReq(o,g).
            // the request received could be one of:
            // SAReq(o,g), LAReq(o,g), SRReq(o,g) and LRReq(o,g).*/
            1.3. Accept all the authorization requests:
                if (isAuthz(u,o,g)) authzReq=authzReq ∪ isAuthz(u,o,g)
                // isAuthz(u,o,g) represents authorization request for user u to access object o.
        }
        Phase 2:{ // Steps 2.1 and 2.2 must be sequential. However, the processing of
            // captured requests in step 2.1 may be done concurrently.
            2.1. For each captured request, invoke the corresponding function in
            table 3 with the appropriate parameters.
            // for example, if userReq(u,g) is SJReq(u,g), invoke userEvent(u,g,join,i,strict).
            2.2. Process each authorization request:
                for each (isAuthz(u,o,g) ∈ authzReq)
                    authzResult(u,o,g)=authzSF(u,o,g);
        }
        Reset all variables appropriately.
    }
}
```

3 Stateful π-system

In this section, we develop a stateful π-system specification that is authorization equivalent to the stateless specification—that is a user will be authorized to access an object in the stateful system if and only if it is also the case in the stateless system. Evidently, the stateless specification is highly abstract and specified using FOTL. The stateful specification that we develop is an incremental step in the direction of a concrete implementation of a system that is reasonably authorization equivalent to the stateless specification. We say “reasonably” because it is our fundamental hypothesis that all practical distributed systems will inevitably face real-world issues such as network delay and caching, which will lead to authorization inconsistencies with the idealized stateless specification. Thus such systems can at most be approximate to the stateless specification. One such approximation is the notion of stale-safety [3] that bounds acceptable delay between the time at which an action (such as reading an object) was known to be authorized and the time at which that action is actually performed. Our future refinements of the stateful π-system will consider various notions of such approximations.

Table 3. Stateful Specification (enforcing well-formedness constraints)

```
int userEvent(User u, Group g, uEvent e, interval t, uSemantics s){
    1. Check that the current uEvent e is not the same as the
    uEvent value in the previous tuple in table(u,g). If so, return 0.
    // This ensures, for example, that a join event is not followed
    // immediately by another join.
    2. Also check, in case the table is empty, then e is not an SL or LL. If so, return 0.
    // This ensures that the first user event entry in table(u,g) is not leave.
    3. Enter <t,e,s> into table(u,g) and return 1.
}
int objectEvent(Object o, Group g, oEvent e, interval t, oSemantics s){
    1. Check that the current oEvent e is not the same as the
    oEvent value in the previous tuple in table(o,g). If so, return 0.
    // This ensures, for example, that an add event is not followed
    // immediately by another add.
    2. Also check, in case the table is empty, then e is not an SR or LR. If so, return 0.
    // This ensures that the first object event entry in table (o,g) is not remove.
    3. Enter <t,e,s> into table(o,g) and return 1.
}
```

As the first transition from an abstract specification towards an implementable specification, the stateful specification that we design is centralized in the sense that authorization decisions are made based on data structures maintained in a specific repository for each user and object. There could be different repositories for different users and objects that may be distributed on the whole. Specifically, we are not concerned about replication of data structures of a user or an object and maintenance of its consistency. We also not concerned about distributing parts of the data structure of a user or an object. Authorization decisions for a specific user to access a specific object are made based on their specific data structures maintained at specific repositories.

Note that the stateless specification simply does not admit traces of actions that do not obey the well-formedness constraints. More importantly, it does not (intentionally) specify how one should handle ill-formed traces. At the stateful specification level of abstraction, one must begin to address such issues. Many strategies may be employed—we will consider one for our stateful specification (discussed later).

3.1 Stateful π-system Design

In the stateful π-system, the data structures that we maintain and consult with for making authorization decisions are simple relations for users and objects in the group—which we refer to informally as tables. For instance, the data structure for a user u in a group g, table(u,g), contains a history of that user's joins and leaves in the group. (The group parameter g is specified for being precise. The reader may safely ignore this in the rest of this paper as we focus only on one group at any time.) The format of each tuple in table(u,g) is: <*time-stamp, event, semantics*>. Here *event* is either join or leave, *semantics* is either strict or liberal and *time-stamp* specifies the time at which this

Table 4. Stateful Specification (Authorization Decision)

```
int authzSF(User u, Object o, Group g){

step 1: Fetch tables table(u,g) and table(o,g). If either table is empty, return 0.
        Merge sort table(u,g) and table(o,g) in ascending order of timestamp.
            In case of same timestamp, follow precedence rules apply:
              (i) Add and Join same timestamp: Add follows Join
              (ii) Join and Remove same timestamp: Join follows Remove
              (iii) Add and Leave same timestamp: Add follows Leave
              (iv) Remove and Leave same timestamp: any order
        Let n be the total number of entries in the merged table.

step 2: for i=1 to n{
          case event[i]=join{
step 2a:      (i) Step down the table looking for an add event. If a leave event is encountered
              prior to add event, continue step 2 for loop. If no add event found, return 0.
              (ii) From the point the add event was found in the table, step down all the way
              to index n ensuring no SL or SR is encountered.
              If SL found, continue step 2. If SR found, continue step 2a from current index.
              (iii) return 1;
          }
          case event[i]=add && eventType[i]=liberal{
step 2b:      (i) Step down the table looking for an LJ event. If a remove event is encountered
              prior to LJ event, continue step 2 for loop. If no LJ event found, return 0.
              (ii) From the point the LJ event was found in the table, step down all the way
              to index n ensuring no SL or SR is encountered.
              If SR found, continue step 2. If SL found, continue step 2b from current index.
              (iii) return 1;
          }
        }
step 3: return 0;
}
```

event occurred as per a global clock. Thus a snapshot of table(u,g) at any point in time gives a chronological history of the user joining and leaving (possibly many times) and whether they were of strict or liberal type. Similarly, a tuple in an object data structure, table(o,g), has the same format as the user table except *event* is either add or remove. Note that the number of tuples in any table is not bounded.[2]

The stateful specification for the π-system is presented in tables 2, 3 and 4. The authzSF function in table 4 returns 1 if a user u is authorized to access an object o, 0 otherwise. It does so by inspecting the data structures: table(u,g) and table(o,g). As mentioned earlier, the stateful π-system must also specify how the requests to join,

[2] Keeping them unbounded has many virtues. For instance, as we will see, this facilitates user data structures not being touched when an object data structure needs to be updated (and vice-versa). Of course, there are other data structure designs where they may be bounded but with different pros and cons.

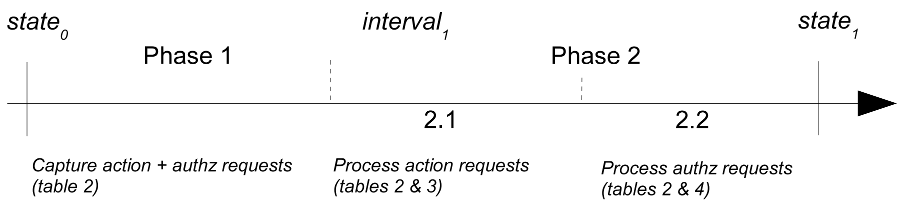

state0
interval1
state1
Phase 1
Phase 2
2.1
2.2
Capture action + authz requests (table 2)
Process action requests (tables 2 & 3)
Process authz requests (tables 2 & 4)

Fig. 2. Stateful π-system Overview

leave, add and remove and requests to ascertain if users are authorized to read objects are processed. Tables 2 and 3 specify one of many possible ways to do this. We discuss each of these 3 components of the stateful π-system in further detail below.

3.2 Stateful π-system Specification

An overview of how the functions in the tables 2, 3 and 4 interact is given in figure 2. Consider the main function in table 2. It receives and processes action requests (requests to join, leave, add and remove) and authorization requests during the time interval between any two clock ticks. The function works in two phases during each time interval. During phase 1, it receives the action and authorization requests. It filters the action requests so that only the first user request and the first object request are captured. (Different strategies for capturing action requests may be employed—e.g. it need not be the first request received that is captured.) This ensures, for instance, that only a join or a leave request of a specific type (strict or liberal) is captured for any given user but not both. However, all authorization requests are captured during phase 1. When phase 1 completes, further new requests are not admitted. During phase 2, first all action requests received in phase 1 are processed using the user and object event processing functions in table 3 and then all the captured authorization requests are evaluated using authzSF function in table 4. At the end of phase 2, the data structures are up-to-date and authorization decisions are complete for all the requests received in phase 1.

Consider the function userEvent in table 3 which processes the user requests received by the function in table 2. The check performed in step 1 ensures that user requests to repeatedly join without an intermittent leave (and vice-versa) are ignored. Similarly, step 2 ensures that the first entry in the table does not begin with a leave operation. If all is well, a new tuple is entered into the table and the function returns 1. The function returns 0 in all other cases. The objectEvent function similarly processes object requests. Note that tables 2 and 3 together achieve well-formedness constraints of stateless π-system specification.

The function authzSF in table 4 returns 1 if a user u is authorized to access an object o in group g, 0 otherwise. This algorithm can be optimized but we keep it straight-forward for simpler presentation. It begins by taking the corresponding user and object tables as input. Note that if either table is empty (i.e., either the user or the object has never been a member of the group), the user is not authorized to read the object. By appending the tuples to the respective tables as the events occur, table(u,g) and table(o,g) are pre-sorted with respect to the time-stamp. The function merge sorts these two tables based on the time-stamp entries to obtain a table of events of u and o in the chronological order of occurrence. In the event a user and object entry in the respective tables have the same time-stamp, we specify precedence rules to resolve the tie for sorting the tuples consistent with temporal operator semantics in the stateless π-system. If Add and Join occur at the same time, Add follows Join. If Join and Remove occur at the same time, Join follows Remove. If Add and Leave occur at the same time, Add follows Leave. Finally, if Remove and Leave occur at the same time, they can be merge sorted in any order. Let the total number of entries in the merged table be n.

The algorithm proceeds by iterating through each tuple in this new merge sorted table. We assume that event[i] fetches the specific event (such as join or add) from the i$^{\text{th}}$ entry in the merged table and eventType[i] fetches the respective semantics (such as strict or liberal) of that event from the same tuple. Each of the two cases in the for loop looks for an overlapping period of authorizing membership between the user and object, much like formulas λ_1 and λ_2. The first case looks for a join event followed by an add event (see figure 1(a)) and the second case looks for an add event followed by a join event (see figure 1(b)). As per λ_2, the second case looks for a liberal add followed by a liberal join. The remaining part of the case statements conduct checks to ensure that there is no subsequent de-authorizing event such as strict leave or remove following this point of authorization. If there is none, the algorithm returns 1 indicating that the user is authorized. Otherwise it returns 0 after step 3.

3.3 Implementation Considerations

Evidently, the stateful specification that has been presented in tables 2, 3 and 4 can be comprehended and implemented by a competent programmer as compared to the temporal logic based stateless specification. Since the stateless specification has been analysed and certain security properties have been proven [4,6] and has been shown to be authorization equivalent to the stateful specification (section 4), the stateful specification also is guaranteed to have those security properties.

As mentioned earlier, the authzSF function in table 4 is not designed for efficiency but for ease of presentation. The worst case time complexity of this function is roughly $\mathcal{O}(n^2)$ where n is the sum of the number of events in the user and object tables. This is because for each of the n iterations of the outer for loop in step 2, the loops in one of the inner case statements could run through a maximum of n iterations.

This stateful specification has a few limitations. For instance, both the user and object tables are unbounded. Nevertheless, this is not a major issue in many practical applications in which membership status of users and objects do not change frequently. Also, due to nature of phases 1 and 2 in table 2, all action requests need to be received before they can be processed. Thus during phase 2 of interval, no requests will be accepted. The ordering of tasks in two phases ensures that the requests received during the time interval will affect the authorization values that hold at the upcoming state. These constraints may be unacceptable for certain application scenarios. Addressing such limitations of the stateful specification is not the primary focus of this paper. Note that the current stateful specification design allows user and object data structures to be maintained in a distributed manner so that if a user membership status changes, it does not require updating data structures of other users and objects in the system. One can design alternate stateful specifications for the same stateless specification with different trade-offs. For instance, one can maintain a single data structure that involves both users and objects. But changes in any object's group membership status will entail updating entries for all users in the system. This would have limitations in distributing it.

4 Equivalence of Stateful and Stateless π-system Specifications

In this section, we show that the stateful specification is authorization equivalent to the stateless specification. That is, in all possible traces, a user will be authorized to access an object at any given state in the stateful π-system if and only if it is also the case in the stateless π-system.

Given that we are dealing with traces in the stateless specification, we propose a similar notion of traces in the stateful specification.

Definition 4 (State in Stateful Specification). *A state in the stateful specification is a specific interpretation of every user and object data structure maintained in the system at the end of every clock tick.*

Definition 5 (Stateful Trace). *A trace in the stateful specification is an infinite sequence of states.*

Definition 6 (Stateful π-system). *The stateful π-system specification, $\pi_{stateful}$, is given in table 2 which consists of functions from tables 3 and 4.*

Our goal now is to show that given a stateless and a corresponding stateful trace, authorization is equivalent in every state. To establish this "correspondence", we specify mappings that would take a stateless trace and create a stateful trace and vice-versa.

Notation. We use σ to denote a stateless trace and $\widehat{\sigma}$ to denote a stateful trace. σ_i refers to state i in a trace σ with infinite states. We use $\sigma_{i,j}$ to denote a state i in σ where we only consider the first j states. Actions are represented using relations. Thus $\langle \mathbf{u}, \mathbf{g} \rangle \in [\![\mathrm{SJ}_{stateless}]\!]\sigma_i$ denotes that a user $\mathbf{u}$ is strictly joined to group $\mathbf{g}$ in state i in a stateless trace σ. Similarly, $\langle i, \mathrm{Join}, \mathrm{Liberal} \rangle \in [\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i$ denotes user $\mathbf{u}$ has liberally joined group $\mathbf{g}$ in state i in a stateful trace $\widehat{\sigma}$.

Note that the time interval that a clock tick corresponds to is abstract. Any event request (such as a request to join) that is processed during a transition from clock tick (state) i to i+1 will receive a time-stamp of i+1. This convention makes stateful specification consistent with the FOTL semantics in the stateless specification.

Definition 7 (Action Trace). *Given a stateless or stateful trace in the π-system, an* action trace *is a sequence of states excluding the authorization relation.*

Definition 8 (Action Equivalence). *A stateful trace $\widehat{\sigma}$ and a stateless trace σ are* action equivalent *if the join, leave, add and remove actions match for every user and object in every group in the corresponding states in $\widehat{\sigma}$ and σ.*

Definition 9 (α-mapping). *Given a stateless trace σ in $\pi_{stateless}$, α-mapping creates an action equivalent stateful trace $\widehat{\sigma}$ in $\pi_{stateful}$.*

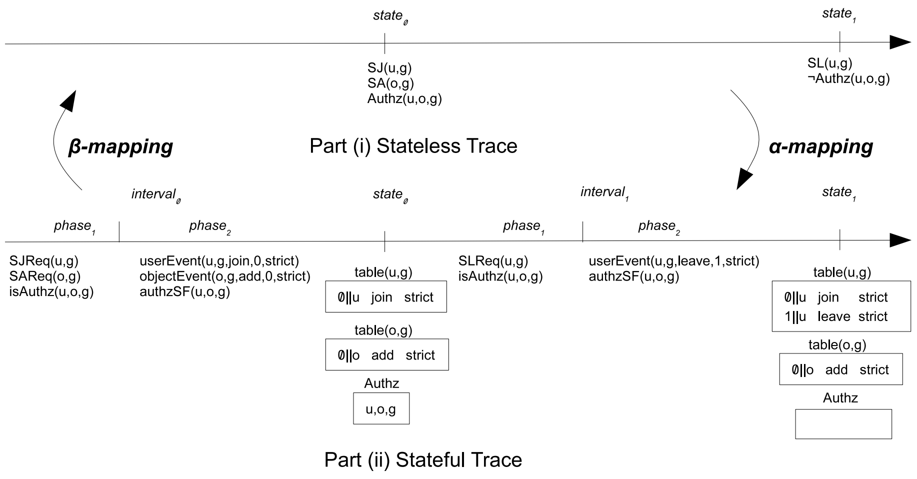


Fig. 3. α and β mapping. Part (i) shows a sample stateless trace and part (ii) shows a corresponding stateful trace. Note that the stateful trace generates the required action and authorization requests during the time interval leading up to the state.

Rules used for α-mapping are straight-forward and given here by example. For example (see figure 3), for each $\langle \mathbf{u}, \mathbf{g} \rangle \in [\![\mathrm{SJ}_{stateless}]\!]\sigma_i$, create an entry $\langle i, \mathrm{Join}, \mathrm{Strict} \rangle$ in $[\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i$. This is achieved by sending a SJReq(**u**,**g**) (see table 2) during phase 1 in the time interval between the state transition from $\widehat{\sigma}_{i-1}$ to $\widehat{\sigma}_i$. Similarly, for each $\langle \mathbf{u}, \mathbf{g} \rangle \in [\![\mathrm{LJ}_{stateless}]\!]\sigma_i$, create an entry $\langle i, \mathrm{Join}, \mathrm{Liberal} \rangle$ in $[\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i$. Similar rules apply to other predicates.

Definition 10 **(β-mapping).** *Given a stateful trace $\widehat{\sigma}$ in $\pi_{stateful}$, β-mapping creates an action equivalent stateless trace σ in $\pi_{stateless}$.*

Rules used for β-mapping are straight-forward and given here by example. For example (see figure 3), for each tuple in $[\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i - [\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_{i-1}$, create that entry in corresponding relation in the stateless trace. That is if $\langle i, \mathrm{Join}, \mathrm{Strict} \rangle \in [\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i - [\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_{i-1}$, then create an entry $\langle \mathbf{u}, \mathbf{g} \rangle$ in $[\![\mathrm{SJ}_{stateless}]\!]\sigma_i$. Similarly, for each $\langle i, \mathrm{Join}, \mathrm{Liberal} \rangle \in [\![\mathrm{table}(\mathbf{u}, \mathbf{g})]\!]\widehat{\sigma}_i$, create an entry $\langle \mathbf{u}, \mathbf{g} \rangle$ in $[\![\mathrm{LJ}_{stateless}]\!]\sigma_i$. Similar rules apply to other operations in the stateful specification.

Lemma 1. *For every action trace σ that is generated by $\pi_{stateless}$, a stateful action trace $\widehat{\sigma}$ constructed using α-mapping is accepted by $\pi_{stateful}$.*

By the term "accepted by" above, we mean that by inputting an α-mapped trace to the stateful π-system, the data structure it maintains must reflect the exact action trace of the stateless π-system (see figure 3 for example).

Lemma 2. *For every action trace $\widehat{\sigma}$ generated by $\pi_{stateful}$, a stateless action trace constructed using β-mapping is accepted by $\pi_{stateless}$.*

By the term "accepted by" above, we mean that the β-mapped stateless action trace will be well-formed as per the stateless π-system specification. The proofs of lemmas 1 and 2 follow directly from their definitions. Due to space limitations, the proofs are provided in [5]. Next, we have the following 2 lemmas.

Lemma 3 (Soundness). *For every trace $\widehat{\sigma}$ accepted by $\pi_{stateful}$, there exists a β-mapped trace σ that is accepted by $\pi_{stateless}$ such that:*

$$\forall i \in \mathbb{N}. \forall t \in \langle \mathcal{U}, \mathcal{O}, \mathcal{G} \rangle . t \in [\![\mathrm{Authz}_{\pi_{stateful}}]\!]\widehat{\sigma}_i \rightarrow t \in [\![\mathrm{Authz}_{\pi_{stateless}}]\!]\sigma_i$$

Lemma 4 (Completeness). *For every trace σ accepted by $\pi_{stateless}$, there exists an α-mapped trace $\widehat{\sigma}$ that is accepted by $\pi_{stateful}$ such that:*

$$\forall i \in \mathbb{N}. \forall t \in \langle \mathcal{U}, \mathcal{O}, \mathcal{G} \rangle . t \in [\![\mathrm{Authz}_{\pi_{stateless}}]\!]\sigma_i \rightarrow t \in [\![\mathrm{Authz}_{\pi_{stateful}}]\!]\widehat{\sigma}_i$$

Due to space limitations, the proofs for lemmas 3 and 4 are provided in [5]. The proofs are inductive.

Theorem 1. *The stateful and stateless π-system specifications are* authorization equivalent. *That is:*

$$\forall i \in \mathbb{N}. \forall t \in \langle \mathcal{U}, \mathcal{O}, \mathcal{G} \rangle . t \in [\![\mathrm{Authz}_{\pi_{stateful}}]\!]\widehat{\sigma}_i \leftrightarrow t \in [\![\mathrm{Authz}_{\pi_{stateless}}]\!]\sigma_i$$

Proof. The theorem follows from lemmas 3 and 4.

The above theorem states that in every state in a stateful trace, the authorization relation is equivalent to that of the corresponding state in a stateﬂess trace. We have shown that a highly abstract temporal logic based stateless specification can be grounded in a concrete stateful specification while maintaining equivalency with respect to authorization.

5 Conclusion and Future Work

We presented a methodology for consistent specification and enforcement of authorization policies. The stateless specification is highly conducive to automated formal analysis using techniques such as model checking. However, it cannot be enforced using the way it is specified. The stateful specification focuses on how to enforce the stateless policy using distributed data structures and associated algorithms. This specification can be implemented by programmers. We have established a formal bridge between a highly abstract stateless specification and a relatively concrete stateful specification. The next level of refinement is to generate distributed specification which account for approximation (for example, due to network delay and caching) with respect to stateless specification and a concrete implementation. Such incremental refinement of policy specification while maintaining consistency at each transition is critical in secure systems design.

Our current stateful specification, although highly distributed, maintains unbounded history of user and object actions. Our follow on work focuses on generating alternate stateful specifications. One approach is to generate a stateful specification with bounded data structures that maintain information about authorization status of each user for each object. While this bounds the data structures, it requires modifying every object's data structures if the user's membership status changes in the group. Another approach is to generate a hybrid specification that combine the pros and cons of the two approaches above and proving equivalence with respect to stateless specification. We believe our methodology can be extended to other application domains with suitable adaptation of proof techniques as needed.

Acknowledgments. The authors are partially supported by grants from AFOSR MURI and State of Texas Emerging Technology Fund.

References

1. Bell, D., La Padula, L.: Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306 (1975)
2. Goguen, J., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy (1982)
3. Krishnan, R., Niu, J., Sandhu, R., Winsborough, W.: Stale-safe security properties for group-based secure information sharing. In: Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering, pp. 53–62. ACM, New York (2008)
4. Krishnan, R., Niu, J., Sandhu, R., Winsborough, W.: Group-centric secure information sharing models for isolated groups. To appear in ACM Transactions on Information and Systems Security (2011). Camera ready copy available at, http://engineering.utsa.edu/ krishnan/ journals/2010-tissecSACMAT.pdf
5. Krishnan, R., Sandhu, R.: Authorization Policy Specification and Enforcement for Group-Centric Secure Information Sharing (Full Version). Tech. Rep. CS-TR-2011-016, University of Texas at San Antonio (September 2011), http://engineering.utsa.edu/ krishnan/ conferences/2011iciss-full.pdf
6. Krishnan, R., Sandhu, R., Niu, J., Winsborough, W.H.: Foundations for group-centric secure information sharing models. In: Proc. of ACM Symposium on Access Control Models and Technologies (2009)
7. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–67 (1977)

Abductive Analysis of Administrative Policies in Rule-Based Access Control

Puneet Gupta, Scott D. Stoller, and Zhongyuan Xu

Department of Computer Science, Stony Brook University, USA

In large organizations, the access control policy is managed by multiple users (administrators). An administrative policy specifies how each user may change the policy. The consequences of an administrative policy are often non-obvious, because sequences of changes by different users may interact in unexpected ways. Administrative policy analysis helps by answering questions such as user-permission reachability, which asks whether specified users can together change the policy in a way that achieves a specified goal, namely, granting a specified permission to a specified user.

This paper presents a rule-based access control policy language, a rule-based administrative policy model that controls addition and removal of rules and facts, and a symbolic analysis algorithm for answering reachability queries. The algorithm can analyze policy rules that refer to sets of facts (e.g., information about users) that are not known at analysis time. The algorithm does this by computing conditions on the initial set of facts under which the specified goal is reachable by actions of the specified users.

1 Introduction

The increasingly complex security policies required by applications in large organizations cannot be expressed in a precise and robust way using access-control lists or role-based access control (RBAC). This has led to the development of attribute-based access control frameworks with rule-based policy languages. These frameworks allow policies to be expressed at a higher level of abstraction, making the policies more concise and easier to administer.

In large organizations, access control policies are managed by multiple users (administrators). An *administrative framework* (also called *administrative model*) is used to express policies that specify how each user may change the access control policy. Several administrative frameworks have been proposed for role-based access control, starting with the classic ARBAC97 model [6]. Fully understanding the implications of an administrative policy can be difficult, due to unanticipated interactions between interleaved sequences of changes by different users. This motivated research on analysis of administrative policies. For example, analysis algorithms for ARBAC97 and variants thereof can answer questions such as user-permission reachability, which asks whether changes by specified users can lead to a policy in which a specified user has a specified permission [5,4,7].

There is little work on administrative frameworks for rule-based access control [1,3], and it considers only addition and removal of facts, not rules.

This paper defines a rule-based access control policy language, with a rule-based administrative framework that controls addition and removal of both facts and rules. We call this policy framework ACAR (Access Control and Administration using Rules). It allows administrative policies to be expressed concisely and at a desirable level of abstraction. Nevertheless, fully understanding the implications of a rule-based administrative policy in ACAR is even more difficult than fully understanding the implications of an ARBAC policy, because in addition to considering interactions between interleaved sequences of changes by different administrators, one must consider possible chains of inferences using rules in each intermediate policy. This paper presents a symbolic analysis algorithm for answering atom-reachability queries for ACAR policies, i.e., for determining whether changes by specified administrators can lead to a policy in which some instance of a specified atom, called the goal, is derivable. To the best of our knowledge, this is the first analysis algorithm for a rule-based policy framework that considers changes to the rules as well as changes to the facts.

It is often desirable to be able to analyze rule-based policies with incomplete knowledge of the facts in the initial policy. For example, a database containing those facts might not exist yet (if the policy is part of a system that has not been deployed), or it might be unavailable to the policy engineer due to confidentiality restrictions. Even if some database of initial facts exists and is available to the policy engineer, more general analysis results that hold under limited assumptions about the initial facts are often preferable to results that hold for only one given set of initial facts, e.g., because the policy might be deployed in many systems with different initial facts.

There are two ways to handle this. In the deductive approach, the user specifies assumptions—in the form of constraints—about the initial policy, and the analysis algorithm determines whether the goal is reachable under those constraints. However, formulating appropriate constraints might be difficult, and might require multiple iterations of analysis and feedback. We adopt an abductive approach, in which the analysis algorithm determines conditions on the set of facts in the initial policy under which the given goal is reachable. This approach is inspired by Becker and Nanz's abductive policy analysis for a rule-based policy language [2], and our algorithm builds on their tabling-based policy evaluation algorithm. The main difference between their work and ours is that they analyze a fixed access control policy: they do not consider any administrative framework or any changes to the rules or facts in the access control policy. Also, they do not consider constructors or negation, while our policy language allows constructors and allows negation applied to extensional predicates.

Our analysis algorithm may diverge on some policies. This is expected, because Becker and Nanz's abductive algorithm (which solves a simpler problem) may diverge, and because reachability for ACAR is undecidable. Undecidability of this problem is a corollary of our proof in [7] that user-permission reachability is undecidable for ARBAC97 extended with parameters, since ARBAC97 policies can be encoded in ACAR in a straightforward way. Identifying classes of policies for which the algorithm is guaranteed to terminate is a direction for future work; for now, we note that the algorithm terminates for the case studies we have considered so far, including a non-trivial fragment of a policy for a healthcare network. Also, the current version of the algorithm is incomplete (it "gives up" in some cases), but the algorithm can be extended to eliminate this incompleteness, following the approach sketched in Section 5.3; for now, we note that the current version of the algorithm already succeeds (does not "give up") for some non-trivial policies, including our healthcare network case study.

2 Policy Framework

Policy Language. The policy language is a rule-based language with constructors (functors) and negation (denoted "`!`"). Predicates are classified as intensional or extensional. Intensional predicates are defined by rules. Extensional predicates are defined by facts. Constructors are used to construct terms representing operations, rules (being added or removed by administrative operations), parameterized roles, etc. The grammar ensures that negation is applied only to extensional predicates; this is why we distinguish intensional and extensional predicates. The grammar appears below. p ranges over predicates, c ranges over constructors (functors), and v ranges over variables. The grammar is parameterized by the sets of predicates, variables, and constructors; these sets may be finite or countable. Predicates and constructors start with a lowercase letter; variables start with an uppercase letter. Constants are represented as constructors with arity zero; the empty parentheses are elided. Subscripts *in* and *ex* are mnemonic for intensional and extensional, respectively. A term or atom is *ground* if it does not contain any variables. A substitution θ is *ground*, denoted ground(θ), if it maps variables to ground terms. A *policy* is a set of rules and facts.

$$\begin{array}{llll}
\mathit{term} ::= v \mid c(\mathit{term}^*) & & \mathit{literal} ::= \mathit{atom}_{ex} \mid !\mathit{atom}_{ex} \mid \mathit{atom}_{in} \\
\mathit{atom}_{ex} ::= p_{ex}(\mathit{term}^*) & & \mathit{rule} ::= \mathit{atom}_{in} \texttt{ :- } \mathit{literal}^* \\
\mathit{atom}_{in} ::= p_{in}(\mathit{term}^*) & & \mathit{fact} ::= \text{ground instance of } \mathit{atom}_{ex}
\end{array}$$

The distinguished predicate `permit(user, operation)` specifies permissions, including permissions for administrative operations, as discussed below.

Administrative Framework. The administrative framework defines an API for modifying policies. Specifically, the operations in the API are `addRule`(*rule*), `removeRule`(*rule*), `addFact`($atom_{ex}$), and `removeFact`($atom_{ex}$). Let AdminOp = {`addRule`, `removeRule`, `addFact`, `removeFact`}. In addition, the framework defines how permission to perform those operations are controlled. These permissions are expressed using the `permit` predicate but given a special interpretation, as described below.

A *permission rule* is a rule whose conclusion has the form `permit(...)`. For an operation *op*, an *op* permission rule is a rule whose conclusion has the form `permit(...,` *op*`(...))` . An *administrative permission rule* is an *op* permission rule with $op \in$ AdminOp. In a well-formed policy, the argument to `addFact` and `removeFact` must be an extensional atom (it does not need to be ground).

A rule is *safe* if it satisfies the following conditions. (1) Every variable that appears in the conclusion outside the arguments of `addRule` and `removeRule` also appears in a positive premise. (2) Every variable that appears in a negative premise also appears in a positive premise. (3) In every occurrence of permit, the second argument is a constructor term, not a variable. (4) `addRule` and `removeRule` do not appear outside the second argument of `permit` in the conclusion. A policy is safe if all rules in the policy are safe.

Policy Semantics. The semantics $[\![P]\!]$ of a policy P is the least fixed-point of F_P, defined by $F_P(I) = \{a\theta \mid (a \texttt{ :- } a_1, \ldots, a_m, !b_1, \ldots, !b_n) \in P \wedge (\forall i \in [1..m] : \ a_i\theta \in I) \wedge (\forall i \in [1..n] : \ b_i\theta \notin I)\}$. To simplify notation, this definition assumes that the positive premises appear before the negative premises; this does not affect the semantics. Intuitively, $[\![P]\!]$ contains all atoms deducible from P. Atoms in the semantics are ground except that arguments of `addRule` and `removeRule` may contain variables. Limiting negation to extensional predicates ensures that F_P is monotonic. By the Knaster-Tarski theorem, the least fixed point of F_P can be calculated by repeatedly applying F_P starting from the empty set. Safety of the policy implies that, during this calculation, whenever $b_i \notin I$ is evaluated, b_i is ground; this simplifies the semantics of negation. We sometimes write $P \vdash a$ (read "P derives a") to mean $a \in [\![P]\!]$.

Fixed Administrative Policy. Our goal in this paper is to analyze a changing access control policy subject to a fixed administrative policy. Therefore, we consider policies that satisfy the *fixed administrative policy requirement*, which says that administrative permission rules cannot be added or removed, except that `addFact` administrative permission rules can be added. We allow this exception because it is useful in practice and can be accommodated easily.

We formalize this requirement as follows. A *higher-order* administrative permission rule is an administrative permission rule whose conclusion has the form $\texttt{permit}(\ldots, op(\texttt{permit}(\ldots, op'(\ldots))))$ with $op \in$ AdminOp and $op' \in$ AdminOp; in other words, it is a rule that permits addition and removal of administrative permission rules. A rule satisfies the fixed administrative policy requirement if either it is not a higher-order administrative permission rule or it is an administrative permission rule having the above form with $op = \texttt{addRule}$ and $op' = \texttt{addFact}$. A policy satisfies this requirement if all of the rules in it do.

Even in a policy with no higher-order administrative permission rules, the available administrative permissions may vary, because addition and removal of other rules and facts may change the truth values of the premises of administrative permission rules.

Administrative Policy Semantics. The above semantics is for a fixed policy. We specify the semantics of administrative operations and administrative permissions by defining a transition relation T between policies, such that $\langle P, U\text{:} \; Op, P')\rangle \in T$ iff policy P permits user U to perform administrative operation Op thereby changing the policy from P to P'.

Rule R is *at least as strict as* rule R' if (1) R and R' have the same conclusion, and (2) the set of premises of R is a superset of the set of premises of R'.

$\langle P, U : \texttt{addRule}(R), P \cup \{R\}\rangle \in T$ if there exists a rule R' such that (1) R is at least as strict as R', (2) $P \vdash \texttt{permit}(U, \texttt{addRule}(R'))$, (3) $R \notin P$, (4) R satisfies the fixed administrative policy requirement, and (5) R satisfies the safe policy requirement. Note that R' may be a partially or completely instantiated version of the argument of `addRule` in the `addRule` permission rule used to satisfy condition (2); this follows from the definition of $\vdash$. Thus, an administrator adding a rule may specialize the "rule pattern" in the argument of `addRule` by instantiating some of the variables in it and by adding premises to it. We call the argument of `addRule` or `removeRule` a "rule pattern", even though it is generated by the same grammar as rules, to emphasize that it can be specialized in these ways, giving the administrator significant flexibility to customize the rules, without giving the administrator additional authority.

$\langle P, U : \texttt{removeRule}(R), P \setminus \{R\}\rangle \in T$ if there exists a rule R' such that R is as least as strict as R', $P \vdash \texttt{permit}(U, \texttt{removeRule}(R'))$, and $R \in P$.

$\langle(P, U : \texttt{addFact}(a), P \cup \{a\}\rangle \in T$ if $P \vdash \texttt{permit}(U, \texttt{addFact}(a))$ and $a \notin P$.

$\langle(P, U : \texttt{removeFact}(a), P \setminus \{a\}\rangle \in T$ if $P \vdash \texttt{permit}(U, \texttt{removeFact}(a))$ and $a \in P$.

Case Study: Healthcare Network. As a case study, we wrote a policy with about 50 rules for a healthcare network (HCN). The HCN policy defines a HCN policy officer (`hcn_po`) role that can add rules that give the facility policy officer (`facility_po`) role for each constituent healthcare facility appropriate permissions to manage the facility's policy. We consider policies for two kinds of facilities: hospitals and substance abuse facilities.

For example, the rule below allows the `facility_po` to add rules that allow the `hr_manager` to appoint a member of a workgroup as head of that workgroup:

```
permit(User, addRule(permit(HRM,
                            addFact(memberOf(Head, wgHead(WG, Fac))))
                     :- hasActivated(HRM, hr_manager(Fac)),
                        !memberOf(HRM, workgroup(WG, Fac, WGtype))
                        memberOf(Head, workgroup(WG, Fac, WGtype))))
  :- hasActivated(User, facility_po(Fac))
```

where `memberOf`(U, R) holds if user U is a member of role R, `hasActivated`(U, R) holds if U has activated R, `WG` is the workgroup name, `Fac` is the facility name, and `WGtype` is `team` or `ward`. The negative premise prevents a `hr_manager` from appointing a head for a workgroup to which he or she belongs.

At Stony Brook Hospital (`sb_hosp`), a member of `facility_po(sb_hosp)` might use this permission to add the following rule, which allows `hr_manager( sb_hosp)` to appoint a team member as the team head, with the additional premise that the user is a clinician at `sb_hosp` with any specialty `Spcty`.

```
permit(HRM, addFact(memberOf(Head, wgHead(WG, sb_hosp))))
  :- hasActivated(HRM, hr_manager(sb_hosp)),
     !memberOf(HRM, workgroup(WG, sb_hosp, team))
     memberOf(Head, workgroup(WG, sb_hosp, team)),
     memberOf(Head, clinician(sb_hosp, Spcty))
```

At Stony Brook Substance Abuse Facility (`sb_saf`), `facility_po(sb_saf)` might add a similar rule except with a stricter additional premise, requiring the team head to have specialty `psychiatry`.

3 Abductive Reachability

This section defines abductive atom-reachability queries and their solutions.

Let a and b denote atoms, L denote a literal, and $\vec{L}$ denote a sequence of literals. An atom a is *subsumed* by an atom b, denoted $a \preceq b$, iff there exists a substitution θ such that $a = b\theta$. For an atom a and a set A of atoms, let $[\![a]\!] = \{a' \mid a' \preceq a\}$ and $[\![A]\!] = \bigcup_{a \in A} [\![a]\!]$.

A *specification of abducible atoms* is a set A of extensional atoms. An atom a is abducible with respect to A if $a \in [\![A]\!]$.

A *goal* is an atom.

Given an initial policy P_0 and a set U_0 of users (the active administrators), the *state graph* for P_0 and U_0, denoted $\mathrm{SG}(P_0, U_0)$, contains policies reachable from P_0 by actions of users in U_0. Specifically, $\mathrm{SG}(P_0, U_0)$ is the least graph (N, E) such that (1) $P_0 \in N$ and (2) $\langle P, U : Op, P' \rangle \in E$ and $P' \in N$ if $P \in N \wedge U \in U_0 \wedge \langle P, U : Op, P' \rangle \in T$.

An *abductive atom-reachability query* is a tuple $\langle P_0, U_0, A, G_0 \rangle$, where P_0 is a policy (the initial policy), U_0 is a set of users (the users trying to reach the goal), A is a specification of abducible atoms, and G_0 is a goal. Informally, P_0 contains rules and facts that are definitely present in the initial state, and $[\![A]\!]$ contains facts that might be present in the initial state. Other facts are definitely not present in the initial state and, since we make the closed world assumption, are considered to be false.

A *ground solution* to an abductive atom-reachability query $\langle P_0, U_0, A, G_0 \rangle$ is a tuple $\langle \Delta, G \rangle$ such that Δ is a ground subset of $[\![A]\!]$, G is a ground instance of G_0, and $\mathrm{SG}(P_0 \cup \Delta, U_0)$ contains a policy P such that $P \vdash G$. Informally, a ground solution $\langle \Delta, G \rangle$ indicates that a policy P in which G holds is reachable from $P_0 \cup \Delta$ through administrative actions of users in U_0.

A *minimal-residue ground solution* to a query is a ground solution $\langle \Delta, G \rangle$ such that, for all $\Delta' \subset \Delta$, $\langle \Delta', G \rangle$ is not a ground solution to the query.

A *tuple disequality* has the form $\langle t_1 \ldots, t_n \rangle \neq \langle t'_1, \ldots, t'_n \rangle$, where the t_i and t'_i are terms.

A *complete solution* to an abductive atom-reachability query $\langle P_0, U_0, A, G_0 \rangle$ is a set S of tuples of the form $\langle \Delta, G, D \rangle$, where Δ is a set of atoms (not necessarily ground), G is an atom (not necessarily ground), and D is a set (interpreted as a conjunction) of tuple disequalities over the variables in Δ and G, such that (1) Soundness: S represents ground solutions to the query, i.e., $\bigcup_{s \in S} [\![s]\!] \subseteq S_{gnd}$, where $[\![\langle \Delta, G, D \rangle]\!] = \{\langle \Delta\theta, G\theta \rangle \mid \mathrm{ground}(\theta) \wedge D\theta = \mathrm{true}\}$ and S_{gnd} is the set of ground solutions to the query, and (2) Completeness: S represents all minimal-residue ground solutions to the query, i.e., $\bigcup_{s \in S} [\![s]\!] \supseteq S_{min\text{-}gnd}$, where $S_{min\text{-}gnd}$ is the set of minimal-residue ground solutions to the query.

Transition Rules:

(root) $(\{\langle G\rangle\} \uplus N, \mathit{Ans}, \mathit{Wait}) \rightarrow (N \cup N', \mathit{Ans}, \mathit{Wait})$
if $N' = \mathbf{generate}_P(G)$

(ans) $(\{n\} \uplus N, \mathit{Ans}, \mathit{Wait}) \rightarrow (N \cup N', \mathit{Ans}[G \mapsto \mathit{Ans}(G) \cup \{n\}], \mathit{Wait})$
if n is an answer node with index G
$\nexists n' \in \mathit{Ans}(G) : n \preceq n'$
$N' = \bigcup_{n'' \in \mathit{Wait}(G)} \mathbf{resolve}(n'', n)$

(goal$_1$) $(\{n\} \uplus N, \mathit{Ans}, \mathit{Wait}) \rightarrow (N \cup N', \mathit{Ans}, \mathit{Wait}[Q' \mapsto \mathit{Wait}(Q') \cup \{n\}])$
if n is a goal node with current subgoal Q
$\exists Q' \in \mathit{dom}(\mathit{Ans}) \ : \ Q \preceq Q'$
$N' = \bigcup_{n' \in \mathit{Ans}(Q')} \mathbf{resolve}(n, n')$

(goal$_2$) $(\{n\} \uplus N, \mathit{Ans}, \mathit{Wait}) \rightarrow (N \cup \{\langle Q\rangle\}, \mathit{Ans}[Q \mapsto \emptyset], \mathit{Wait}[Q \mapsto \{n\}])$
if n is a goal node with current subgoal Q
$\forall Q' \in \mathit{dom}(\mathit{Ans}) \ : \ Q \not\preceq Q'$

Auxiliary Definitions:

$\langle G; []; S; \vec{c}; R; \Delta\rangle \preceq \langle G; []; S'; \vec{c}'; R'; \Delta'\rangle$ iff $|\Delta| \geq |\Delta'| \wedge (\exists \theta \ . \ S = S'\theta \wedge \Delta \supseteq \Delta'\theta)$

for an answer node $n = \langle \_; []; Q'; \_; \_; \Delta'\rangle$, and Q'' and Δ'' fresh renamings of Q' and Δ',

$$\mathbf{resolve}(\langle G; [Q, \vec{Q}]; S; \vec{c}; R; \Delta\rangle, n) = \begin{cases} \{n'\} \text{ if unifiable}(Q, Q'') \\ \qquad \text{where } \theta = \text{mostGeneralUnifier}(Q, Q'') \\ \qquad\qquad n' = \langle G; \vec{Q}\theta; S\theta; [\vec{c}; n]; R; \Delta\theta \cup \Delta''\theta\rangle \\ \emptyset \quad \text{otherwise} \end{cases}$$

$$\mathbf{generate}_{P,A}(G) = \bigcup_{(Q \leftarrow \vec{Q}) \in P} \mathbf{resolve}(\langle G; [Q, \vec{Q}]; Q; []; Q \leftarrow \vec{Q}; \emptyset\rangle, \langle G; []; G; []; \_; \emptyset\rangle)$$
$$\cup\ (\text{if } G \in [\![A]\!] \text{ then } \{\langle G; []; G; []; \text{abduction}; \{G\}\rangle\} \text{ else } \emptyset)$$

Fig. 1. Becker and Nanz's algorithm for tabled policy evaluation with proof construction and abduction

4 Becker and Nanz's Algorithm for Tabled Policy Evaluation with Proof Construction and Abduction

This section briefly presents Becker and Nanz's algorithm for tabled policy evaluation extended with proof construction and abduction [2]. This section is based closely on the presentation in their paper.

Becker and Nanz's algorithm appears in Figure 1. It defines a transition system, in which each state is a tuple of the form $(N, \mathit{Ans}, \mathit{Wait})$, where N is a set of nodes, *Ans* is an answer table, and *Wait* is a wait table, as defined below, and where the transitions between states are defined by the transition rules in the upper half of the figure. Disjoint union $\uplus$ is used for pattern matching on sets: N matches $N_1 \uplus N_2$ iff $N = N_1 \cup N_2$ and $N_1 \cap N_2 = \emptyset$.

A *node* is either a *root node* $\langle G\rangle$, where G is an atom, or a tuple of the form $\langle G; \vec{Q}; S; \vec{c}; R; \Delta\rangle$, where G is an atom called the *index* (the goal whose derivation this node is part of), $\vec{Q}$ is a list of subgoals that remain to be solved in the derivation of the goal, S is the partial answer (the instance of G that can be derived using the derivation that this node is part of), $\vec{c}$ is the list of child nodes of this node, R is the rule used to derive this node from its children in the derivation of S, and Δ is the residue (the set of atoms abduced in this derivation). Note that, in the definition of **generate**, we use "abduction" as the name of the rule used to derive an abduced fact. If the list Q of subgoals is empty, the node is called an *answer node* with answer S. Otherwise, it is called a *goal node*, and the first atom in Q is its *current subgoal*. Each answer node is the root of a proof tree; goal nodes (representing queries) are not in proof trees. Selectors for components of nodes are: for $n = \langle G; \vec{Q}; S; \vec{c}; R; \Delta \rangle$, index$(n) = G$, subgoals$(n) = \vec{Q}$, pAns$(n) = S$, children$(n) = \vec{c}$, rule$(n) = R$, and residue$(n) = \Delta$.

An *answer table* is a partial function from atoms to sets of answer nodes. The set $Ans(G)$ contains all answer nodes for the goal G found so far.

A *wait table* is a partial function from atoms to sets of goal nodes. The set $Wait(G)$ contains all those nodes whose current subgoal is waiting for answers from $\langle G \rangle$. Whenever a new answer for $\langle G \rangle$ is produced, the computation of these waiting nodes is resumed.

The auxiliary definitions in the lower half of Figure 1 define the subsumption relation $\preceq$ on nodes and the **resolve** and **generate** functions. Intuitively, if $n \preceq n'$ (read "n is subsumed by n'"), then the answer node n provides no more information than n', so n can be discarded. **resolve**(n, n') takes a goal node n and an answer node n' and combines the current subgoal of n with the answer provided by n' to produce a new node with fewer subgoals. $\mathbf{generate}_{P,A}(G)$ generates a set of nodes for a query $\langle G \rangle$ by resolving G against the rules of policy P, and by abducing G if G is abducible with respect to A. Constructors are not considered in [2], but the algorithm can handle them if the functions for matching and unification are extended appropriately. .

The *initial state* for goal G is $(\{\langle G \rangle\}, \{G \mapsto \emptyset\}, \{G \mapsto \emptyset\})$. A state S is a *final state* iff there is no state S' and such that $S \rightarrow S'$. Given a goal G, start with the initial state for G and apply transition rules repeatedly until a final state is reached. In that final state, $Ans(G)$ represents all derivable instances of G.

5 Analysis Algorithm

The algorithm has three phases. Phase 1 transforms the policy to eliminate `addRule` and `removeRule`. Phase 2 is a modified version of Becker and Nanz's tabling algorithm described above; it produces candidate solutions. Recall that their algorithm attempts to derive a goal from a fixed policy. We modify the tabling algorithm, and transform its input, to enable it to compute sets of policy updates (i.e., administrative operations) needed to derive the goal. However, modifying the tabling algorithm to incorporate a notion of time (i.e., a notion of the order in which updates to the policy are performed, and of the resulting sequence of intermediate policies) would require extensive changes, so we do not do that. Instead, we introduce a third phase that checks all conditions that depend on the order in which administrative operations are performed. These conditions relate to negation, because in the absence of negation, removals are unnecessary, and additions can be done in any order consistent with the logical dependencies that the tabling algorithm already takes into account.

5.1 Phase 1: Elimination of `addRule` and `removeRule`

The policy P' obtained by elimination of `addRule` and `removeRule` from a policy P is not completely equivalent to P—in particular, the state graphs $\mathrm{SG}(P, U_0)$ and $\mathrm{SG}(P', U_0)$ differ, and some kinds of properties, such as availability of permissions, are not preserved. However, P' is equivalent to P in the weaker sense that using P' in place of P in an abductive atom-reachability query does not change the answer to the query. Informally, this is because the answer to such a query depends only on the "upper bounds" of the derivable facts in reachable policies, not on the exact sets of derivable facts in each reachable policy, and this transformation preserves those upper bounds.

Elimination of `removeRule`. The policy $\mathrm{elimRmRule}(P)$ is obtained from P by simply deleting all `removeRule` permission rules (recall that safety allows `removeRule` to appear only in the conclusion of such rules). This eliminates transitions that remove rules defining intensional predicates, and hence eliminates transitions that make intensional predicates smaller. Since negation cannot be applied to intensional predicates, making intensional predicates smaller never makes more facts (including instances of the goal) derivable. Therefore, every instance of the goal that is derivable in some policy reachable from P is derivable in some policy reachable from $\mathrm{elimRmRule}(P)$. Conversely, since $\mathrm{SG}(\mathrm{elimRmRule}(P_0), U_0)$ is a subgraph of $\mathrm{SG}(P_0, U_0)$, every instance of the goal that is derivable in some policy reachable from $\mathrm{elimRmRule}(P)$ is derivable in some policy reachable from P. Therefore, the elimRmRule transformation does not affect the answer to abductive atom-reachability queries.

Elimination of `addRule`. We eliminate `addRule` by replacing `addRule` permission rules (recall that safety allows `addRule` to appear only in the conclusion of such rules) with new rules that use `addFact` to "simulate" the effect of `addRule`. Specifically, the policy $\mathrm{elimAddRule}(P)$ is obtained from P as follows. Let R be an `addRule` permission rule $\mathtt{permit}(U, \mathtt{addRule}(L \texttt{ :- } \vec{L}_1)) \texttt{ :- } \vec{L}_2$ in P. Rule R is replaced with two rules. One rule is the rule pattern in the argument of `addRule`, extended with an additional premise using a fresh extensional predicate $\mathtt{aux}_R$ that is unique to the rule: $L \texttt{ :- } \vec{L}_1, \mathtt{aux}_R(\vec{X})$, where the vector of variables $\vec{X}$ is $\vec{X} = \mathrm{vars}(L \texttt{ :- } \vec{L}_1) \cap (\mathrm{vars}(\{U\}) \cup \mathrm{vars}(\vec{L}_2))$. The other is an `addFact` permission rule that allows the user to add facts to this new predicate: $\mathtt{permit}(U, \mathtt{addFact}(\mathtt{aux}_R(\vec{X}))) \texttt{ :- } \vec{L}_2$. The auxiliary predicate $\mathtt{aux}_R$ keeps track of which instances of the rule pattern have been added. Recall from Section 2 that users are permitted to instantiate variables in the rule pattern when adding a rule. Note that users must instantiate variables that appear in the rest of the `addRule` permission rule, i.e., in $\mathrm{vars}(\{U\}) \cup \mathrm{vars}(\vec{L}_2)$, because if those variables are not grounded, the `permit` fact necessary to add the rule will not be derivable using rule R. Therefore, each fact in $\mathtt{aux}_R$ records the values of those variables. In other words, an `addRule` transition t in $\mathrm{SG}(P_0, U_0)$ in which the user adds an instance of the rule pattern with $\vec{X}$ instantiated with $\vec{c}$ is "simulated" in $\mathrm{SG}(\mathrm{elimAddRule}(P_0), U_0)$ by an `addFact` transition t that adds $\mathtt{aux}_R(\vec{c})$.

Note that $\mathrm{SG}(P_0, U_0)$ also contains transitions t' that are similar to t except that the user performs additional specialization of the rule pattern by instantiating additional variables in the rule pattern or adding premises to it. Those transitions are eliminated by this transformation, in other words, there are no corresponding transitions in $\mathrm{SG}(\mathrm{elimAddRule}(P_0), U_0)$. This is sound, because those transitions lead to policies in which the intensional predicate p that appears in literal L (i.e., L is $p(\ldots)$) is smaller, and as argued above, since negation cannot be applied to intensional predicates, eliminating transitions that lead to smaller intensional predicates does not affect the answer to abductive atom-reachability queries.

Applying this transformation to a policy satisfying the fixed administrative policy requirement produces a policy containing no higher-order administrative permission rules.

From the above arguments, we conclude: For every policy P_0, set U_0 of users, and atom a not in excludedAtoms, $\mathrm{SG}(P_0, U_0)$ contains a policy P with $a \in [\![P]\!]$ iff $\mathrm{SG}(\mathrm{elimAddRule}(\mathrm{elimRmRule}(P_0)), U_0)$ contains a policy P' with $a \in [\![P']\!]$, where excludedAtoms is the set of atoms of the form $\mathtt{permit}(\ldots, \mathtt{addRule}(\ldots))$, $\mathtt{permit}(\ldots, \mathtt{removeRule}(\ldots))$, $\mathtt{aux}_R(\ldots)$, or $\mathtt{permit}(\ldots, \mathtt{addFact}(\mathtt{aux}_R(\ldots)))$. From this, it is easy to show that answers to abductive atom-reachability queries are preserved by this transformation. Subsequent phases of the algorithm analyze the policy $\mathrm{elimAddRule}(\mathrm{elimRmRule}(P_0))$.

5.2 Phase 2: Tabled Policy Evaluation

Phase 2 is a modified version of Becker and Nanz's algorithm. It considers three ways to satisfy a positive subgoal: through an inference rule, through addition of a fact (using an `addFact` permission rule), and through abduction (i.e., by assumption that the subgoal holds in the initial policy and still holds when the rule containing it as a premise is evaluated).

To allow the algorithm to explore addition of facts as a way to satisfy positive subgoals, without directly modifying the algorithm, we transform `addFact` permission rules into ordinary inference rules. Specifically, each `addFact` permission rule $\mathtt{permit}(U, \mathtt{addFact}(a))$ `:-` $\vec{L}$ is replaced with the rule a `:-` $\vec{L}, \mathtt{u0}(U)$. The transformation also introduces a new extensional predicate `u0` and, for each $u \in U_0$, the fact $\mathtt{u0}(u)$ is added to the policy. This transformation changes the meaning of the policy: the transformed rule means that a necessarily holds when $\vec{L}$ holds, while the original `addFact` permission rule means that a might (or might not) be added by an administrator when $\vec{L}$ holds. This difference is significant if a appears negatively in a premise of some rule. This change in meaning is acceptable in phase 2, because phase 2 does not attempt to detect conflicts between negative subgoals and added facts. This change in the meanings of rules used in phase 2 does not affect the detection of such conflicts in phase 3.

The algorithm considers two ways to satisfy a negative subgoal: through removal of a fact (using a `removeFact` permission rule) and through abduction (i.e., by assumption that the negative subgoal holds in the initial policy and still holds when the rule containing it as a premise is evaluated).

To allow the algorithm to explore removal of facts as a way to satisfy negative subgoals, `removeFact` permission rules are transformed into ordinary inference rules with negative conclusions. Specifically, each `removeFact` permission rule $\texttt{permit}(U, \texttt{removeFact}(a))$ `:-` $\vec{L}$ is replaced with the rule $!a$ `:-` $\vec{L}, \texttt{u0}(U)$.

Let elimAddRmFact(P) denote the policy obtained from P by transforming `addFact` and `removeFact` rules as described above. An *administrative node* (or "admin node", for short) is a node n such that rule(n) is a transformed `addFact` or `removeFact` permission rule. isAdmin(n) holds iff n is an administrative node.

The algorithm can abduce a negative extensional literal $!a$ when this is consistent with the initial policy, in other words, when a is not in P_0. To enable this, in the definition of **generate**, we replace "$G \in [\![A]\!]$" with "$G \in [\![A]\!] \vee (\exists a \in \text{Atom}_{ex}.\ a \notin P_0 \wedge G$ is $!a)$", where Atom_{ex} is the set of extensional atoms. If a is not ground, disequalities in d_{init} in phase 3 will ensure that the solution includes only instances of a that are not in P_0.

The tabling algorithm treats the negation symbol "!" as part of the predicate name; in other words, it treats p and $!p$ as unrelated predicates. Phase 3 interprets "!" as negation and checks appropriate consistency conditions.

The tabling algorithm explores all derivations for a goal except that the subsumption check in transition rule (ans) in Figure 1 prevents use of the derivation represented by answer node n from being added to the answer table and thereby used as a sub-derivation of a larger derivation if the partial answer in n is subsumed by the partial answer in an answer node n' that is already in the answer table. However, the larger derivation that uses n' as a derivation of a subgoal might turn out to be infeasible (i.e., have unsatisfiable ordering constraints) in phase 3, while the larger derivation that uses n as a derivation of that subgoal might turn out to be feasible. We adopt the simplest approach to overcome this problem: we replace the subsumption relation $\preceq$ in transition rule (ans) with the α-equality relation $=_\alpha$, causing the tabling algorithm to explore all derivations of goals. α-equality means equality modulo renaming of variables that do not appear in the top-level goal G_0.

An undesired side-effect of this change is that the algorithm may get stuck in a cycle in which it repeatedly uses some derivation of a goal as a sub-derivation of a larger derivation of the same goal. Exploring the latter derivation is unnecessary, because it will be subjected in phase 3 to the same or more constraints as the former derivation. Therefore, we modify the definition of **resolve** as follows, so that the algorithm does not generate a node n' corresponding to the latter derivation: we replace unifiable(Q, Q'') with unifiable(Q, Q'') $\wedge$ noCyclicDeriv(n'), where

$$\begin{aligned}&\text{noCyclicDeriv}(n') = \nexists d \in \text{proof}(n').\ \text{isAns}(d)\\ &\quad \wedge \langle \text{index}(d), \text{pAns}(d), \text{residue}(d)\rangle =_\alpha \langle \text{index}(n'), \text{pAns}(n'), \text{residue}(n')\rangle\end{aligned}$$

where the *proof* of a node n, denoted proof(n), is the set of nodes in the proof graph rooted at node n, i.e., $\text{proof}(n) = \{n\} \cup \bigcup_{n' \in \text{children}(n)} \text{proof}(n')$, and where isAns($n$) holds iff n is an answer node. noCyclicDeriv(n') does not check whether rule(n') = rule(d) or subgoals(n') = subgoals(d), because exploration of n' is unnecessary, by the above argument, regardless of the values of rule(n') and subgoals(n').

We extend the algorithm to store the *partial answer substitution*, denoted $\theta_{\mathrm{pa}}(n)$, in each node n. This is the substitution that, when applied to index(n), produces pAns(n). In the **generate** function, the θ_{pa} component in both nodes passed to **resolve** is the empty substitution. In the **resolve** function, $\theta_{\mathrm{pa}}(n')$ is $\theta \circ \theta_{\mathrm{fr}} \circ \theta_{\mathrm{pa}}(n_1)$, where θ_{fr} is the substitution that performs the fresh renaming of Q' and Δ', and n_1 denotes the first argument to **resolve**.

In summary, given an abductive atom-reachability query of the form in Section 3, phase 2 applies the tabling algorithm, modified as described above, to the policy elimAddRmFact(elimAddRule(elimRmRule(P_0))) with the given goal G_0 and specification A of abducible atoms.

5.3 Phase 3: Ordering Constraints

Phase 3 considers constraints on the execution order of administrative operations. The ordering must ensure that, for each administrative node or goal node n, (a) each administrative operation n' used to derive n occurs before n (this is a "dependence constraint") and its effect is not undone by a conflicting operation that occurs between n' and n (this is an "interference-freedom constraint"), and (b) each assumption about the initial policy on which n relies is not undone by an operation that occurs before n (this is an "interference-freedom constraint").

The overall ordering constraint is represented as a Boolean combination of labeled ordering edges. A labeled ordering edge is a tuple $\langle n, n', D\rangle$, where the label D is a conjunction of tuple disequalities or false, with the interpretation: n must precede n', unless D holds. if D holds, then n and n' operate on distinct atoms, so they commute, so the relative order of n and n' is unimportant.

Pseudocode for phase 3 appears in Figures 2 and 3. The algorithm generates the overall ordering constraint, puts the Boolean expression in DNF, and checks, for each clause c, whether the generated ordering constraints can be satisfied, i.e., whether they are acyclic. If so, the disequalities labeling the ordering constraints do not need to be included in the solution. However, if the generated ordering constraints are cyclic, then the algorithm removes a minimal set of ordering constraints to make the remaining ordering constraints acyclic, and includes the disequalities that label the removed ordering constraints in the solution. After constraints have been checked, negative literals are removed from the residue; this is acceptable, because the problem definition asks for a representation of only minimal-residue ground solutions, not all ground solutions The algorithm can easily be extended to return a *plan* (a sequence of administrative operations that leads to the goal) for each solution.

Repeated Administrative Operations. Tabling re-uses nodes, including, in our setting, administrative nodes. This makes the analysis more efficient and avoids unnecessary repetition of administrative operations in plans. However, in some cases, administrative operations need to be repeated; for example, it might be necessary to add a fact, remove it, and then add it again, in order to reach the goal. The current version of our algorithm cannot generate plans with repeated administrative operations, but it does identify when repeated operations might be necessary, using the function mightNeedRepeatedOp, and returns a message

```
$solutions = \emptyset$
for each node $n_g \in Ans(G)$
  // consistency constraint: disequalities that ensure consistency of initial state,
  // i.e., positive literals are distinct from negative literals.
  $d_{\mathrm{init}} = \bigwedge\{\mathrm{args}(a) \neq \mathrm{args}(b) \mid a \in \mathrm{facts}(P_0) \cup \mathrm{residue}(n_g) \wedge !b \in \mathrm{residue}(n_g) \wedge \mathrm{unifiable}(a, b)\}$
  if $\neg$satisfiable($d_{\mathrm{init}}$)
    continue
  endif
  $O$ = orderingConstraints($n_g$)
  if ($\exists$ clause $c$ in $O$. the ordering constraints in $c$ are acyclic)
    // the ordering constraints for $n_g$ are satisfiable without imposing disequalities.
    // intersect residue with Atom$_{ex}$ (the extensional atoms) to remove negative literals.
    $solutions = solutions \cup \{\langle \mathrm{pAns}(n_g), \mathrm{residue}(n_g) \cap \mathrm{Atom}_{ex}, d_{\mathrm{init}}\rangle\}$
  else
    // the ordering constraints for $n_g$ are not satisfiable in general, but might
    // be satisfiable if disequalities are imposed to ensure that some
    // administrative operations operate on distinct atoms and therefore commute.
    for each clause $c$ in $O$
      if mightNeedRepeatedOp($c, n_g$)
        // the current version of the algorithm does not support repeated operations
        return "repeated operations might be needed"
      endif
      $D_{\mathrm{ord}} = \emptyset$
      // $c$ is a conjunction (treated as a set) of labeled ordering constraints.
      // remove some ordering constraints $F$ from $c$ to make the remaining ordering
      // constraints acyclic, and insert in $D_{\mathrm{ord}}$ the conjunction $d$ of $d_{\mathrm{init}}$ and the
      // disequalities labeling the removed ordering constraints.
      $Cyc$ = set of all cycles in ordering constraints for clause $c$
      $FAS = \{F \mid F$ contains one edge selected from each cycle in $Cyc\}$
      // $smFAS$ is the set of $\subseteq$-minimal feedback arc sets (FASs) for clause $c$
      $smFAS = \{F \in FAS \mid \nexists F' \in FAS.\ F' \subset F\}$
      for each $F$ in $smFAS$
        $d = d_{\mathrm{init}} \wedge \bigwedge\{d' \mid \langle n_1, n_2, d'\rangle \in F\}$
        if satisfiable($d$) $\wedge \neg(\exists d' \in D_{\mathrm{ord}}.\ d' \subseteq d)$
          $D_{\mathrm{ord}} = D_{\mathrm{ord}} \cup \{d\}$
        endif
      endfor
      $solutions = solutions \cup \{\langle \mathrm{pAns}(n_g), \mathrm{residue}(n_g) \cap \mathrm{Atom}_{ex}, d\rangle \mid d \in D_{\mathrm{ord}}\}$
    endfor
  endif
endfor
return $solutions$
```

Fig. 2. Pseudo-code for Phase 3

indicating this. Specifically, mightNeedRepeatedOp(c, n_g) returns true if some node n in c is a child of multiple nodes in proof(n_g); in such cases, it might be necessary to replace n with multiple nodes, one for each parent, in order to satisfy the ordering constraints. To achieve this, the algorithm can be modified so that, if mightNeedRepeatedOp returns true, the algorithm re-runs phases 2 and 3 but this time constructs new answer nodes, instead of re-using tabled answers, for the nodes identified by mightNeedRepeatedOp as possibly needing to be repeated.

```
function orderingConstraints($n_g$)
 $\theta = \theta_{\mathrm{pa}}(n_g)$
 // dependence constraint: an admin node $n_s$ that supports $n$ must occur before $n$.
 $O_{\mathrm{dep}} = \bigwedge\{\langle n_s, n, \mathrm{false}\rangle \mid n \in \mathrm{proof}(n_g) \wedge (\mathrm{isAdmin}(n) \vee n = n_g) \wedge n_s \in \mathrm{adminSupport}(n)\}$
 // all of the constraints below are interference-freedom constraints.
 // a removeFact node $n_r$ that removes a supporting initial fact of a node $n$ must occur
 // after $n$.
 $O_{\text{rm-init}} = \bigwedge\{\langle n, n_r, \mathrm{args}(a)\theta \neq \mathrm{args}(\mathrm{pAns}(n_r))\theta\rangle \mid$
          $n \in \mathrm{proof}(n_g) \wedge (\mathrm{isAdmin}(n) \vee n = n_g) \wedge n_r \in \mathrm{proof}(n_g) \wedge \mathrm{isRmFact}(n_r)$
          $\wedge\, n \neq n_r \wedge a \in \mathrm{supportingInitFact}(n) \wedge \mathrm{unifiable}(!a, \mathrm{pAns}(n_r))\}$
 // an addFact node $n_a$ that adds a fact whose negation is a supporting initial fact
 // of a node $n$ must occur after $n$.
 $O_{\text{add-init}} = \bigwedge\{\langle n, n_a, \mathrm{args}(a)\theta \neq \mathrm{args}(\mathrm{pAns}(n_a))\theta\rangle \mid$
          $n \in \mathrm{proof}(n_g) \wedge (\mathrm{isAdmin}(n) \vee n = n_g) \wedge n_a \in \mathrm{proof}(n_g) \wedge \mathrm{isAddFact}(n_a)$
          $\wedge\, n \neq n_a \wedge\, !a \in \mathrm{supportingInitFact}(n) \wedge \mathrm{unifiable}(a, \mathrm{pAns}(n_a))\}$
 // an addFact node $n_a$ that adds a supporting removed fact of a node $n$ must occur
 // either before the removal of that fact or after $n$.
 $O_{\text{add-rmvd}} =$
 $\bigwedge\{\langle n_a, n_r, \mathrm{args}(\mathrm{pAns}(n_a))\theta \neq \mathrm{args}(\mathrm{pAns}(n_r))\theta\rangle \vee \langle n, n_a, \mathrm{args}(\mathrm{pAns}(n_a))\theta \neq \mathrm{args}(\mathrm{pAns}(n_r))\theta\rangle \mid$
      $n \in \mathrm{proof}(n_g) \wedge (\mathrm{isAdmin}(n) \vee n = n_g) \wedge n_r \in \mathrm{adminSupport}(n) \wedge \mathrm{isRmFact}(n_r)$
      $\wedge\, n_a \in \mathrm{proof}(n_g) \wedge \mathrm{isAddFact}(n_a) \wedge n \neq n_a \wedge \mathrm{unifiable}(!\mathrm{pAns}(n_a), \mathrm{pAns}(n_r))\}$
 // a removeFact node $n_r$ that removes a supporting added fact of a node $n$ must occur
 // either before the addition of that fact or after $n$
 $O_{\text{rm-added}} =$
 $\bigwedge\{\langle n_r, n_a, \mathrm{args}(\mathrm{pAns}(n_a))\theta \neq \mathrm{args}(\mathrm{pAns}(n_r))\theta\rangle \vee \langle n, n_r, \mathrm{args}(\mathrm{pAns}(n_a))\theta \neq \mathrm{args}(\mathrm{pAns}(n_r))\theta\rangle \mid$
      $n \in \mathrm{proof}(n_g) \wedge (\mathrm{isAdmin}(n) \vee n = n_g) \wedge n_a \in \mathrm{adminSupport}(n) \wedge \mathrm{isAddFact}(n_a)$
      $\wedge\, n_r \in \mathrm{proof}(n_g) \wedge \mathrm{isRmFact}(n_r) \wedge n \neq n_r \wedge \mathrm{unifiable}(!\mathrm{pAns}(n_a), \mathrm{pAns}(n_r))\}$
 // conjoin all ordering constraints and convert the formula to disjunctive normal form.
 $O = \mathrm{DNF}(O_{\mathrm{dep}} \wedge O_{\text{rm-init}} \wedge O_{\text{add-init}} \wedge O_{\text{add-rmvd}} \wedge O_{\text{rm-added}})$
 // for each clause $c$ of $O$, merge labeled ordering constraints for the same pair of nodes.
 for each clause $c$ in $O$
    while there exist $n_1, n_2, D, D'$ such that $c$ contains $\langle n_1, n_2, D\rangle$ and $\langle n_1, n_2, D'\rangle$
       replace $\langle n_1, n_2, D\rangle$ and $\langle n_1, n_2, D'\rangle$ with $\langle n_1, n_2, D \wedge D'\rangle$ in $c$
    endwhile
 endfor
 return $O$
```

args(a) = a tuple containing the arguments of atom a

$\mathrm{support}(n) = \{n' \in \mathrm{proof}(n) \mid \mathrm{isAns}(n') \wedge n' \neq n$
$\neg\exists n_a.\mathrm{isAdmin}(n_a) \wedge \mathrm{descendant}(n, n_a) \wedge \mathrm{descendant}(n_a, n')\}$

$\mathrm{adminSupport}(n) = \{n' \in \mathrm{support}(n) \mid \mathrm{isAdmin}(n')\}$

$\mathrm{supportingInitFact}(n) = \{\mathrm{pAns}(n') \mid n' \in \mathrm{support}(n)$
$\wedge\, (\mathrm{rule}(n') \in \mathrm{facts}(P_0) \vee \mathrm{rule}(n') = \mathrm{abduction})\}$

Fig. 3. Ordering constraints for an answer node n_g

5.4 Implementation and Experience

We implemented the analysis algorithm in 5000 lines of OCaml and applied it to part of the policy P_{HCN} for our healthcare network case study with 30 administrative permission rules. We included facts about a few prototypical users in P_{HCN}: `fpo1`, a member of `facility_po(sb_hosp)`; `clin1`, a clinician at `sb_hosp`;

and `user1`, a user with no roles. A sample abductive atom-reachability query that we evaluated has $P_0 = P_{\mathrm{HCN}}$, $U_0 = \{$`fpo1`, `user1`$\}$, $A = \{$`memberOf(User, workgroup(WG, sb_hosp, team))`$\}$, and $G_0 =$ `memberOf(GoalUser, wgHead( cardioTeam, sb_hosp))`. The analysis takes about 1.5 seconds, generates 2352 nodes, and returns five solutions. For example, one solution has partial answer `memberOf(GoalUser, wgHead(cardioTeam, sb_hosp))`, residue $\{$`memberOf( GoalUser, workgroup(cardioTeam, sb_hosp, team))`$\}$, and tuple disequality $\langle$`GoalUser`$\rangle \neq \langle$`fpo1`$\rangle$. The disequality reflects that `fpo1` can appoint himself to the `hr_manager(sb_hosp)` role, can then appoint himself and other users as members of `cardioTeam`, and can then appoint other users as team head, but cannot then appoint himself as team head, due to the negative premise in the sample rules at the end of Section 2.

References

1. Becker, M.Y.: Specification and analysis of dynamic authorisation policies. In: Proc. 22nd IEEE Computer Security Foundations Symposium (CSF), pp. 203–217 (2009)
2. Becker, M.Y., Nanz, S.: The Role of Abduction in Declarative Authorization Policies. In: Hudak, P., Warren, D.S. (eds.) PADL 2008. LNCS, vol. 4902, pp. 84–99. Springer, Heidelberg (2008)
3. Becker, M.Y., Nanz, S.: A logic for state-modifying authorization policies. ACM Transactions on Information and System Security 13(3) (2010)
4. 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(4), 242–255 (2008)
5. Li, N., Tripunitara, M.V.: Security analysis in role-based access control. ACM Transactions on Information and System Security 9(4), 391–420 (2006)
6. 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)
7. Stoller, S.D., Yang, P., Gofman, M., Ramakrishnan, C.R.: Symbolic reachability analysis for parameterized administrative role based access control. Computers & Security 30(2-3), 148–164 (2011)

Towards Detection of Botnet Communication through Social Media by Monitoring User Activity

Pieter Burghouwt, Marcel Spruit, and Henk Sips

Delft University of Technology, The Netherlands
{P.Burghouwt,H.J.Sips}@tudelft.nl, M.E.M.Spruit@hhs.nl

Abstract. A new generation of botnets abuses popular social media like Twitter, Facebook, and Youtube as Command and Control channel. This challenges the detection of Command and Control traffic, because traditional IDS approaches, based on statistical flow anomalies, protocol anomalies, payload signatures, and server blacklists, do not work in this case. In this paper we introduce a new detection mechanism that measures the causal relationship between network traffic and human activity, like mouse clicks or keyboard strokes. Communication with social media that is not assignably caused by human activity, is classified as anomalous. We explore both theoretically and experimentally this detection mechanism by a case study, with Twitter.com as a Command and Control channel, and demonstrate successful real time detection of botnet Command and Control traffic.

1 Introduction

Social media, like Twitter, Facebook, and Youtube create a new communication opportunity for malicious botnets. A Command and Control (C&C) channel is crucial for a botnet. Bots need regular instructions from the botmaster, for example to initiate or synchronize an attack, upload harvested data, or update the malware [17]. A large number of countermeasures has been developed and deployed against C&C in general. Botnets react to these measures by using new communication mechanisms that are harder to detect and repress [18]. With the arrival of social media based C&C, there are now 4 major C&C-mechanisms:

1. IRC is a simple and proven C&C technique. It is attractive for the botmaster, because of its simple management, with a real time or "tight" control over the botnet through a permanently established connection. However especially for this type of C&C, many countermeasures have been developed. An example is the work of Cook et al. [2].
2. A more sophisticated C&C-mechanism uses peer-to-peer communication, based on protocols like Kademlia [3]. The intrinsic decentralized structure is difficult to counter and the use of peer-to-peer protocols is widespread, due to popular file sharing and communication protocols, like Bittorrent and Skype. However the absence of centralized control makes management

complex, and due to firewalls and NATs, many peers cannot receive incoming connections, resulting in network of which the availability and stability highly depends on a limited number of nodes [12].

3. Another C&C-mechanism uses HTTP to exchange information. The popularity of HTTP makes anomaly detection difficult. In addition, many botnets decentralize the HTTP-service by fast fluxing A-records, IP-addresses of DNS-servers, or even domain names [14]. This makes it difficult to bring the service down by IP or domain blacklisting [15].
4. A rather new C&C-mechanism uses social media for C&C. A botmaster posts instructions as messages on a popular social medium, like Twitter or Facebook. Bots fetch the instructions by regularly polling certain pages. Examples of such botnets are: Trojan.Whitewell, that uses Facebook [11], TwitterNET, that uses Twitter [13], and Trojan 2.0, that uses Google groups [6].

If a botnet imitates the communication patterns of normal users that visit a popular social medium, detection will become very difficult with conventional network IDS-techniques, because there are no anomalous addresses, domain names, protocols, or ports involved and a large fraction of the daily legitimate traffic of normal computers consists of visits to social media. A further increase of the C&C invisibility is possible by steganographic techniques, to hide the commands in apparently normal messages, account flux, by making the bots visit different accounts or even different media, and SSL/TLS encryption, to impede content inspection in the network. Many social media offer already HTTPS access as an alternative to the still popular unencrypted HTTP access.

Difficult detection is not the only reason that social media based C&Cs have the potential to become the most important botnet control mechanism on the Internet. Other beneficial properties from the perspective of the botmaster are: simple implementation, simple management of applications that read from and write to social media, scalability, and high availability of the social media services.

In this paper we address the detection of social media based C&C-traffic, by introducing a detection mechanism that measures causality between user activity and network traffic. The presence or absence of certain key strokes and mouse clicks is used to determine if network traffic is user- or botnet- originated.

Section 2 gives an overview of related existing work in the areas of botnet C&C-detection and the observation of user activity in the detection process.

Section 3 introduces a detection mechanism that uses mouse clicks and key strokes as a proof of user commitment to visit a social medium. If egress traffic starts within a small time window directly after a user event, it is classified as user intended. Traffic outside this time window consists of both legitimate traffic that is indirectly induced by user events, and botnet C&C traffic that is not induced by any user event.

The detection mechanism is elaborated with Twitter.com as representative example of a social medium.

Section 4 discusses detection improvements, related with legal automatic traffic and potential evasion techniques.

2 Related Work

Since causality detection is partly based on measurement of network traffic, we start with some examples of existing detection mechanisms that detect C&C-traffic by passively monitoring the network. Examples are: recognition of known command strings [5], clustering similar traffic for correlation with observed attacks [7], observation of new repeating destination patterns or atoms [4], and observation of fast changes in DNS-records [9]. However, with botnet traffic that imitates user originated visits to popular social websites, these detection approaches have limited results, due to the close resemblance to legal traffic.

Honeypots [16] can detect botnet traffic without false positives. However, after detection a signature of the malicious traffic must be derived and distributed for misuse detection. An example is the blacklisting of a suspicious user account. This makes the total system not real time with the risk of too much delay between the detected and actual properties of the traffic.

Another aspect, which is related with the detection mechanism, to be presented in this paper, is the measurement of user commitment. A CAPTCHA, as introduced by Ahn et al. [1], is a popular test to remotely distinguish between computer programs and users. CAPTCHA's are often used in Web 2.0 applications, to prevent automated account creation. Obviously an intensive use of CAPTCHA's does not promote the accessibility of a website. As most users do not create a new account every five minutes, the nuisance is limited in that case. However, the use of CAPTCHA's for every social medium visit, results in an unacceptable burden. Of course cryptographic signatures can be used to relate later traffic with an initial CAPTCHA verification, but it can result in a complex trust mechanism. Vo et al. [20] propose a system that authenticates physical computers with social media accounts after an initial CAPTCHA test. However once authenticated, the system does not really prevent botnets to use the account or visit public pages of users.

Gummadi et al. propose NAB, Not-a-Bot [8]. The system uses TPM-backed attestations of user activity, that are sent with web- and email requests to an external verifier, where the decision is made if the traffic is legitimate or not. Gummadi et al. focus primarily on the implementation of the attestation mechanism and give no details of the time-related detection mechanism. Verification of the detector performance is done indirectly by combining offline keyboard- and mouse logs with network traces. It shows that NAB can reduce the rate of intense bot-generated traffic, like DoS and spam. NAB, can be seen as complementary to our work, by delivering a possible practical implementation of a trusted agent that attests user events to the actual detector.

Kartaltepe et al. [10] present a systematic overview of the evolution in social media botnets and approaches in countermeasures. Our detection approach can be classified in their work as the detection of a client-sided self-concealing process. The absence of human-computer interaction computer defines a process as potential self concealing. Kartaltpepe et al. do not elaborate on detection mechanisms that use this self concealing property.

3 Detection Principle

Network traffic does not occur spontaneously. Something triggers a traffic flow. In the case of a visit to a social medium, the trigger is usually a keyboard or mouse event, caused by a human user. However if a bot visits a social medium to fetch new instructions, or upload harvested information, the traffic is not triggered by user events, but by internal state changes of the malware. This allows for detection of botnet traffic to social media by the absence of user events that could potentially have triggered the traffic.

Figure 1 shows a schematic overview of our proposed detection system. Network traffic (3) is captured from an inserted network bridge. If a traffic flow is initiated to a social medium, without preceding keyboard events (1) or mouse events (2), the flow is classified as potential bot-originated by the causality detector.

There are several ways to implement the taps to intercept the keyboard and mouse events. For example by hooks in the operating system of the client computer that catches the events. Another possibility is the insertion of a hardware device that monitors the signals on the bus from the user devices to the client computer. In case of a virtual client computer, the tap can even be implemented in the hypervisor. The possibility of implementing taps, causality detection, and bridge completely outside the client computer results in a detector that is less visible and more resistant against direct attacks.

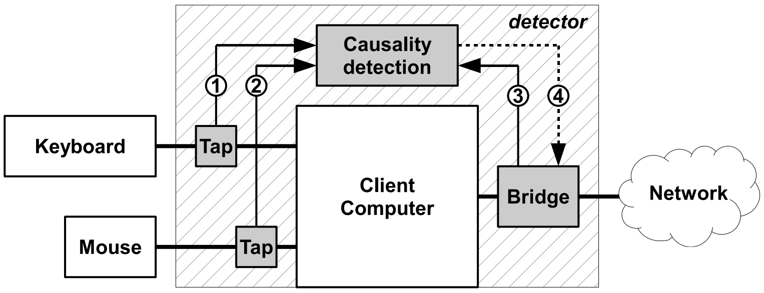


Fig. 1. Schematic overview of a detector which correlates activity of keyboard (1) and mouse (2), with captured network traffic (3)

The causality detection works with a small time window that starts directly after a user event and discriminates between flows that are caused or not caused by the user event. It does not depend on known signatures; hence communication of zero day-bots can also be detected. Moreover, in contrast to most anomaly detection mechanisms, the classification is real time, resulting in the potential block of a malicious flow at the moment of the first egress packet.

In the remainder of this section we elaborate on the detection algorithm, estimate the performance, and show by experiment that it really detects botnet traffic. For the ease of the explanation, we focus on Twitter.com as a representative example of a popular social medium, hence all results can be extended to other social media. We assume that all legal traffic to Twitter.com is directly caused by user events and all bot-originated traffic is not synchronized with user activity. In Section 4 we discuss important exceptions to these assumptions, like legitimate automatic traffic and detector evasion by synchronization with user activity.

3.1 Detection of Botnet Traffic to Twitter.com

The detection algorithm proposed in this section is linked to browser access to Twitter.com with Internet Explorer and Firefox. In Section 4 we discuss alternative client scenarios to communicate with Twitter. In the browser scenario we identify three specific user events that can exclusively trigger Twitter traffic:

- Left mouse click, typically on Twitter link;
- Enter key, typically during login or completion of message or “Tweet”;
- F5-key to reload a page

Normal Twitter traffic always starts with the request of an object from Twitter.com. Examples of requested Twitter.com-objects are: a timeline with tweets, a search instruction, or a login. Directly after the loading of the first html-formatted object, additional objects, like scripts, media, advertisements, and tracking objects are loaded from other domains, like Twimg.com and Google.com. Bots that use Twitter as a C&C-channel must start with a request of a Twitter.com-object, because other sequences are unusual and raise suspicion. The Twitter.com-object can directly contain C&C instructions or results, or link to other objects with C&C-related content. Our detection algorithm tests for the presence of relevant user events within a certain time frame, just before an egress flow to Twitter.com is initiated. Figure 2 shows the relevant timing.

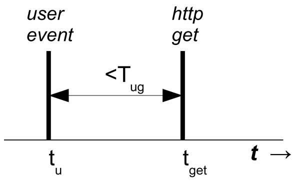


Fig. 2. Time diagram of a user event that directly results in a visit to Twitter.com. Parameters are defined in Table 1.

A Twitter.com flow is classified as non-bot if the flow is user induced, as illustrated by the implication:

$$t_{get} - t_u < T_{ug} \rightarrow user\, induced \tag{1}$$

A complicating factor is DNS. If the client cache does not contain a valid DNS record of Twitter.com, a DNS lookup will take place between the user event and the actual visit to Twitter.com. Figure 3 shows the relevant timing for the second more complex scenario. The Twitter.com flow with is now classified as non-bot if the flow is user induced, as illustrated by the implication:

$$(t_{dnsq} - t_u < T_{ud}) \wedge (t_{dnsa} - t_{dnsq} < T_{dd}) \wedge (t_{get} - t_{dnsa} < T_{dg}) \rightarrow user\, induced \tag{2}$$

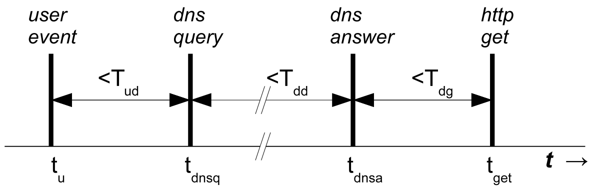


Fig. 3. Time diagram of a user events that results in a DNS-lookup, followed by a visit to Twitter.com. Parameters are defined in Table 1.

Table 1. Description of the parameters, used in the algorithm

Parameter	Description
t_u	user event(release in left mouse click, press of F5 or Enter key)
t_{get}	HTTP GET-request to Twitter.com
t_{dnsq}	DNS query to resolve Twitter.com
t_{dnsa}	DNS answer to resolve Twitter.com
T_{ug}	maximum permitted time between t_u and t_{get}
T_{ud}	maximum permitted time between t_u and t_{dnsq}
T_{dd}	maximum permitted time between $t_{dnsq}$and t_{dnsa}
T_{dg}	maximum permitted time between t_{dnsa} and t_{get}

Detection performance depends largely on a proper value of the defined windows T_{ug}, T_{ud}, and T_{dg}. Large windows increase the probability that egress botnet traffic is coincidentally initiated inside a window, with a false negative as a result. Conversely small windows increase the probability that user-induced traffic starts just after a window, with a false positive as a result. The size of T_{dd} depends on the maximum expected response time of the DNS-server. The value of T_{dd} is not critical in the detection process, because it is not a response time of the potentially infected client computer. Moreover causality between DNS request and DNS response can also be determined by the UDP client port or DNS payload instead of timing.

3.2 Empirical Estimation of Optimal Time Windows

The optimal value of T_{ug}, T_{ud}, and T_{dg} is the worst case response time of the involved client computer. This value is determined by many factors, like the amount of processing involved in the response, hardware capabilities, real time properties of the operating system, and the actual load of the system. We estimated the values by measuring Twitter-related response times on a HP DC7900, a main stream Personal Computer, in a test set up, as in Figure 1, but with a logging facility instead of a causality detector. The keyboard/mouse taps were implemented by a separate microcontroller board, inserted in the PS2-cables of both the mouse and the keyboard. Captured events were signaled to the logging device over a high speed serial connection. All other functions were implemented in a fast computer with a Linux operating system.

We repeatedly visited the same timeline of a Twitter account with a Firefox browser under Windows XP. Response times were measured for 3 different user events, both under idle and high load conditions. The latter condition was established by simultaneous downloads of 5Mb/s+ traffic from 10+ sources. The three user events were: F5 reload, mouse click (M1), and mouse click with empty DNS-cache (M2). Every measurement was repeated 100 times. The average and extreme response times for each scenario are presented in Table 2.

Table 2. Summary of measured response times of Twitter.com visits with a HP DC7900 Windows XP PC. F5 is the response to a F5-reload, M1 is the response to a Mouse click with preloaded DNS-cache and M2 with empty DNS-cache.

		Idle			High concurrent load		
Case	Interval	t_{min}(ms)	t_{av}(ms)	t_{max}(ms)	t_{min}(ms)	t_{av}(ms)	t_{max}(ms)
F5	t_{get}-$t_{u,key}$	0.1	3	163	0.2	15	115
M1	t_{get}-$t_{u,mouse}$	16	17.6	32.1	16	27	45
M2	t_{dnsq}-$t_{u,mouse}$	4.7	5.8	7.1	5.3	14	21
	t_{get}-t_{dnsa}	0.3	0.6	2.7	0.3	3.2	7.3

Only 2 of the 600 measured response times were above 100ms, with a maximum response time of 163ms.

3.3 Theoretical Performance of the Detector

To determine the Detection Rate (DR) and False Positive Rate (FPR), we start by an analysis of visits to Twitter.com with a valid DNS cache. This simple scenario uses only one time window $T_w = T_{ug}$. At the end of the next subsection we will extend the analysis for the more complex scenario with DNS-traffic.

Only a Twitter.com flow that starts within the causal window T_w is classified as user-caused and hence normal. All other Twitter.com flows are classified as anomalous. Figure 3 illustrates a generic example of three causal windows of which two are partly overlapping. Overlap can take place if the window size is in the order of magnitude of the minimum time between user events.

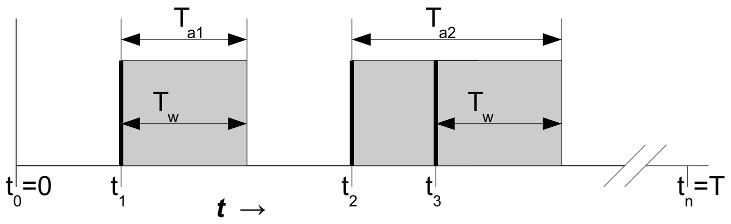


Fig. 4. Example of 3 user events (t_1, t_2, and t_3) with their causal windows with length T_w. Traffic that is initiated outside these windows is classified as anomalous.

Detection Rate. Assume multiple events during some observation interval T and define t_i as the moment the i^th user event is observed. The boundaries t_0 and t_n are not user events, but the respective start and stop time of the observation interval. Let h(t) be a function that has the value of 0 if t is inside a causal window, and the value of 1 if t is outside any of the causal windows:

$$h(t) = \begin{cases} 0 & if\, \exists\, i, 0 \leq t - t_i \leq T_w \\ 1 & if\, \nexists\, i, 0 \leq t - t_i \leq T_w \end{cases} \tag{3}$$

The Detection Rate (DR) is the probability that a bot-originated flow starts outside any of the causal windows and hence is classified as anomalous.

$$\begin{aligned} DR &= P\left(h(t) = 1 | t = random\, start\, of\, C\&C\, flow\right) \\ &= \frac{1}{T}\int_0^T h(t)dt = \frac{1}{T}\sum_{i=1}^{n} H_i \end{aligned} \tag{4}$$

with

$$H_i = \begin{cases} 0 & if\, t_i - t_{i-1} \leq T_w \\ t_i - t_{i-1} - T_w & if\, t_i - t_{i-1} > T_w \end{cases} \tag{5}$$

We define T_{av} as the average time between 2 successive user events during the observation interval. If $T_{av} >> T_w$ the effect of overlapping windows is negligible and Eq. 4 reduces to:

$$DR \approx 1 - \frac{T_w}{T_{av}} \tag{6}$$

Eq. 6 shows that the DR is optimal with a small causal window T_w and a large average time between user events.

To estimate DR in a worst case scenario, we used the reported results of odometric measurements of the Workpace RSI-software among 50000 users during a 4 week period [19]. The work shows a daily peak of 3400 mouse clicks and 11600 key strokes during 2.8 hours. If we assume that 5% or less of the pressed keys is a carriage return or F5, the average time between user events can be calculated: $T_{av} = (2.8 * 3600)\,/\,(3400 + 11600 * 0.05) \approx 2.5s$. Based on 163ms as the maximum measured response time in Table 2, we choose T_w= 200ms, resulting in a detection rate: $DR = 1 - (0.2/2.5) = 0.91$.

In practice the the average DR is expected to be higher, because:

1. the average time between user events over a day is significantly higher than 2.5 seconds, because most computers are only intensely used during a small part of the day[19];
2. the effective frequency of left mouse clicks is lower, because the presented frequency also includes right mouse clicks and double clicks[19].

In the more more complex scenario with a DNS-lookup there are two critical windows: T_{ud} and T_{dg}. By replacing both windows by one window $T_{\mathrm{w}} = T_{\mathrm{ud}}+T_{\mathrm{dg}}$, the DR is calculated by the same math. Based on 21+7.3ms as the maximum measured response time in Table 2, we choose $\mathrm{T_w}$ = 21+7.3 = 30ms, resulting in a DR of 0.99.

False Positive Rate. False Positives are caused by a response time that is larger than the defined causal window $\mathrm{T_w}$. Eq. 7 expresses the resulting False Positive Rate (FPR). It depends on the distribution of the response time.

$$\begin{aligned} FPR &= P\left(t_{get} - t_u > T_w \,|\, user\, triggered\, flow\right) \\ &= 1 - \int_0^{T_w} p(t)dt \end{aligned} \tag{7}$$

The function p(t) is the probability distribution of the response time . The observations in Section 3.2 did not lead to a well-defined model for this distribution, because many factors influence the response time of the computer. However it is considered safe to assume that $FPR < 0.01$ for the choice $T_{\mathrm{w}} = 200ms$ in the case of the observed computer, because in our tests none of the 600 measured response times exceeded 163ms.

3.4 Experimental Evaluation of the Detection Algorithm

We tested the algorithm in a real situation. The detector of Section 3.1 was implemented with time windows $T_{\mathrm{ug}} = T_{\mathrm{ud}} = T_{\mathrm{dg}} = 200ms$, as explained in Section 3.3, and $T_{\mathrm{dd}} = 10s$. In addition to logging, the traffic to Twitter.com, outside the specified time window was stopped by the real time insertion of firewall rules. During 8 hours a client computer was used for text processing, email reading/writing, drawing, browsing, and of course using Twitter. At the same time the computer was infected by a bot. The used bot was a recent variant of the Twitternet bot as discovered by Nazario [13]. This variant is also known by Kasperski lab as Backdoor.Win32.Twitbot.c. It polls exactly every 20 seconds a Twitter account and uses SSLv1. Of course in this case we could directly detect this flow by the rarely used SSLv1-protocol and the high query-rate. However, to test the performance of the our detection mechanism, we only used the algorithm of Section 3.2. Table 3 presents the results of the experiment. The results show a very succesful detection of botnet traffic and support the theoretically derived performance of the detection algorithm.

Table 3. Experimental results of the detection algorithm during an 8 hours observation of an infected computer, with $T_{ug} = T_{ud} = T_{dg} = 200$ms and $T_{dd} = 10$s

Parameter	Value
total left mouse click rel. events	1620
total Enter key press events	414
total F5 key press events	28
total flows	7170 (1467 bot + 5703 user)
Detection Rate	0.987 (1448 of 1467 illegal flows)
False Positive Rate excluding reloads	1 (0 of 43 legitimate Twitter flows)
False Positive Rate including reloads	0.6 (17 of 43 legitimate Twitter flows)

The 17 false positives were all caused by automatic reloads of already open Twitter pages. In practice this was not a problem, because the user could still update the pages by a manual reload. The issue of automatic reloads is further discussed in the next section.

4 Special Cases of Twitter Traffic

In the previous sections we have assumed that legal traffic to Twitter.com could only be triggered by human activity and that botnet traffic was never triggered by human activity. Unfortunately, in reality the situation is more complex. We elaborate in this section on two important exceptions:

- automatic legal traffic by applications that poll Twitter periodically;
- user synchronized botnet traffic to evade detection.

4.1 Automatic Legal Traffic

Automatic notifications, which were blocked in the experiment, are an example of legal automatic traffic to Twitter.com. In the case of Twitter we identified three important groups of legal automatic traffic:

- If a timeline or a search page of Twitter.com is loaded, the browser polls approximately every 30 seconds Twitter.com for new messages. New messages result default in a notification, but the page with the new content is not loaded.
- A timeline or hash tag can be polled as RSS-feed. An agent loads a certain Twitter page in RSS-format periodically. Update frequencies are low (typically once a day for IE and once an hour for Firefox).
- A timeline or hash tag can also be polled by special agents like Tweetdeck. These applications periodically load timelines or search results of Twitter.com in .json, .xml, or .rss format.

In the absence of direct user activity, our detector will classify these traffic types as botnet-originated. We explore here some solutions to deal with this false positive problem and restrict ourselves to solutions that can be implemented on the network in the neighborhood of the client.

The first and most rigorous solution is to let the detector prevent this kind of traffic, as we did in the experiment. In certain restricted situations, like a corporate network, this can be an acceptable compromise between security and functionality, because employers can still use Twitter, but only manually.

A second solution is content inspection of automatic Twitter traffic. Unexpected formats or short subscription periods are examples of observations that can classify automatic traffic as malicious. The detector of Section 3.1 is still an essential first step for classifying potential malicious traffic. In the case of SSL-traffic, proxy constructions, or key exchange between the client and the inspecting device, are necessary.

A third solution is the use of white lists that allows for automatic traffic to certain locations of Twitter.com. This can only work if the majority of this list is automatically maintained. For example the detector can allow Twitter locations that have successfully been visited by earlier user events. Also CAPTCHA's can be used, as a condition to manually add locations in the whitelist. For a more thorough elaboration further research is necessary.

4.2 Evasion by User Synchronized Botnet Traffic

The presented detector can only detect botnet traffic if the traffic is not synchronized with user activity. However it is possible to design a bot that waits with its C&C traffic until a suitable user event takes place. In that case the detector will wrongly classify the traffic as human originated with a false negative as a result.

To cope with this evasion technique we suggest some enhancements of the original presented detector. Instead of only looking at separate user events, the enhanced detector uses multiple user and network events, to determine anomalous triggers of Twitter traffic.

For example if a user triggers Twitter traffic by the F5-key, it is the reload of a recently loaded page. If a bot triggers communication on this event, it will result in new traffic, which is a detectable anomaly.

Another example is user-triggered Twitter traffic by the Enter-key. This happens in most legitimate cases immediately after typing the URL, during a login, a search, or after typing a new tweet. Also immediately after the Enter-event, the user will normally wait a few seconds with further events, until complete response. If a bot chooses a random Enter-event to start communication, there is a high probability that the mentioned conditions are not met, resulting in detectable anomalous traffic. Of course a sophisticated bot can wait until all conditions are met, but in that situation there is a high probability of two simultaneous flows to Twitter.com, the bot flow and a user flow, which is again a detectable anomaly.

The left mouse click is an attractive event for a bot to synchronize, because it is the most important trigger of legitimate Twitter traffic. Again observation of earlier and later events can lead to detection of an anomalous situation. For example in case of a double click, the bot starts its communication before it can observe the possible second mouse click, because if it waits to long, the traffic will be outside of the causal window. The result is a detectable anomaly. Also the presence of other user activity immediately after the mouse click can indicate anomalous traffic. Further research is necessary to work these suggestions out into clear algorithms and determine the resulting performance and other consequences, like the potential loss of real time detection.

5 Conclusions and Future Work

We have presented a detection algorithm that successfully detects botnet C&C-traffic to Twitter.com, by observing only causal relationships between observed network traffic and observed user events that potentially can trigger traffic to Twitter.com. The three observed user events that can trigger a visit to Twitter.com are: the Enter key press, the Left Mouse Button release, and the F5 key press. Theory, combined with empirical data, predicts an excellent detection rate and false positive rate. The proposed detection algorithm and its predicted performance are supported by experimental data with real user traffic and botnet traffic. The real time detection permits complete blocking of a C&C-traffic flow. There are feasible solutions that can cope with false positives of automatic legal Twitter traffic and false negatives of bots that synchronize communication on certain user events in order to imitate user intended traffic. Further research is needed to work the proposed solutions out in more detail and to extend the proposed detection mechanism to a more generic technique that can successfully detect botnet C&C-traffic to various social media.

References

1. Ahn, L.V., Blum, M., Hopper, N., Langford, J.: Captcha: Using Hard Ai Problems for Security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 294–311. Springer, Heidelberg (2003)
2. Cooke, E., Jahanian, F., McPherson, D.: The zombie roundup: Understanding, detecting, and disrupting botnets. In: Proc. of the USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet SRUTI 2005. USENIX Association, Cambridge (2005)
3. Davis, C.R., Fernandez, J.M., Neville, S., McHugh, J.: Sybil attacks as a mitigation strategy against the storm botnet. In: Proc. of the 3rd International Conference on Malicious and Unwanted Software MALWARE 2008. IEEE, Alexandria (2008)
4. Giroire, F., Chandrashekar, J., Taft, N., Schooler, E., Papagiannaki, D.: Exploiting Temporal Persistence to Detect Covert Botnet Channels. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 326–345. Springer, Heidelberg (2009)
5. Goebel, J., Holz, T.: Rishi: Identify bot contaminated hosts by irc nickname evaluation. In: Proc. of the first USENIX Workshop on Hot Topics in Understanding Botnets HOTBOTS 2007. USENIX Association (2007)

6. Gorman, G.O.: Google groups trojan (2009)
http://www.symantec.com/connect/blogs/google-groups-trojan
(visited January 2011)
7. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proc. of the 17th USENIX Security Symposium SECURITY 2008. USENIX Association, Berkeley (2008)
8. Gummadi, R., Balakrishnan, H., Maniatis, P., Ratnasamy, S.: Not-a-bot: Improving service availability in the face of botnet attacks. In: Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation NSDI 2009. USENIX Association, Berkeley (2009)
9. Holz, T., Gorecki, C., Rieck, K., Freiling, C.: Measuring and detecting fast-flux service networks. In: Proc. of Symposium on Network and Distributed System Security NDSS 2008. The Internet Society (2008)
10. Kartaltepe, E.J., Morales, J.A., Xu, S., Sandhu, R.: Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 511–528. Springer, Heidelberg (2010)
11. Lelli, A.: Trojan.whitewell: What's your (bot) facebook status today? (2009),
http://www.symantec.com/connect/blogs/
trojanwhitewell-what-s-your-bot-facebook-status-today
(visited December 2010)
12. Mol, J.J.D., Pouwelse, J.A., Epema, D.H.J., Sips, H.J.: Free-riding, fairness, and firewalls in p2p file-sharing. In: Proc. of the Eighth International Conference on Peer-to-Peer Computing P2P 2008. IEEE (2008)
13. Nazario, J.: Twitter-based botnet command channel (August 2009),
http://asert.arbornetworks.com/2009/08/
twitter-based-botnet-command-channel/ (visited October 2010)
14. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: Proc. of the 3rd International Conference on Malicious and Unwanted Software MALWARE 2008. IEEE, Alexandria (2008)
15. Porras, P., Saidi, H., Yegneswaran, V.: A foray into conficker's logic and rendezvous points. In: Proc. of the Second USENIX Workshop on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, and More LEET 2008. USENIX Association, Boston (2009)
16. Provos, N.: A virtual honeypot framework. In: Proc. of the 13th Conference on the USENIX Security Symposium SSYM 2004. USENIX Association, San Diego (2004)
17. Schiller, C., Binkley, J.: Botnets: The Killer Web Applications, 1st edn. Syngress Publishing, Rockland MA (2007)
18. Stinson, E., Mitchell, J.: Towards systematic evaluation of the evadability of bot/botnet detection methods. In: Proc. of the 2nd Conference on USENIX Workshop on Offensive Technologies WOOT 2008. USENIX Association, Berkeley (2008)
19. Taylor, K.: An Analysis of Computer Use across 95 Organisations in Europe, North America and Australasia. Tech. rep., Wellnomics (2007)
20. Vo, N.H., Pieprzyk, J.: Protecting web 2.0 services from botnet exploitations. In: Proc. of the 2nd Workshop on Cybercrime and Trustworthy Computing CTC 2010. IEEE, Washington, DC (2010)

Finding Non-trivial Malware Naming Inconsistencies

Federico Maggi, Andrea Bellini, Guido Salvaneschi, and Stefano Zanero

Dipartimento di Elettronica e Informazione, Politecnico di Milano

Abstract. Malware analysts, and in particular antivirus vendors, never agreed on a single naming convention for malware specimens. This leads to confusion and difficulty—more for researchers than for practitioners—for example, when comparing coverage of different antivirus engines, when integrating and systematizing known threats, or comparing the classifications given by different detectors. Clearly, solving naming inconsistencies is a very difficult task, as it requires that vendors agree on a unified naming convention. More importantly, solving inconsistencies is impossible without knowing exactly where they are. Therefore, in this paper we take a step back and concentrate on the problem of finding inconsistencies. To this end, we first represent each vendor's naming convention with a graph-based model. Second, we give a precise definition of inconsistency with respect to these models. Third, we define two quantitative measures to calculate the overall degree of inconsistency between vendors. In addition, we propose a fast algorithm that finds non-trivial (i.e., beyond syntactic differences) inconsistencies. Our experiments on four major antivirus vendors and 98,798 real-world malware samples confirm anecdotal observations that different vendors name viruses differently. More importantly, we were able to find inconsistencies that cannot be inferred at all by looking solely at the syntax.

1 Introduction

The current threat landscape is characterized by money-driven campaigns [1] mainly spread through drive-by download [2], more than by self-replicating code. Classic polymorphic viral engines gave way to multiple layers of packing, obfuscation, recompilation, and advanced self-update mechanisms. As a consequence, a rising number of unique malware specimens, often mutated versions of known malware, spurred a transformation in the mechanisms of action of antiviruses, which rely more and more on generic signatures and heuristics [3].

Because of historical reasons and vendor-specific policies, malware naming has never followed any conventions [3] (e.g., vendors and researchers used to name viruses based on characteristic they found interesting). Unfortunately, naming inconsistencies are a real problem when trying to correlate useful data across different antiviruses. Even simple problems such as comparing top-ten threat lists are in turn very difficult[1]. Consequently, researchers have concentrated on solving inconsistencies and proposed

[1] http://infosecurity-us.com/view/6314/malware-threat-reports-fail-to-add-up

both pragmatic approaches (e.g., VGrep, Wild List[2])and new naming conventions (e.g., CARO[3]).

However, *finding* and *understanding* naming inconsistencies is a necessary (and missing) step before *solving* them. To address this, we extend the notion of "consistency" presented in [4], and propose a systematic approach for quantifying and finding inconsistencies. By observing these inconsistencies, and armed with the knowledge of a vendor's detection methodology, an expert can investigate the inconsistencies.

We experimentally identify a number of strong inconsistencies, confirming that the problem is deep and structural. Also, we show that inconsistencies are not uniformly spread across different antiviruses (i.e., some vendors are more consistent, while others are wildly different). Last, we find large groups of inconsistently-labeled samples which cannot be made consistent in any sensible way.

In summary, we make the following contributions:

- We define a systematic technique to create simple yet effective graph-based models of vendors' naming conventions (§3.2) by means of which we formally define the concept of *consistency*, *weak inconsistency* and *strong inconsistency* (§3.3).
- We propose two quantitative measures that evaluate the overall degree of inconsistency between two vendors' naming conventions (§3.3) and, more importantly, we define a simple algorithm that finds the inconsistent portions of graph model.
- We describe the results obtained by applying the proposed techniques on a real-world dataset comprising 98,798 unique malware samples, scanned with four real antivirus products, visualize and analyze consistencies, strong and weak inconsistencies (§4), and briefly explain how these can be solved (§3.3).

2 Malware Naming Inconsistencies

Although variants of viruses and worms were relatively common, they tended to form just a small tree of descendants. Therefore, even with different conventions (e.g., calling a child "virus.A" as opposed to "virus.1"), such trees were easy to match across different vendors (e.g., with VGrep[4]). Even polymorphic viruses did not pose a serious challenge in this scheme. An effort to standardize names was CARO, which proposed the following naming convention: `<type>://<platform>/<family name>.<group name>.<length>.<sub variant><devolution><modifiers>`. However, this effort was unsuccessful. Even if it had been, a standard syntax would solve just a subset of the problem, without reconciling different family or group names between vendors.

The CME initiative[5] tried to deal with the problem by associating a set of different specimens to a single threat, but the approach proved to be unfeasible. At the same

[2] `http://www.sunbelt-software.com/ihs/alex/vb_2007_wildlist_paper.pdf`

[3] `http://www.people.frisk-software.com/ bontchev/papers/naming.html`

[4] `http://www.virusbtn.com/resources/vgrep/index.xml`

[5] `http://cme.mitre.org/cme/`

time, most malware authors began to use malware kits, and to borrow or steal code from each other. As a result, many samples may descend from a mixture of ancestors, creating complex phylogenies that are not trees anymore, but rather lattices. This, in turn, motivated the evolution of antivirus engines, which now rely on generic signatures including behavior-based techniques inspired by anomaly detection approaches. Consequently, correlating the outcomes of different antiviruses is even more complex [5]. For instance, in [4] signature-based antiviruses are compared with behavioral classifiers by means of consistency (i.e., similar samples must have similar labels), completeness and conciseness of the resulting detection. This work has highlighted the presence of a non-negligible number of inconsistencies (i.e., different labels assigned to similar samples).

However, as of today, no complete and reliable method exists to find inconsistencies. Therefore, before consolidating malware names, we first need to quantify and spot them precisely.

3 Finding Naming Inconsistencies

We hereby describe a two-phase, practical approach to build a high-level picture of inconsistencies in malware naming conventions across a given set of antivirus products or vendors (henceforth named "vendors" for simplicity). Our goal is to spot inconsistencies that go beyond well-known syntactic differences. Given a list of unique samples, our method produces a *qualitative* comparison, finds subsets of samples labeled inconsistently, and produces two *quantitative* indicators of the degree of inconsistency.

Phase 1 (modeling). For each vendor, we model malware names according to structural similarity between their *labels* (§3.2).

Phase 2 (analysis). We compare the aforesaid models quantitatively by means of a set of structural and numerical indicators (§3.3).

For instance, when patterns such as "`-backdoor`", "`.backdoor.`", "`-backdoor-dialer`", or "`backdoor.dialer`" are found, we assume that, according to the vendor under examination, the samples are all characterized by being "backdoors", and thus **Phase 1** organize them in the same group. In other words, we model each vendor by means of the groupings induced by its naming convention. In **Phase 2**, two vendors are considered consistent if they both group samples together in the same manner, regardless of the actual labeling. For instance, a group comprising sample m_1 (labeled as "`foo-backdoor`") and sample m_2 (labeled as "`bar-backdoor`") is consistent with a group comprising the same exact samples labeled as "`blah-trojan`" and "`foobar-trojan`", respectively.

3.1 Types of Inconsistency

We define two different types of inconsistencies:

Weak Inconsistency: One vendor divides the set of samples into more groups, whereas the other vendor groups them all together, thus creating a "one-to-many" mapping as opposed to one or more "one-to-one" mappings. This inconsistency is weak as it descents from the different granularities adopted by vendors).

Strong Inconsistency: Vendors spread samples in multiple groups, such that there is no mapping between the groups .

In §3.3 we further formalize these definitions by means of models constructed in **Phase 1**, and define a fast algorithm to identify them.

3.2 Phase 1: Naming Convention Modeling

We use a simple, top-down hierarchical approach (§3.2) to transform the initial set of malware samples into nested sub-sets, such that each (sub-)set contains only samples labeled with similar *string patterns*. Patterns are extracted offline for each vendor (§3.2).

Pattern Extraction. Our technique is centered around four *pattern classes*, marked with angular brackets (i.e., `<class>`):

`<type>`	distinctive activity of the malicious code (e.g., "`backdoor`", "`worm`", or "`dialer`", "`packed`", "`tool`").
`<family>`	name of a specific malware family (e.g., "`Conficker`", "`Mudrop`", "`Fokin`", "`Allaple`").
`<platform>`	operating system (e.g., "`W32`", "`WNT`") or language interpreter (e.g., "`JS`", "`PHP`").
`<version>`	malicious code version (e.g., "`B`" and "`D`" in labels "`PHP:IRCBot-B`" and "`PHP:IRCBot-D`"), or information to disambiguate "releases" or signatures (e.g., "`gen`", "`gen44`", "`damaged`").

This small, generic set of pattern classes allows to analyze several vendors. New classes can be added and extend our approach to virtually any vendor. Our analysis on real samples revealed that each class can contain either one simple pattern or a hierarchy:

Simple Pattern: occurrence of a string of a given class, e.g., `<type>` = `Trojan`. Usually, `<platform>` and `<family>` occur as simple patterns (e.g., `<platform>` = `Win32` | `PHP`).

Hierarchy: occurrence of more simple patterns of the same class, e.g., `<type1>` = "`Trojan`" and `<type2>` = "`Dropper`" are both of class `<type>`. For example, when vendors specify both a threat type and sub-type, this leads to hierarchies of simple patterns, denoted as concatenated simple patterns in order of precedence, e.g., `<type>` = `<type1>/<type2>/<type3>`, `<version>` = `<version1>/<version2>`. We use the slash separator just to describe our results, though it by no means reflects any syntax.

Simple patterns can be constructed either manually, from a handful of labels, or by leveraging automatic inference tools to derive the most probable syntax of a given set of strings for subsequent manual revision. However, as manual revision would be required anyway to ensure accurate results, we opt for a heuristic approach (detailed in §3.2), that allows us to extract the patterns in a semi-automatic fashion. Hierarchies of patters of the same class are ordered with respect to their relative frequency of appearance. For instance, given one vendor and simple patterns `<typeX>` and `<typeY>`, $X < Y$ if `<typeX>` occurs more than `<typeY>` on a given set of malware sample. If they have the same frequency, the hierarchy is replaced by a simple pattern `<type>`, which contains the common substring between `<typeX>` and `<typeY>`.

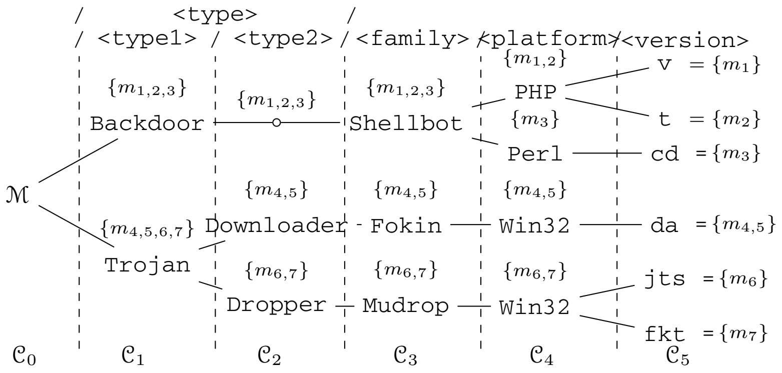


Fig. 1. Example output of **Phase 1** on a set $\mathcal{M}$ with seven samples. For instance, $\mathcal{C}_1 = \{\{m_1, m_2, m_3\}, \{m_4, m_5, m_6, m_7,\}\}$ and $\mathcal{C}_{2-4} = \{\{m_3\}, \{m_6, m_7\}, \{m_4, m_5\}, \{m_1, m_2, m_3\}, \{m_1, m_2\}\}$. Note that, `<type>` comprises `/<type1>/<type2>/`.

Tree-Based Models. Given a set $\mathcal{M}$ of labeled samples, we run the following procedure for each vendor on a given set of patterns. We consider one class of patterns at time.

We first split the initial set of labels according to `<type>`. For example, given `<type>` = "`Backdoor|Trojan`", the samples labeled as `Backdoor.Perl.Shellbot.cd`, `Backdoor.PHP.Shellbot.v` and `Backdoor.PHP.Shellbot.t` fall in the same sub-set, whereas `Trojan-Downloader.Win32.Fokin.da`, `Trojan-Dropper.Win32.Mudrop.fkt` and `Trojan-Dropper.Win32.Mudrop.jts` fall in a different one. If the vendor under consideration adopts hierarchical patterns, this step is repeated for each sub-pattern. Continuing the above example, the trojan samples are separated in two different sub-sets.

When a (sub-)set can be split no further according to the same pattern class, we consider the `<family>`. In our example, the only possible split is by means of "`Fokin`" and "`Mudrop`", as "`Shellbot`" induces no splits. Then the first set is split in two sub-sets, one containing only `Backdoor.Perl.Shellbot.cd` and one with `Backdoor.PHP.Shellbot.t` `Backdoor.PHP.Shellbot.v`. Further splits are performed according to the different `<version>` patterns (if any). More precisely, "`.v`" and "`.t`" forms two sub-sets as well as "`.fkt`", and "`.jts`" do.

We stop when the latest pattern class has been considered. In our example, the procedure ends after one split induced by the `<version>`. At each split, we store the links between the sets thus constructing a tree, rooted in the initial set, as exemplified in Fig. 1.

Definition 1 (Naming Tree). *Given a set $\mathcal{M}$ of malware names, we define* naming tree *the output of* ***Phase 1****, which is $\mathcal{C}_d(\mathcal{M}) \subset \wp(\mathcal{M})$, where d is either: (1) a number that indicates the depth in the tree, e.g., $\mathcal{C}_1$, (2) an interval between depths in the tree, e.g., $\mathcal{C}_{1-2}$, or (3) a mnemonic expression ($\mathcal{M}$ is omitted when implicit from the context).*

In Fig. 1, $\mathcal{C}_1 = \{\{m_1, m_2, m_3\}, \{m_4, m_5, m_6, m_7,\}\}$ and $\mathcal{C}_{2-4} = \{\{m_3\}, \{m_6, m_7\}, \{m_4, m_5\}, \{m_1, m_2, m_3\}, \{m_1, m_2\}\}$. The whole tree is $\mathcal{C} = \mathcal{C}_0 = \mathcal{C}_0(\mathcal{M}) = \{\mathcal{M}\}$, or $\mathcal{C}^v$, where v is the vendor under examination. We indicate portions of the naming tree with mnemonic expressions; for instance, "/*/<family>/*" denotes the portion of naming tree corresponding to the <family>, that are $\mathcal{C}_3 = \mathcal{C}_{(/*/\texttt{<family>}/*)} = \{\{m_1, m_2, m_3\}, \{m_4, m_5\}, \{m_6, m_7\}\}$. Actual substrings can be used as well: $\mathcal{C}_{/\texttt{Backdoor}/*} = \{\{m_1, m_2, m_3\}\}$. A hierarchy of patterns always refer to the lowest depth. For instance, $\mathcal{C}_2 = \mathcal{C}_{(/*/\texttt{<type2>}/*)} = \{\{m_1, m_2, m_3\}, \{m_4, m_5\}, \{m_6, m_7\}\}$.

Implementation Details

Pattern Extraction. The extraction procedure is run for each vendor and takes (1) a set of malware labels $\mathcal{L}$ and (2) an a small set of *separators*, [/:.-_!] (this can be customized easily by analyzing the frequency of symbols in the labels corpus). The algorithm iteratively breaks labels into substrings. At each iteration an operator reviews a set of candidate substrings and assign them to an appropriate pattern class. Pattern classes are initially empty, e.g., <type> = ''. At the i-th iteration a random, small (e.g., 10) subset of labels $\mathcal{L}_i \subseteq \mathcal{L}$ is selected and labels are broken into substrings according to separators. Then, the operator assigns each unique substring to the appropriate class. For example, if Win32, Allaple, Trojan, and PHP are found, the appropriate class is updated, i.e., <platform>$_i$ = Win32 | PHP, <type>$_i$ = Trojan, <family>$_i$ = Allaple. All substrings extracted from each label in $\mathcal{L}_i$ must be assigned to exactly one class. Labels with at least one substring not assigned to any class are postponed for subsequent analysis (and removed from $\mathcal{L}_i$). Alternatively, the operator can add new separators as needed to handle the current subset of labels. When labels in $\mathcal{L}_i$ are covered, $\mathcal{L}$ is reduced by removing all the labels that can be parsed with the existing patterns. Then, the next random sample $\mathcal{L}_{i+1} \subseteq \mathcal{L} \backslash \mathcal{L}_i$ is drawn ($\mathcal{L}_{i+1}$ may include postponed labels). The runs until $\mathcal{L} = \varnothing$.

The larger each random sample size is, the faster and more accurate this procedure becomes, also depending on the operator's experience. However, this procedure needs to be ran only once per vendor and, more importantly, the time and effort required decrease from vendor to vendor, as patterns can be reused (e.g., family and platforms recur across vendors with minimal variations). In real-world examples, a minority of labels may deviate from the patterns (e.g. when labels are handwritten by malware analysts).

Singletons. Consider patterns <version> = v|t and a set $\{m_1, m_2\}$, where m_1 = Backdoor.PHP.Shellbot.v, m_2 = Backdoor.PHP.Shellbot.t. A split would produce two sub-sets $\{m_1\}$, $\{m_2\}$.

To one end, one outlier is not representative of the pattern, e.g., "t" or "v". To the other hand, since our goal is to analyze consistency, we expect that, if two vendors are consistent, they would produce similar sets, also including "outliers". For this reason, to take into account both the observations, sets of size below a certain threshold, T_o, are labeled with a special pattern, <misc> that encode such "uncertainty".

For example, /Backdoor/Shellbot/PHP/ identifies the set $\{m_1, m_2\}$, whereas the label /Backdoor/Shellbot/<misc>/ identifies $\{m_3\}$. Note that, more miscellaneous sets may exist at the same depth.

Named Depth. For different vendors, a named depth (e.g., "<family>"), may correspond to different numerical depths. Therefore, while constructing the naming trees we construct the mapping between numerical and named depths.

3.3 Phase 2: Comparing Vendors

We compare two vendors A, B by means of their naming trees $\mathcal{C}^A, \mathcal{C}^B$ (Def. 1). We first calculate two indicators (§3.3) that quantify the degree of inconsistency between naming conventions between vendors; and then we spot inconsistencies (§3.3).

Naming trees are hierarchies of sets. However, we compare sets derived by "cutting" naming trees at a given depth d, omitted for simpler notation: $\mathcal{C}^A = \mathcal{C}^A_d$ and $\mathcal{C}^B = \mathcal{C}^B_d$.

Quantitative Comparison. We define the *naming convention* distance, which expresses the overall difference between the naming conventions of A and B, and the *scatter measure*, which expresses the average number of sets of one vendor that are necessary to cover each set of the other vendor (and vice versa).

Definition 2 (Naming convention distance). *The* naming convention distance *between vendors A and B is defined as the average distance between their sets.*

$$D(\mathcal{C}^A, \mathcal{C}^B) := \frac{1}{2}\left(\frac{\sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B)}{|\mathcal{C}^A|} + \frac{\sum_{c \in \mathcal{C}^B} \delta(c, \mathcal{C}^A)}{|\mathcal{C}^B|}\right) \tag{1}$$

$\delta(c, \mathcal{C}') = \min_{c' \in C'} d(c, c')$ *being the minimum diff. between $c \in \mathcal{C}$ and any set of $\mathcal{C}'$.*

The denominator is such that $D(\cdot, \cdot) \in [0, 1]$, and $d(c, c') = 1 - J(c, c') \in [0, 1]$, where $J(c, c')$ is the Jaccard index. A similar distance was used to measure the similarity between sets of overlapping trees of sets [6].

Definition 3 (Scatter Measure). *The* scatter measure *between vendors A and B is defined as the average number of sets in each vendor's model that are necessary to cover one set drawn from the other vendor's model (and vice-versa). More formally:*

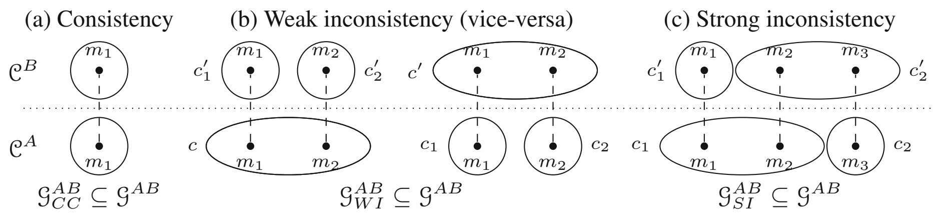
$$S(\mathcal{C}^A, \mathcal{C}^B) := \frac{1}{2}\left(\frac{\sum_{c \in \mathcal{C}^A} |\Gamma(c, \mathcal{C}^B)|}{|\mathcal{C}^A|} + \frac{\sum_{c \in \mathcal{C}^B} |\Gamma(c, \mathcal{C}^A)|}{|\mathcal{C}^B|}\right) \tag{2}$$

where $\Gamma(c, \mathcal{C}')$ is the scatter set.

Definition 4 (Scatter set). *The* scatter set *of c by $\mathcal{C}'$ is $\Gamma(c, \mathcal{C}') := \{c' \in \mathcal{C}' \mid c \cap c' \neq \varnothing\}$.*

In other words, Γ contains sets of $\mathcal{C}'$ (e.g., model of vendor B) that have at least one element (e.g., malware sample) in common with a given $c \in \mathcal{C}$ (e.g., model of vendor A). As $\mathcal{C}^A$ and $\mathcal{C}^B$ are *partitioned*, $|\Gamma(c, \mathcal{C}')|$ is the number of sets of $\mathcal{C}'$ that build c.

Structural Comparison. We recognize the inconsistencies defined in §3.1. To this end, trees at a given depth are first represented as undirected graphs with cross-vendor edges, and then searched for inconsistent sub-graphs. This comparison applies only when $\mathcal{C}^A$ and $\mathcal{C}^B$ are partitioned into flat sets. Therefore, only for this analysis, we assume that sets are drawn from trees at leaf depth (i.e., `<version_n>`), representative of the whole label.



(a) Consistency
(b) Weak inconsistency (vice-versa)
(c) Strong inconsistency
$\mathcal{C}^B$
$\mathcal{C}^A$
m_1
m_2
m_3
c
c'
c_1
c_2
c'_1
c'_2
$\mathcal{G}^{AB}_{CC} \subseteq \mathcal{G}^{AB}$
$\mathcal{G}^{AB}_{WI} \subseteq \mathcal{G}^{AB}$
$\mathcal{G}^{AB}_{SI} \subseteq \mathcal{G}^{AB}$

Fig. 2. Instances of consistencies $\mathcal{G}^{AB}_{CC}$, weak inconsistencies $\mathcal{G}^{AB}_{WI}$ and strong inconsistencies $\mathcal{G}^{AB}_{SI}$, i.e., connected components of the linked naming tree $\mathcal{G}^{AB}$ of vendors A vs. B. Each vertical line represents a malware sample.

Definition 5 (Linked Naming Trees). *Given* $\mathcal{C}^A$ *and* $\mathcal{C}^B$ *the* linked naming tree *is an undirected graph* $\mathcal{G}^{AB} := \langle \mathcal{V}^{AB}, \mathcal{E}^{AB} \rangle$, *where* $\mathcal{V} = \mathcal{C}^A \cup \mathcal{C}^B$ *and* $\mathcal{E}^{AB} = \{(c, c') \mid c \in \mathcal{C}^A, c' \in \mathcal{C}^B \wedge c \cap c' \neq \varnothing\}$.

In other words, $\mathcal{G}^{AB}$ encodes the links between sets of labeled samples. Given a set c of samples labeled by A, and a set c' of samples labeled by B, we set an edge from c to c' only if c' has at least one sample in common with c. In §3.3 we extend this concept with edges weighted proportionally to the number of samples shared between c and c'. Therefore, we reduce the problem of recognizing inconsistencies to finding connected components of $\mathcal{G}^{AB}$, for which efficient algorithms (e.g., [7]) exist. The connected components are then analyzed automatically to distinguish among:

Consistency (CC). (Fig. 2a) The connected component has two sets with the same samples (samples of A may have different labels than samples of B).

Weak Inconsistency (WI). (Fig. 2b) The connected component contains only one set $c \in \mathcal{V}^A = \mathcal{C}^A$, and all sets $c' \in \mathcal{V}^B = \mathcal{C}^B$ are its subsets $c' \subset c$. In this case, vendor B adopts more fine-grained naming convention than vendor A. Despite $\mathcal{C}^A$ and $\mathcal{C}^B$ are not identical, vendors disagree only on the amount of information in each label.

Strong Inconsistency (SI). The connected component contains more than one set for each vendor (e.g., for sets c'_1, c_1, c'_2, c_2 in Fig. 2c). As sets are partitions of the entire set of malware samples, there must be at least four sets $c_1, c_2 \in \mathcal{V}^A = \mathcal{C}^A$, $c'_1, c'_2 \in \mathcal{V}^B = \mathcal{C}^B$ such that the following condition holds: $c_1 \bigcap c'_1 \neq \varnothing \quad \wedge \quad c_2 \bigcap c'_2 \neq \varnothing \quad \wedge \quad c'_2 \cap c_1 \neq \varnothing$. In other words, the sets share some samples without being all subsets of each other. The inconsistency, which includes all sets of the connected component, is caused by inherently different naming conventions. Once found, these inconsistencies can be solved by fusing, say, c_1 with c_2.

Implementation Details

Scatter Set Coverage. Our implementation incorporates a measure of *coverage*, σ, in scatter sets $\Gamma(c, \mathcal{C}')$ (Def. 4), defined as $\sigma(\Gamma) := |c \cap \bigcup c'|/|c|\%$, where the union is calculated for any $c' \in \Gamma(c, \mathcal{C}')$. The coverage quantifies the percentage of samples in c (e.g., a set of vendor A) shared with the union of scatter sets derived from $\mathcal{C}'$ (e.g., a set tree of vendor B). Scatter sets can be selected by their σ, and thus, given a threshold $T_\sigma \in [0, 100]$, the *minimum scatter set* of c with respect to $\mathcal{C}'$ can be selected as $\widehat{\Gamma}_{T_\sigma} : \nexists \Gamma(c, \mathcal{C}')$ for $\sigma(\Gamma) \geq T_\sigma \wedge |\Gamma| < |\widehat{\Gamma}|$: The smallest scatter set that covers c of at least T_σ.

Weighted Linked Naming Trees. The edges of the linked naming trees (Def. 5) are weighted with the following weighting function:

$$W(c, c') := \max\left\{\frac{|c \cap c'|}{|c|}\%, \frac{|c \cap c'|}{|c'|}\%\right\}, \forall (c, c') \in \mathcal{E}^{AB} \tag{3}$$

Each edge encodes the degree of "overlapping" between two sets c and c' originated from A and B, respectively. Note that, our normalization ensures that weights quantify the actual fraction of c shared with c', regardless of the size of c', which can be disproportionally larger than c (and vice-versa). Our analysis can be thus parametrized by a threshold $T_W \in [0, 100]$, used to convert weighted graphs into graphs by pruning edges $e = (c, c')$ below T_W, i.e., $W(c, c') < T_W$.

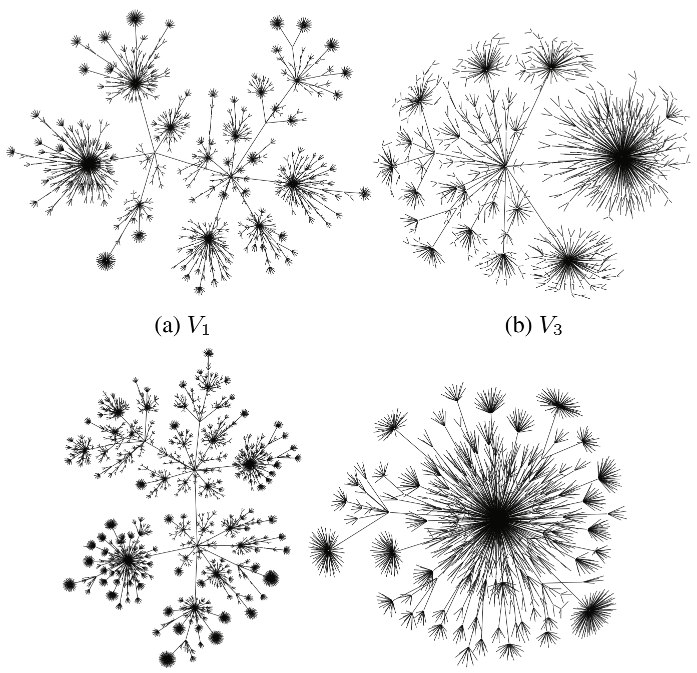
4 Experimental Measurements

Microsoft	V_1	`<type>:<plat>/<family>[.gen[!<ver1>]|<ver2>]`	4,654 labels
Antiy	V_2	`<type>.<plat>/<family>[.<ver1>.<ver2>]`	23,603
Kaspersky	V_3	`<type>/<plat>.<family>[.gen]`	2,122
Avast	V_4	`<plat>/<family>[-gen|-<ver>]`	4,350

These naming conventions cover the vast majority of the samples. These vendors are good candidates because the "richness" of their naming convention allows a granular analysis, which spans from `<type>` to `<version>`. Adding more vendors is computationally feasible, although the number of unique couples drawn from the set of vendors would grow quickly. Therefore, from a presentation perspective, this may yield cluttered and confusing diagrams. Given that our goal in this paper is to evaluate our method and show that it finds structural inconsistencies, as opposed to simply quantifying them, using four vendors, totaling six comparisons, seems sufficient and actually clearer.

Vendor V_4 includes no `<type>`. We manually analyzed this case and discovered that the `<family>`, which is instead present in the syntax, is seldom used also to hold information about the threat type (e.g., "Malware", "Dropper", "Trojan" in Fig. 4). As this happens quite seldom, it is reasonable to consider it as part of the semantic of the naming convention. For this reason, only for vendor V_4, we safely consider threat type and family name at the same level of importance. Note that, other vendors handle this exception by assigning `<family>` = "`generic`".

Dataset. Our dataset $\mathcal{M}$, generated on Sep 13, 2010, comprises 98,798 distinct malware samples identified by their hashes. We derived the labels $\mathcal{L}^{V_1}, \mathcal{L}^{V_2}, \mathcal{L}^{V_3}, \mathcal{L}^{V_4}$ via VirusTotal, an online service which allows to scan samples with multiple vendors simultaneously. We selected a set of 98,798 samples recognized by the majority of the four main vendors. Frequent labels in the datasets include, for instance, "`TrojanSpy:Win32/-Mafod!rts`", "`Net-Worm.Win32.Allaple.b`", "`Trojan/Win32.Agent.gen`", "`Trojan:Win32/Meredrop`", "`Virus.Win32.Induc.a`". A minority of labels deviates from these conventions. For example, in V_4 only eight labels (0.00809% of the dataset) contain "`gen44`" instead of "`gen`". Similarly, five labels (0.00506% of the dataset) of V_2 contain a third version string. Other cases like the "`@mm`" suffix in V_1 labels (101 labels, about 0.10223% of the dataset) fall outside the above convention.


(a) V_1 (b) V_3

(c) V_2 (d) V_4

Fig. 3. Visual comparison of naming trees of each vendor

From a purely syntactic point of view, these cases are similar to the presence of keywords that often mark special samples (e.g., "`packed`"). We handled this handful of outliers manually.

Phase 1 was run on the dataset to create a model, i.e., naming tree, for each vendor, $\mathcal{C}^{V_1}, \mathcal{C}^{V_2}, \mathcal{C}^{V_3}, \mathcal{C}^{V_4}$. Next, quantitative and structural analysis of **Phase 2** have been run. The outcome of this experimentation is presented and discussed in the remainder of this section, after a brief visual overview of the naming trees.

4.1 Naming Tree Visual Comparison

Different naming conventions induce naming trees that are structurally dissimilar, as evident even at a high level (Fig. 3): V_4 splits samples very early in many sub-sets based on their `<family>`, whereas other vendors use finer conventions and form sparser trees.

For ease of visualization, in Fig. 4 we extracted a slice of 50 samples from one set (i.e., classified as `packed-Klone` by V_3, used as a comparison baseline). Vendor V_1 spreads the same samples onto 8 sets, including `worm`, `trojan`, and `pws` subsets. Also, a separate subset holds samples not even considered malicious by V_1. This example, drawn from a real dataset, also shows that the labels' granularity varies across vendors. For instance, V_1, which adopts a granularity of 1 to 3, splits `worms` (depth 1) in `Pushbots` and `Miscellaneous` (depth 2)[6]. Instances of the same behavior occur

[6] In this example, singletons are visualized as such, but actually contain more than one sample; this is because we sliced a set of 50 samples from a full naming tree.

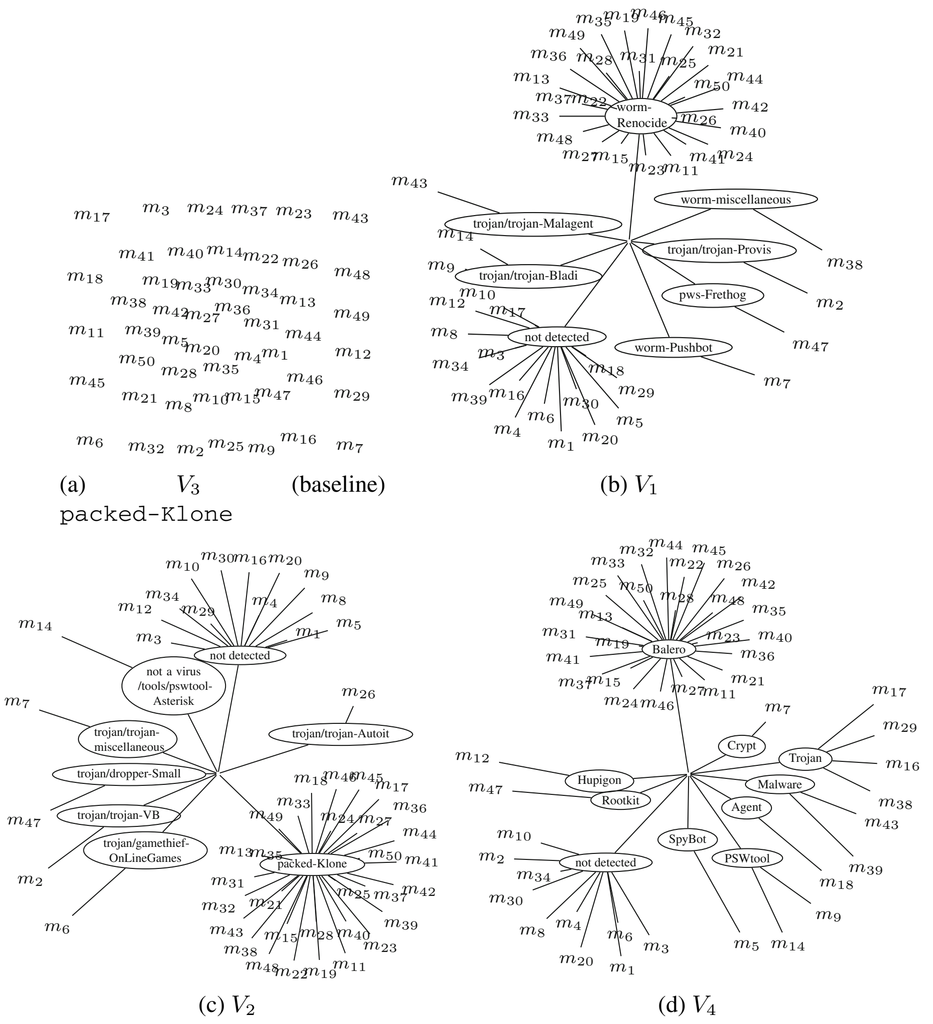


Fig. 4. A flat set extracted from V_3's naming tree cut at `<family>` depth. Other vendors group the same set of samples differently. Only one baseline set is shown, although the same behavior can be observed also with other baselines, i.e., V_1, V_2, V_4.

in V_2, with depths of 1 to 4, and V_4, with depth is 1. We now analyze these "visual" discrepancies thoroughly.

4.2 Singletons and "Not Detected" Samples

Certain values of T_W may yield isolated singletons (i.e., singletons from one vendor with no corresponding sets in the counterpart). Depending on the structure of the linked naming trees, by varying T_W, these nodes may either link to another single node (i.e., consistency), or to several nodes (i.e., inconsistency). Optimistically, we could count them as consistencies because, for a certain, low value of T_W, at least one linked set exists. On the other hand, we could consider such nodes as potential inconsistencies. Due to this inherent ambiguity, we ignore singleton nodes to avoid biased results.

During pattern extraction, we treat samples that are not detected as malicious by one vendor as labels containing only a `<type_n>` string. These are not exactly naming inconsistencies, as they depend on detection accuracy more than on naming structures. Indeed, they can originate from false positives or false negatives. These nodes link to several other sets, and thus spur a minority of very large inconsistencies—possibly up to the whole graph. This may bias the quantitative comparison. Hence, we removed such sets from the following analysis. More precisely, the scatter measure discussed in §4.3 ignores the scatter sets originating from these sets. Similarly, the linked naming trees (Def. 5) used for structural comparison, discussed in §4.3, was pruned by removing "`not detected`" sets (i.e., nodes). Also, for consistency with the choice of excluding singleton nodes, we also removed nodes *only* connected to such nodes.

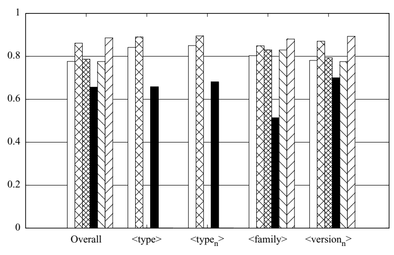
4.3 Quantitative Comparison

Naming Convention Distance. Fig. 5a summarizes the distance for each unique couple of vendors A vs. B, quantifying the overall inconsistency between the vendors' naming conventions. The overall consistency is higher (i.e., distance is lower) at `<version_n>` depth than at `<family>` depth, and is also higher at `<family>` than at `<type>` depth. Interestingly, this contradicts the intuitive conjecture that lower levels in the naming tree would exhibit progressively lower consistency. Also, vendors V_2 and V_3 are remarkably more consistent than the other couples, especially at `<family>` depth. These two vendors exhibit small structural inconsistencies as also noted in §4.3.

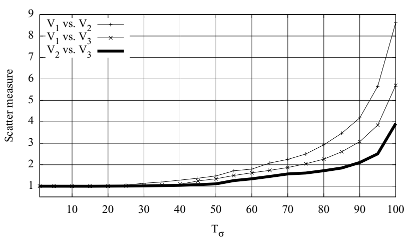
Scatter Measure. The scatter measure how elements of one set of vendor A are distributed (i.e., scattered) onto (multiple) sets of vendor B (and viceversa). We calculate the scatter measure at different values of coverage, $\sigma(\Gamma)$, of the scatter set (i.e., the set of sets in $\mathcal{C}^B$ that corresponds to the set $c \in \mathcal{C}^A$ under examination, and vice versa). We do this from A to B and vice versa, and vary a threshold T_σ. Therefore, we calculate $S(\mathcal{C}^A, \mathcal{C}^B)$ for $T_\sigma \in \{1\%, 5\%, 10\%, \ldots, 95\%, 100\%\}$. Low values of T_σ lead to lower, optimistic, values of $S(\cdot, \cdot)$, reflecting the existence of small scatter sets, which are selected even if they cover only a slight portion of the set under examination. Contrarily, higher values of T_σ unveil the *real* scatter sets, that are those with substantial overlapping.

Fig. 5(b–d) summarizes the results of this experiment for each couple of vendors at different "cuts" of the naming trees: $d \in \{$`<type_n>`, `<family>`, `<version_n>`$\}$. As expected from previous analysis (yet contradicting intuitive presuppositions), the scatter measure decreases at lower depths, except for V_2 vs. V_3, which reveal their overall consistency, especially at `<family>` level—as we concluded from Fig. 5a.

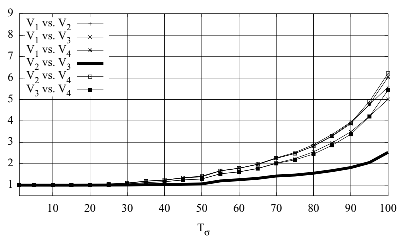
Another interesting comparison is V_1 vs. V_3, which, according to Fig. 5a, show remarkable distance and thus can be considered different from one another. First, Fig. 5(b–d) confirms this conclusion. In addition, these vendors tend to have divergent scatter measures (for increasing values of T_σ), especially at `<type_n>` depth (Fig. 5b), thus revealing that they disagree more on threat types than on versions. Interestingly, this



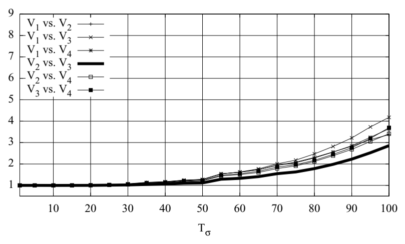
(a) Naming distance



(b) Scatter measure d = <type_n>



(c) Scatter measure d = <family>



(d) Scatter measure d = <version_n>

Fig. 5. Naming convention distance (a) at different depths of the naming trees, and scatter measure (b–d) between each two vendors at different values of T_σ. Relatively high distance between vendors is observed. Notably, the depth (e.g., <type>, <family>) negligibly influences the distances, except for V_2 vs V_3, which exhibit slightly more similarity in terms of <version>. At $T_\sigma = 1.0\%$, the scatter measure is optimistic as almost no coverage is required to find matching sets between vendors; at $T_\sigma = 100\%$ the comparison is realistic because, in order to match sets between vendors, complete coverage is required. On average, almost every vendor have sets that scatter onto 2–5 sets of another vendor. Vendors V_2 vs. V_3 exhibit a steady scatter measure within 1–4, confirming their high degree of consistency according to the naming distance (a).

cannot be inferred by observing their grammars, which look similar at a first glance. Manual examination reveals that V_1 and V_3 agree on the use of the keyword ``.gen'' to indicate the use of "generic" malware signatures. A negligible minority of samples are labeled with an additional progressive number (e.g., ``.gen44'') by V_3, which cannot be safely considered as proper version of the malware.

Structural Comparison. The connected components of the linked naming trees, $\mathcal{G}^{AB}$, constructed by connecting corresponding sets between $\mathcal{C}^A$ and $\mathcal{C}^B$ (as described in §3.3) are good spots for finding consistencies, weak inconsistencies or strong inconsistencies. As shown in Fig. 2, consistencies contain exactly two nodes (i.e., sets), whereas weak and strong inconsistencies comprise several nodes. Weak inconsistencies are 1-to-N relationships, where N indicates the granularity of one vendor with respect to the other, and by no means indicate a "badness" of an inconsistency. For example, a 1-to-3 weak inconsistency, simply means that one vendor uses 3 different labels, whereas the other vendor groups same malware samples in one set. Contrarily, strong inconsistencies are M-to-N relationships, and M or N are good indicators of the significance of the inconsistency: The more nodes are present in a connected component, the more complex the

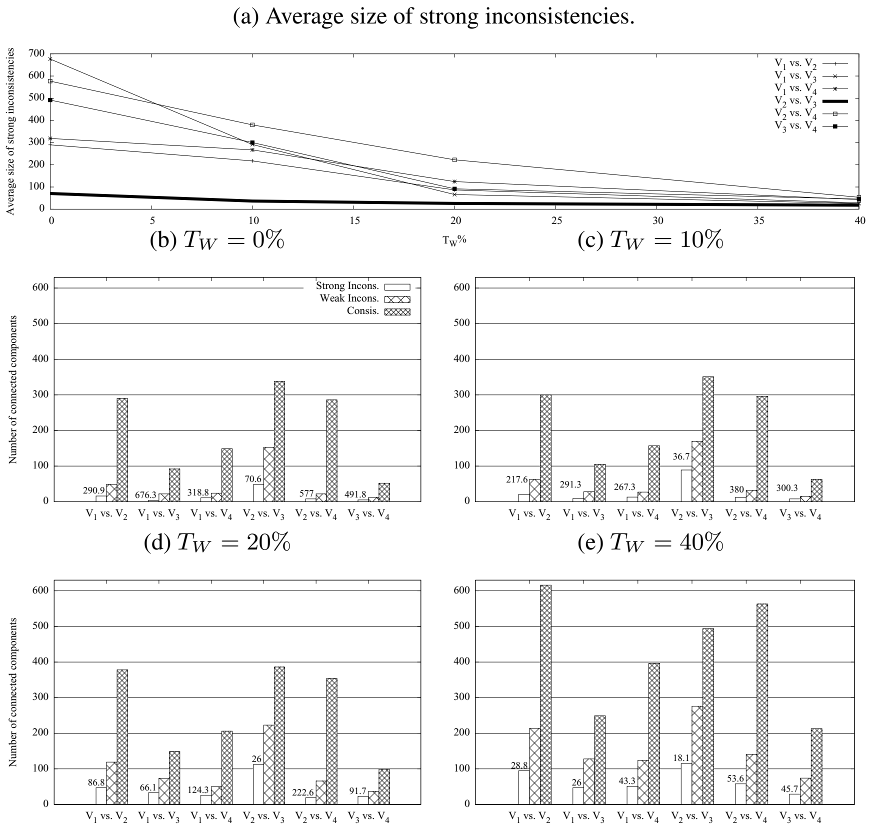
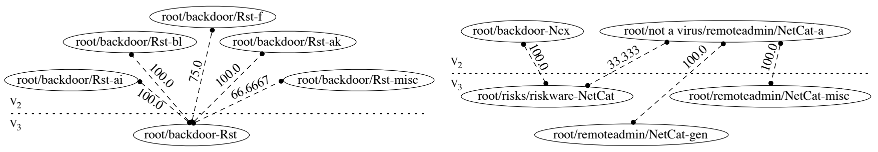


Fig. 6. Number of structural consistencies compared to strong and weak inconsistencies for different values of the edge weight threshold, T_W (see §3.3). For strong inconsistencies, the average number of inconsistent sets (i.e., those forming the graph's connected component) is reported. Note that, several small inconsistencies are preferable (because easier to analyze and resolve) as opposed to one, large inconsistency.

web of relationships between labels is. In general, many small, strong inconsistencies are better than one big, strong inconsistency: Small inconsistencies can be easily visualized, analyzed, and reduced to weak inconsistencies (e.g., by removing one or two nodes, or by fusing sets). We kept track of the size of the components that yield strong inconsistencies at different values of $T_W \in \{0\%, 10\%, 20\%, 40\%\}$, that is, we removed edges with weight below T_W from the weighted graphs. At $T_W = 0$ the comparison is irrealistic, as outliers may create spurious links, not reflecting the overall characteristic of naming conventions, thus leading to the wrong conclusion that many strong inconsistencies exist. Also, high values (e.g., $T_W > 50\%$) may produce biased (optimistic) conclusions, as relevant relations between naming conventions are excluded.

Fig. 6a shows the average size of strong inconsistencies: V_2 vs. V_3 are once again the most consistent vendors, with the lowest average size of strong inconsistencies (i.e., from 18.1 to 70.6). In Fig. 6(b–e), V_2 vs. V_3 show the highest number of consistencies



(a) Weak inconsistency (b) Strong inconsistency

Fig. 7. A real instance of a weak inconsistency (a) and strong inconsistency (b) between V_2 and V_3, which are the best-matching found. This randomly-selected weak inconsistency shows a case of name specialization, where V_2 uses finer labels than V_3.

(for $T_W < 40\%$) and inconsistencies, thus their graph is well-fragmented in many small consistencies and many small inconsistencies.

Although inconsistencies are generally more infrequent than consistencies, the number of strong inconsistencies is significant. This is exacerbated by the average size of strong inconsistencies, which is quite high. For instance, even at $T_W = 40\%$ some vendors have strong inconsistencies comprising up to 53.6 nodes on average. Comparing this observation with Fig. 5(b–d) (scatter measures), we note that the average number of sets that are scattered (across vendors) onto multiple sets is rather low. However, despite scatter is quite limited (e.g., less than 5 sets for some vendors), it often yields strong inconsistencies, because it occurs both from A to B and vice versa.

Examples of Found Inconsistencies. Fig. 7 shows two representative, real cases of strong and weak inconsistencies between $A = V_2$ and $B = V_3$, for $T_W = 0\%$. As mentioned in §3.3, weak inconsistencies indicate different granularities used by the vendors' labels that, in the lucky case of Fig. 7a, are easy to recognize. However, strong inconsistencies are less trivial to spot by comparing labels, as shown in Fig. 7b: This strong inconsistency is difficult to find by analyzing the labels, also because it involves multiple families (e.g., `NetCat` belongs to two different types: `riskware` and `remoteadmin` for the same vendor).

5 Conclusions

Our method is useful for finding inconsistencies as well as for comparing classifications (e.g., a ground truth vs. a classification produced by a novel approach being tested) by means of the number of inconsistencies contained. A non-intuitive result is that, when a vendor's set is inconsistently mapped onto several sets of another vendor, trying to map back those sets to the first vendor spreads the inconsistencies even further. In other words, there is no guarantee that a closed subset of malware on both sides that can be mapped consistently exists. This also entails, in our point of view, that any usage of such classifications as a ground truth for clustering techniques or other automated analysis approaches should be carefully evaluated.

Future work may address the fact that pattern extraction sometimes requires manual intervention to assign string patterns to appropriate. However, without vendor support, we had to manually analyze only a few tenths of labels. This limitation could be mitigated with community-driven efforts. Also, as malware naming conventions may change over time, we should incorporate a notion of "evolution" of a naming convention.

References

1. Carr, J.: Inside Cyber Warfare: Mapping the Cyber Underworld. O'Reilly Media, Inc. (2009)
2. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: WWW, pp. 281–290. ACM, New York (2010)
3. Kelchner, T.: The (in)consistent naming of malcode. Comp. Fraud & Security (2), 5–7 (2010)
4. Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 178–197. Springer, Heidelberg (2007)
5. Harley, D.: The game of the name malware naming, shape shifters and sympathetic magic. In: CEET 3rd Intl. Conf. on Cybercrime Forensics Education & Training, San Diego, CA (2009)
6. Goldberg, M.K., Hayvanovych, M., Magdon-Ismail, M.: Measuring similarity between sets of overlapping clusters. In: SocialCom, Minneapolis, MN (August 2010)
7. Tarjan, R.: Depth-First Search and Linear Graph Algorithms. SIAM J. on Comp. 1(2) (1972)

Taint-Enhanced Anomaly Detection*

Lorenzo Cavallaro[1] and R. Sekar[2]

[1] Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
[2] Department of Computer Science, Stony Brook University, USA

Abstract. Anomaly detection has been popular for a long time due to its ability to detect novel attacks. However, its practical deployment has been limited due to false positives. Taint-based techniques, on the other hand, can avoid false positives for many common exploits (e.g., code or script injection), but their applicability to a broader range of attacks (non-control data attacks, path traversals, race condition attacks, and other unknown attacks) is limited by the need for accurate policies on the use of tainted data. In this paper, we develop a new approach that combines the strengths of these approaches. Our combination is very effective, detecting attack types that have been problematic for taint-based techniques, while significantly cutting down the false positives experienced by anomaly detection. The intuitive justification for this result is that a successful attack involves unusual program behaviors that are exercised by an attacker. Anomaly detection identifies unusual behaviors, while fine-grained taint can filter out behaviors that do not seem controlled by attacker-provided data.

1 Introduction

System-call based anomaly detection has been popular among researchers due to its effectiveness in detecting novel application-layer attacks [1,4,7,8,10,13,26,29,30,32]. These techniques typically learn a model of an application's behavior during a training phase, which is then compared with behaviors observed during a detection phase. Deviations are flagged as potential intrusions. A key benefit of these techniques is that they require no policy specification. They are thus ideal for detecting unknown attacks.

The key assumption behind anomaly detection techniques is that attacks manifest unusual program behaviors. While experience to date supports this assumption, the converse does not hold: not all unusual behaviors are attacks. As a result, anomaly detection suffers from a high rate of false positives that impacts its practical deployment.

Recently, taint-tracking approaches [28,19,22,20,33] have become popular for defending against common software exploits. Their strength stems from their ability to accurately reason about the use of untrusted data (i.e., data that may be coming from an attacker) in security-critical operations. By using policies to distinguish between safe and unsafe uses of "tainted" data, these techniques can detect many common software vulnerability exploits, including those based on memory corruption [28,19,33], and SQL, command or script injection [20,22,27,33,25].

The main advantage of taint-tracking approaches is that accurate, application-independent policies can be developed for the above attacks. These policies express the

* This work was supported in part by an NSF grant CNS-0831298, an ONR grant N000140710928, and an AFOSR grant FA9550-09-1-0539.

general principle that *tainted data should be limited to non-control purposes;* and *control-data, such as code pointers, scripts, or commands, should be untainted.* Since attackers prize their ability to take control over a victim application, taint policy enforcement has proved to be very effective against the most popular attacks prevalent today. At the same time, due to the difficulty of developing accurate policies, many less popular (but still very dangerous) data attacks are not addressed by taint-based techniques, e.g., memory corruption attacks on non-control data [3], path traversals, race conditions. Anomaly detection remains the best option for such attacks.

We present a new technique, called *taint-enhanced anomaly detection* (TEAD), that combines the strengths of system-call-based anomaly detection with fine-grained taint-tracking. This combination is effective, detecting attack types that have been problematic for taint-based techniques, while significantly cutting down the false positives experienced by anomaly detection. The intuitive justification for this result is that a successful attack involves unusual program behaviors that are exercised by an attacker. Anomaly detection identifies unusual behaviors, while fine-grained taint can reduce false positives by filtering out behaviors that are not dictated by attacker-provided data.

As with any other taint-based technique, TEAD begins with a specification of the set of taint-sources and taint-sinks. Currently, our taint-sinks include all system calls and a few other functions such as `printf`, and functions used for communicating with external entities such as database servers or command interpreters. Like many previous techniques [26,4,8,7], our models rely on the contexts in which sink functions are invoked. This model is augmented with information about taintedness of arguments. In the simplest case, this information will indicate whether each argument of a sink function is tainted. More generally, the model captures information such as the components of aggregate data (e.g., fields of a C-structure, or components of a path name) that can be tainted, or the lexical structure of tainted data (e.g., whether tainted data should be alphanumeric or can contain various special characters). Attacks are flagged when there is a significant change from tainting patterns observed during training. Some of the advantages that TEAD can offer are:

1. Since tainted events correspond to a subset of events observed at runtime, TEAD's scope for false positives (FPs) is correspondingly reduced. In particular, TEAD can work with limited training data on untainted events since it triggers alarms only on tainted events. Note that a reduction in false positives can have an indirect effect on reducing false negatives (FNs), since a lower FP can allow the detection threshold to be lowered.
2. TEAD can be combined with more inclusive notions of taint, including those that account for control-flows. Previous taint-based vulnerability defenses have largely ignored control-dependencies, while also limiting taint propagation via pointers. This was done in order to reduce false positives. With TEAD, the training phase can help reduce these false positives by discarding those control dependences that were also observed during training.
3. TEAD is deployed in conjunction with taint policies to guard against most common exploits such as control-flow hijacks. This combination can mitigate problems faced by learning-based techniques due to attacks in training data—the most common attack types can be filtered out by removing event traces that violate

taint policies. Likewise, it can also improve resistance to sophisticated mimicry attacks [31,12], since these attacks rely on control-flow hijacks. Indeed, since taint-tracking involves reasoning about input/output data, TEAD does not suffer from a versatile mimicry attack [21] that can defeat previous system-call anomaly detection techniques that all ignored arguments to operations such as `read` and `write` [1,4,7,8,10,13,26,29,30,32].

2 Approach Description

2.1 Fine-Grained Taint-Tracking

TEAD relies on fine-grained taint information. In principle, it could be based on many of the existing techniques for this purpose, but clearly, performance is an important factor. For this reason, our implementation relies on DIVA [33], which is implemented using a source-to-source transformation of C programs. On server programs—often the focus of intrusion detection—DIVA has reported an overhead of about 5%. Binary taint-trackers exist as well [24], although they tend to be less mature and/or more expensive.

Like all taint-tracking techniques, DIVA relies on a specification of taint sources such as network read operations. In particular, on return from a system call that is a taint source, DIVA uses taint source specifications to mark the data returned by the system call as tainted. DIVA is a byte-granularity taint-tracker, and hence will mark each byte of data read from an untrusted source as tainted. This taint information is maintained in a global bit-array called `tagmap`. In particular, the taint associated with a byte of memory located at an address `A` is given by `tagmap[A]`. DIVA's source-to-source transformation ensures that data reads, arithmetic operations, and data writes are all augmented so as to propagate taint. DIVA can transform arbitrary C-programs, including various libraries, when their source-code is available. If source is unavailable, it can make use of summarization functions that capture taint propagation. In particular, after a call to a function f whose source code is unavailable, DIVA's transformation will introduce a call to f's summarization function in order to correctly propagate taint. (DIVA's taint-source marking specifications are a special case of summarization functions.)

Although DIVA's focus is on capturing direct data flows, it does incorporate features to track some control flows. These features enable it to correctly handle certain frequently encountered constructs such as the use of translation tables. This factor reduced the need for using full control dependency tracking in our implementation of TEAD. Even though DIVA operates on C-programs, it is applicable to many interpreted languages such as PHP and shell scripts. This applicability has been achieved by transforming the interpreters themselves to perform taint-tracking. As discussed before, TEAD enforces policies that block control-flow hijack, SQL injection and command injection attacks during its operation. This is done using the policies incorporated into DIVA as described in [33].

2.2 Taint-Enhanced Behavior Models

As with any other taint-based technique, TEAD begins with a specification of the set of taint-sources and taint-sinks. Currently, our taint-sinks include all system calls and a few other functions such as `main`, and `printf`, and functions used for communicating with external entities such as database servers or command interpreters. Unlike policy enforcement techniques such as DIVA that require policies to be associated with each sink, TEAD simply requires the identification of sinks. As a result, TEAD can handle numerous sinks without any significant specification efforts.

Since taint is a property of data, we focus primarily on learning properties of system call arguments. Let Σ be the set of all the sinks, and $s(a_1, a_2, \cdots, a_n) \in \Sigma$ be a generic sink, where $a_1, a_2, \cdots, a_n$ denote the sink's arguments. Rather than learning properties common to all invocations of s, our approach learns properties that are specific to each context in which s is invoked. In our prototype, the context is simply the calling location, except that calls made within shared libraries are traced back to program locations from which these library functions were invoked [26] (more refined notions of contexts, such as those of [4,8,7] could be used as well). The use of calling contexts increases the accuracy of models. For instance, a context-sensitive model can distinguish between `open` system calls made from two different parts of a program. If one of these is used to open a configuration file and the other one is used to open a data file, then it would be possible for the model to capture the intuitive property that a configuration file name cannot be controlled by the attacker, but a data file name may be.

As with other anomaly-based approaches, TEAD builds models during a *learning* phase. Deviations from this model are identified and reported as anomalies during a *detection* phase. Below, we detail the types of information embedded into TEAD models.

2.3 Coarse-Grained Taint Properties

For each argument a_i of each sink s, TEAD learns if any of its bytes are tainted. More generally, TEAD could learn whether a_i has control dependence, data dependence, or no dependence on tainted inputs, but control dependence is not tracked in our current prototype. For pointer arguments, taintedness of pointers could be learned, as well as the taintedness of the objects pointed by the pointer. At detection time, an alarm is raised if an argument that was not tainted during training is now found to be tainted. This approach can detect many buffer overflow attacks that modify system call arguments, as opposed to modifying control flows. For instance, an attack may overwrite a filename that is supposed to represent a file containing public data with `/etc/shadow`. This may allow the attacker to obtain the contents of `/etc/shadow` that he may subsequently use for an offline dictionary attack.

2.4 Fine-Grained Taint Properties

For aggregate data, the above approach may lose too much information by combining taint across the entire data structure. To improve precision, TEAD refines taint properties to capture more details regarding different parts of the data that may be tainted. The simplest case to handle in this regard are C-structures. For them, TEAD associates a taint with each field of a `struct`. This is particularly useful for some system calls, e.g., `sendmsg`, `writev`, etc. A more complex case concerns non-struct data, such as strings. String data, such as file names, database queries, scripts and commands are frequently targeted in data attacks. TEAD includes several algorithms for learning fine-grained taint properties that is motivated by such use of string data. We organize these algorithms based on whether a sink argument a_i is *fully* or *partially* tainted.

Properties of Fully Tainted Arguments

Maximum length (MaxTaintLen). This property is an approximation of the maximum permissible length l_{max} of a tainted argument. This helps to detect attacks that overflow buffers with the intent to overwrite security sensitive data.

Structural inference (StructInf). Often, an attacker may not try to overflow any buffers. Instead, he may try to modify the normal structure of an argument to bypass some security checks. To this end, the structure of a_i is inferred so that each byte is clustered in proper class. Currently, our model classifies (or maps) uppercase letters (A-Z) to the class represented by `A`, lowercase letters (a-z) to `a`, numbers (0-9) to `0`, and so on. Each other byte belongs to a class on its own. For instance, if the model sees an `open("/etc/passwd", ...)` system call invocation, the finite state automaton (FSA) which is generated for the string `/etc/passwd` will recognize the language `/a*/a*`. We further simplify the obtained FSA by removing byte repetition, as we are not concerned about learning lengths with this model. The final FSA will thus recognize the simplified language `/a/a`. If during detection the structure of the considered argument is different from the one learned, an alarm will be raised.

It can be noted that for particular sinks, trying to infer their (tainted) argument structure can lead to FPs if the structure for that sink is highly variable, e.g., arbitrary binary data read from the network. For this reason, our prototype limits fine-grained learning to those sinks and arguments where it is explicitly specified. An alternative possibility is to limit it to string data (i.e., `char *`).

Properties of Partially Tainted Arguments

In this case, a tainted argument consists of a combination of tainted and untainted bytes. The tainted portion is subjected to the learning of the aforementioned properties, while the following learning rules are considered for the untainted part.

Untainted common prefix (UCP). It is often the case for string-valued data that the interpretation of the trailing components is determined by the leading components. Examples include filenames which are interpreted within the directory specified by the leading components of the string; and command arguments that are interpreted by a leading command name. TEAD learns the longest common untainted prefix for every sink argument that is partially tainted. This algorithm can be easily generalized to learn a small number of common prefixes, rather than a single prefix.

Allowable set of tainted characters (ATC). Many applications expect tainted data will be limited to a subset of the alphabet. For instance, tainted components of filenames may be expected to be free of "`/`" and "`.`" characters in some contexts. In other contexts, they may be expected to be free of special characters such as "`;`" or whitespace characters. To capture such constraints, we can learn the set of characters that cannot appear in tainted context. To improve convergence, we utilize character classes as before, e.g., upper-case and lower-case characters and numbers. But most punctuation and whitespace characters form a class of their own.

3 Implementation

As mentioned before, our TEAD prototype relies on the DIVA implementation, which takes a program P as input and produces P_T, a semantically-equivalent taint-enhanced version of it. In particular, for *every* taint sink (or source) f, DIVA allows us to associate a wrapper f_w that will be called before (or after) it. Our prototype uses these wrappers to learn properties of sink arguments, as well as for marking source arguments as tainted (e.g., those coming from the network). We enable DIVA policies that protect against control-flow hijack and command injection attacks during detection phase, and to filter out attack-containing traces during training.

P_T is monitored during the *training* phase and a log file is created. The log file includes sink names and their context information (e.g., calling site), sink arguments and, for each argument, byte-granularity taint information. For instance, a typical log entry looks like the following:

```
read@0x8048f5c 3 arg0={ A:U } arg1={ A:U V[0-98]:T C:99:0:ls -la } arg2={ A:U }
```

The meaning is as follows. The sink name (`read`) is followed by its calling site (`0x8048f5c`). Next, the number of arguments follows (`3`) and details about these arguments are recorded. For instance, the entry for the second argument (`arg1`) tells us that the address (`A`) where the sink buffer of size 99 (`V[0-98]`) is stored is untainted (`A:U`), while the buffer content is tainted (`V[0-98]:T`). The content of the tainted buffer, which starts at offset 0, is `ls -la`[1]. This information will be used by the next step.

The log file is analyzed *off-line* to build a profile $\mathcal{M}$ of the behavior of P_T by using the aforementioned information. In particular, (i) identical events, that is events whose names and call sites are identical, are merged into a single event instance, and (ii) *untainted* events are inserted into the model just for evaluation reason.

For instance, considering the previous example, the tainted sink `read` invoked at the calling site `0x8048f5c` has its first and third argument untainted, while the second argument a_1 is tainted. Moreover, l_{max}, the maximum length for a_2 is 99 while, its structure is given by `a -a`. The profile created during this step is serialized and reloaded during the next step. P_T is then monitored during the *detection* phase. Deviations from the model learned in the previous step are reported.

4 Evaluation

4.1 Effectiveness in Detecting Attacks

TEAD main focus is on non-control data attacks. This section considers attacks taken from [3] and other sources. Where necessary, we slightly modify the example to show that our approach is effective even when some of the specifics of an attack are changed.

[1] To avoid noise into the log file we actually base64 encode buffer contents which are decoded by the off-line log analyzer to create the application profile.

Format String Attack against User Identity Data. A version of WU-FTPD is vulnerable to a format string vulnerability in the SITE EXEC command. The non-control data exploit of this vulnerability, described by Chen *et al.* [3], is based on the following code snippet that is used to restore the effective userid of the ftp server to that of the user logged in. (This restoration follows operations where the server briefly needed to utilize root privilege to perform setsockopt operation.)

```
1 FILE *getdatasock(...) {
2     ...
3     seteuid(0);
4     setsockopt(...);
5     ...
6     seteuid(pw->pw_uid);
7     ...
```

The attack aims to overwrite the pw_uid field to zero, so that the restoration code will leave the process with root-privileges. This would enable the current user, a non-privileged user, to assume root privileges on the system hosting the server.

Our approach detects this attacks in two different ways. It either considers whether the seteuid's argument is tainted, or it detects structural divergence in the tainted arguments of the printf-like function used to exploit the format string vulnerability. The latter method relies on the presence of a particular memory error vulnerability, and can be detected using taint policies as well. For this reason, we focus on the former method. In particular, our approach learns that the seteuid argument pw->pw_uid at line 6 was always *untainted* during training. During an attack, pw->pw_uid is marked as tainted, since it was overwritten by copying some attacker provided data. This causes the argument of seteuid to become tainted, thus raising an alarm.

It is worth pointing out that, in this context, taint-based learning seems to provide better results than what could be achieved with a conventional anomaly detection technique, even if the latter relies on very comprehensive training data. For instance, a conventional anomaly detector observing a limited number of authenticated users may be vulnerable to an attack where attackers are able to impersonate one of such users.

Heap Corruption Attacks Against Configuration Data. We report on two heap-based memory error vulnerabilities and attacks as described by Chen *et al.* [3].

Null HTTPD. This attack aims to overwrite the CGI-BIN configuration string. Note that the name of every CGI program invoked by a client will be prefixed with this string. Thus, by changing it from its default value of /usr/local/httpd/cgi-bin to the string /bin, a malicious client would be able to execute programs such as the shell interpreter on the server.

In this scenario, it can be observed that the available options for the attacker are mainly two: (a) to either completely overwrite the original CGI-BIN configuration string, or (b) partially overwrite the configuration string. In this latter case, the goal would typically be to perform a path traversal to ascend above the original CGI-BIN directory and then to descend into a directory such as /bin or /usr/bin. For simplicity, let us consider that the sink of interest here is the open system call.

(a) During training, our approach would learn that the first argument of `open` system call invoked at a context C is a combination of untainted data (i.e., the original `CGI-BIN` configuration string) and tainted data (i.e., the command derived from untrusted input); and has an *untainted prefix* that includes all of the original `CGI-BIN` string. Thus, the UCP model will detect this attack since the common untainted prefix observed during training is no longer present.

(b) In this case, the untainted prefix property may still hold. However, the set of allowable tainted characters (i.e., the ATC model) are different due to the presence of tainted characters such as "`.`" and "`/`". In addition, structural inference would have learned previously that the tainted component of `open` argument consisted of alphanumeric characters, whereas now it has a different structure that consists of a long, alternating sequence of alphanumeric and special characters.

Netkit Telnetd. The attack described in [3] exploits a heap-based buffer overflow vulnerability. It aims to corrupt the program name which is invoked upon login request by referencing the `loginprg` variable as shown by the following code snippet.

```
void start_login(char *host, ...) {
    addarg(&argv, loginprg);
    addarg(&argv, "-h");
    addarg(&argv, host);
    addarg(&argv, "-p");
    execve(loginprg, argv);
}
```

As a result of a successful attack, the application invokes the program interpreter `/bin/sh -h -p -p` (underlined characters are tainted). This raises an alarm in the UCP model: during detection, the untainted prefix contained the entire command name, which should be longer than the current untainted prefix `/bin`.

SquirrelMail Command Injection. Shell command injections can be somewhat tricky for policy-based techniques. In particular, a typical policy would be one that prevents tainted whitespace or shell metacharacters except inside quoted strings. Unfortunately, such a policy is susceptible to false positives as well as false negatives. False positives arise with applications that permit untrusted users to specify multiple command arguments. (SquirrelMail itself is one such application.) These applications may, in fact, be incorporating checks to ensure that the arguments are safe. On the other hand, false negatives may arise because a string literal may be passed down to a command that further parses the string into components that are handed down to another command interpreter, e.g., consider `bash -c 'ls -l xyz; rm *'`. For these reasons, we believe command injections are better handled by a technique such as TEAD that can utilize learning to fine-tune the characters that can legitimately appear within arguments.

Specifically, SquirrelMail version 1.4.0 suffered from a shell command injection vulnerability in version 1.1 of its GPG plugin. The vulnerability involves the use of a shell command to invoke `gpg` to encrypt email contents. `gpg` program needs to access the public key of the recipient, so the recipient name should be provided as a command-line argument. SquirrelMail retrieves the name of the recipient from the "to" field on the email composition form. An attacker can provide a malicious value for this email field, such as "`nobody; rm -rf *`" that causes SquirrelMail to delete files. This attack can easily be detected by the ATC model: in the absence of attacks, tainted characters in the `execve` argument will never include shell delimiters such as semicolons.

Stack Buffer Overflow Attack against User Input Data. The exploitation of this stack-based buffer overflow vulnerability was somewhat tricky but the authors of [3] were able to bypass the directory traversal sanity check enforced by the application. In summary, after the directory traversal check and before the input usage, a data pointer is changed so that it points to a second string which is not subjected to the application-specific sanity check anymore, thus it can contain the attack pattern (similar to a TOCT-TOU). Other than this TOCTTOU aspect, this attack is very similar to case (b) of the Null HTTPD attack, and hence is detected in the same way.

Straight Overflow on Tainted Data. The following example is from Mutz *et al.* [18]. The memory error attack is simple. The `user_filename` array obtained at line 7 (`gets` function) is guarded by a security check (`privileged_file` function at line 9) which checks whether `user_filename` specifies a name of a privileged file or not. In the affirmative case, the program prints an error message and quits. Otherwise (i.e., a non privileged file), more data is read into the array `user_data` using the function `gets` at line 14, and the file name specified by `user_filename` is opened at line 15. Instead of corrupting `write_user_data` return address, an attacker can overwrite past the end of `user_data` and overflow into `user_filename`. As the overflow happens after the security check performed at line 9, an attacker can specify a privileged file name for `user_filename` that will be replaced subsequently by the overflow attack.

```
 1 void write_user_data(void) {
 2
 3    FILE *fp;
 4    char user_filename[256];
 5    char user_data[256];
 6
 7    gets(user_filename);
 8
 9    if (privileged_file(user_filename)) {
10       fprintf(stderr, "Illegal␣filename.␣Exiting.\n");
11       exit(1);
12    }
13    else {
14       gets(user_data);         // overflow into user_filename
15       fp = fopen(user_filename, "w");
16       if (fp) { fprintf(fp, "%s", user_data); fclose(fp); }
17    }
18 }
```

Our approach detects this data attack by learning the maximum length l_{max} of the tainted arguments of the `gets` invoked at line 7, and 14, during the learning phase.

Format Bug to Bypass Authentication. The following example has been described by Kong *et al.* in [11]. Normally, the variable `auth` is set to 1 or 0 depending on the fact that the right authentication credential is given as input or not (line 5). An attacker, however, can exploit the format string vulnerability at line 11 and overwrite `auth` with a non-null value so that the subsequent check of the credential at line 12 will grant an access to the system.

```
1  void do_auth(char *passwd) {
2      char buf[40];
3      int auth;
4
5      if (!strcmp("encrypted_passwd", passwd))
6          auth = 1;
7      else
8          auth = 0;
9
10     scanf("%39s", buf);
11     printf(buf);            // format string
12     if (auth) access_granted();
13 }
```

This attack can be stopped by modeling tainted format string directives. By modeling the *tainted* format string of the `printf` function invoked at line 11 our approach learns whether tainted format directives have been used during the training step, along with their structure (structural inference on tainted arguments). If no tainted formatting directives are learned during the learning phase, then no tainted formatting directives can be subsequently encountered during detection phase without raising an alarm.

4.2 False Positives

Table 1 shows the false positives rate we obtained by conducting experiments on the `proftpd` ftp server and `apache` web server. Table 2 attributes these false positives to each of the models used.

Table 1. Overall False Positives

Program	# Traces (Learning)	# Traces (Detection)	Overall FP rate
proftpd	68,851	983,740	1.7×10^{-4}
apache	58,868	688,100	2.5×10^{-3}

As shown by Table 2, the majority of false positives were caused by violation of structural inference models. We expected a relatively high number of false positives as the model proposed in § 2 is a simple non-probabilistic model. The main focus of this paper is to show the benefits of using taint information to improve the false positive rates of anomaly detection, so we have not emphasized the use of sophisticated learning techniques. The use of more sophisticated learning techniques (e.g., [17,1]) is orthogonal to our technique, and can further reduce false positives.

Table 2. False Positives Breakdown

Program	Tainted Events	UCP	StructInf	MaxTaintLen	Overall FP Rate
`proftpd`	3.0×10^{-5}	0	1.4×10^{-4}	0	1.7×10^{-4}
`apache`	0	0	2.4×10^{-3}	0	2.5×10^{-3}

To assess the effectiveness of taint information in reducing the false positives of learning-based anomaly detection, we carried out the following test. We computed the false positive rate of the anomaly detection techniques underlying TEAD, when taint information is ignored. In particular, an alarm is counted each time a system call s is invoked in a context different from those observed during training. This led to false positive rates of 2.4×10^{-4} and 4.3×10^{-4} for `proftpd` and `apache` respectively. We then compute the false positive rate that would be observed when taintedness of the argument was taken into account. In particular, an alarm was raised only if the anomalous system call was also associated with anomalous taint, i.e., a previously untainted argument was now tainted. This reduced the false positives on these programs to 3.0×10^{-5} and zero respectively. Thus, the use of taint information reduces the false positive rate by about an order of magnitude or more.

As a second way to assess the impact of taint-tracking on false positives, we compared the fraction of system calls that were tainted during the learning and detection phase. As Table 3 depicts, half of the traces of `apache` have been considered during detection, while only a small fraction of them have been considered for `proftpd`. By omitting the rest, TEAD can avoid FPs that may arise due to them.

Table 3. Fraction of tainted system calls

Program	# Traces (Learning)	# Tainted (%)	# Traces (Detection)	# Tainted (%)
`proftpd`	68,851	2,986 (4.3%)	983,740	7,120 (0.72%)
`apache`	58,868	46,059 (82.1%)	688,100	354,371 (51.5%)

4.3 Performance Overheads

The dominant source of overhead in TEAD is that of taint-tracking. Overhead due to DIVA has been reported in [33] to be about 5% for I/O-intensive applications such as `apache`, and about 50% for CPU-intensive applications. Most real-world applications experience overheads in between these two figures.

Our preliminary results indicate that the additional overhead introduced by TEAD is relatively low. So far, we have measured only the overheads due to `proftpd`, but we hope to measure `apache` overhead in the near-future. In particular, for `proftpd`, we experienced slowdowns of 3.10% due to taint-tracking only, 5.90% due to taint-tracking and model profiling during the learning phase, and 9.30% due to taint-tracking and model matching during the detection phase. `proftpd` overheads were measured when the program was subjected to a variety of operations, including changing of directories, file uploads and downloads, and recursive directory listing.

Note that taint-learning overhead includes only the overhead of logging, and omits the cost of offline learning that uses these logs. In contrast, detection is performed online, and hence its overheads are higher.

We point out that unlike DIVA, whose overhead increases for CPU-intensive computation, TEAD's anomaly detectors experience higher overheads for I/O-intensive computations, e.g., computations characterized by a high rate of system calls.

5 Related Work

Anomaly Detection Based on System Calls. Forrest *et al.* [6,9] first introduced anomaly detection techniques based on system calls made by applications. This system is built following the intuition that the "normal" behavior of a program P can be characterized by the sequences of system calls it invokes during its executions in an attack-free environment. In the original model, the characteristic patterns of such sequences, known as N-grams, are placed in a database and they represent the language L characterizing the normal behavior of P. To detect intrusions, sequences of system calls of a fixed length are collected during a detection phase, and compared against the contents of the database. This technique was subsequently generalized in [32] to support variable-length system-call sequences.

The N-gram model is simple and efficient but it is associated with a relatively high false alarm rate, mainly because some *correlations* among system calls are not captured in the model. Furthermore, it is susceptible to two types of attacks, namely *mimicry* [31] and *impossible path execution* (IPE). Newer algorithms have since been developed to address these drawbacks, primarily by associating additional context with each system call. Reference [26] uses the location of system call invocation as the calling context, while References [4] and [7] can potentially use the entire list of return addresses on the call stack at the point of system-call invocation. Techniques have also been developed that rely on static analysis for building models [30,8], as opposed to learning.

These newer models make mimicry and IPE attacks harder, but they still remain possible. In particular, the use of calling contexts make mimicry attacks difficult: although an attacker may be able to make one system call using an exploit, the attack code will not be able to resume control after the execution of this system call. This is because the IDS will otherwise observe a return address on the stack that resides in attacker-provided code, and hence would not be in the IDS model. Kruegel *et al.* then devised a clever approach [12] that relied on corrupting data items such as saved register contents or local variables in order to reacquire control after making a system call. Moreover, Chen et al [3] demonstrated powerful attacks that don't modify control flows at all — instead, they only change non-control data, yet achieve the same end goals achieved by (the more popular) control-flow hijack attacks.

The above developments focused more research efforts on techniques that incorporate system call argument data into IDS models [13,29,1,17,18,15]. Unfortunately, since most of these techniques do not reason about bulk data arguments such as the data read from (or written to) files or the network, they remain vulnerable to a class of mimicry attacks [21]. This attack works on any *I/O-data-oblivious IDS,* i.e., IDS that may possess perfect knowledge about system calls, their sequencing, and their argument values, with the singular exception of data buffer arguments to read and write operations. Since TEAD examines read buffers and write buffers to check for anomalous taint, it is *not* I/O-data-oblivious, and hence this attack is not applicable to TEAD.

Dependency and Taint-Based Techniques. The core idea behind TEAD was described in an abstract in [2]. This paper represents the full development of those core ideas.

Dataflow anomaly detection [1] provides an interesting contrast with TEAD. In particular, dataflow anomaly detection uses learning to infer information flows. For this reason, it focuses on so-called binary relations that capture relationships between the arguments of different system calls. In contrast, TEAD relies on actual information flows present in a program. Since it has access to information flow already, it uses only *unary relations* in the terminology of dataflow anomaly detection, and does not consider relationships between arguments of different system calls. Key benefits of dataflow anomaly detection over TEAD are: (a) it does not require access to source code, and (b) it has much lower overheads. On the other hand, taint-tracking is much more reliable as compared to dataflow inference, which should lead to a lower false positive rate for TEAD.

Whereas dataflow anomaly detection focuses on security-sensitive data such as file names and file descriptors, dataflow inference [25] is concerned with inferring more general dataflows, e.g., between the data read and written by an application. This necessitates the use of more powerful matching algorithms as compared to the simpler (exact or prefix-matching) algorithms used in dataflow anomaly detection. While dataflow inference can provide low overheads and avoids the need for heavy-weight instrumentation, it is limited to applications that do not perform complex transformations on inputs.

SwitchBlade [5] has some similarity with TEAD in combining taint-tracking with system call anomaly detection. However, the similarity is only superficial since our goals as well as the techniques are quite different. In particular, TEAD uses taint information to improve attack detection and false positives of a typical system call anomaly detector. SwitchBlade's main goal is not one of improving anomaly detection, but instead, to reduce the overhead of taint-based policy enforcement techniques [28,19,33]. In particular, they aim to stop control-flow hijacks that are prevented by [28,19,33], but do so without the overheads of runtime taint-tracking. They achieve this by using system call anomaly detection as a low-overhead filter to screen out potential exploits from normal behaviors. These potential exploits are then verified by replaying them on a taint-tracked version of the victim process. If a taint policy violation is observed during replay, an attack is reported. Otherwise, the new behavior is added to the IDS model. To reduce the likelihood of missing exploits, SwitchBlade develops new techniques that personalize system-call IDS models to each deployment site, and injects random system calls into the model.

Sarrouy et al [23] also observe that attacks result from tainted data that feeds into system calls. However, their technical approach is quite different from ours. In particular, their approach does not rely on system call models, but instead, captures invariants involving a program's internal state that may hold at various points during execution.

Ming et al [16] are concerned with the problem of improving the accuracy of so-called gray-box anomaly detectors that focus on data [13,29,1,14]. In particular, these techniques may end up learning properties that were observed during training but do not necessarily hold in general. Ming et al show that taint-tracking can resolve such questions, and establish whether training observations truly support the rules learned by an anomaly detector. In contrast, our work shows that by leveraging taint information, we can extend the class of attacks that can be reliably detected by an anomaly detector.

In summary, although numerous works have studied learning-based anomaly detection and information-flow tracking, none of them have considered our approach of augmenting anomaly detection models with taint data in order to reliably detect non-control data attacks that have been challenging for all previous intrusion detection techniques.

6 Conclusion

In this paper, we presented a new approach which combines fine-grained taint-tracking and learning-based anomaly detection techniques. By exploiting the information provided by the taint-tracking component, our approach was able to detect a variety of non-control data attacks that have proved to be challenging for previous intrusion detection or policy enforcement techniques. False positives, one of the main drawbacks of learning-based approaches, are caused due to the fact that training can never be exhaustive. Our approach limits this drawback as it considers only tainted traces, which usually are a small percentage of the whole traces executed by an application. Our evaluation results show that the false positive rate of a learning-based approach is reduced by about a factor of ten due to the use of taint-tracking.

References

1. Bhatkar, S., Chaturvedi, A., Sekar, R.: Dataflow anomaly detection. In: IEEE Security and Privacy (2006)
2. Cavallaro, L., Sekar, R.: Anomalous taint detection (extended abstract). In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 417–418. Springer, Heidelberg (2008)
3. Chen, S., Xu, J., Sezer, E.C., Gauriar, P., Iyer, R.K.: Non-Control-Data Attacks Are Realistic Threats. In: USENIX Security Symposium (2005)
4. Feng, H., Kolesnikov, O., Fogla, P., Lee, W., Gong, W.: Anomaly Detection using Call Stack Information. In: IEEE Symposium on Security and Privacy (2003)
5. Fetzer, C., Susskraut, M.: Switchblade: enforcing dynamic personalized system call models. In: EuroSys (2008)
6. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A Sense of Self for Unix Processes. In: IEEE Symposium on Security and Privacy (1996)
7. Gao, D., Reiter, M.K., Song, D.: Gray-box extraction of execution graphs for anomaly detection. In: ACM CCS (October 2004)
8. Giffin, J.T., Jha, S., Miller, B.P.: Efficient context-sensitive intrusion detection. In: NDSS (2004)
9. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion Detection Using Sequences of System Calls. Journal of Computer Security (1998)
10. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. Journal of Computer Security (JCS) 6(3), 151–180 (1998)
11. Kong, J., Zou, C.C., Zhou, H.: Improving Software Security via Runtime Instruction-level Taint Checking. In: Workshop on Architectural and System Support for Improving Software Dependability (2006)

12. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Automating Mimicry Attacks Using Static Binary Analysis. In: USENIX Security Symposium (2005)
13. Kruegel, C., Mutz, D., Valeur, F., Vigna, G.: On the detection of anomalous system call arguments. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 326–343. Springer, Heidelberg (2003)
14. Li, P., Park, H., Gao, D., Fu, J.: Bridging the gap between data-flow and control-flow analysis for anomaly detection. In: Annual Computer Security Applications Conference (2008)
15. Liu, A., Jiang, X., Jin, J., Mao, F., Chen, J.: Enhancing System-Called-Based Intrusion Detection with Protocol Context. In: IARIA SECURWARE (August 2011)
16. Ming, J., Zhang, H., Gao, D.: Towards Ground Truthing Observations in Gray-Box Anomaly Detection. In: International Conference on Network and System Security (2011)
17. Mutz, D., Valeur, F., Kruegel, C., Vigna, G.: Anomalous System Call Detection. ACM Transactions on Information and System Security 9(1), 61–93 (2006)
18. Mutz, D., Robertson, W., Vigna, G., Kemmerer, R.A.: Exploiting Execution Context for the Detection of Anomalous System Calls. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 1–20. Springer, Heidelberg (2007)
19. Newsome, J., Song, D.X.: Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In: NDSS (2005)
20. Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., Evans, D.: Automatically Hardening Web Applications Using Precise Tainting (2005)
21. Parampalli, C., Sekar, R., Johnson, R.: A practical mimicry attack against powerful system-call monitors. In: AsiaCCS (2008)
22. Pietraszek, T., Berghe, C.V.: Defending Against Injection Attacks Through Context-Sensitive String Evaluation. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 124–145. Springer, Heidelberg (2006)
23. Sarrouy, O., Totel, E., Jouga, B.: Building an Application Data Behavior Model for Intrusion Detection. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 299–306. Springer, Heidelberg (2009)
24. Saxena, P., Sekar, R., Puranik, V.: Efficient fine-grained binary instrumentation with applications to taint-tracking. In: CGO (April 2008)
25. Sekar, R.: An efficient black-box technique for defeating web application attacks. In: NDSS (2009)
26. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors. In: IEEE Symposium on Security and Privacy (2001)
27. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: POPL (2006)
28. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S.: Secure Program Execution via Dynamic Information Flow Tracking. In: ASPLOS (2004)
29. Tandon, G., Chan, P.: Learning rules from system call arguments and sequences for anomaly detection. In: on Data Mining for Computer Security (2003)
30. Wagner, D., Dean, D.: Intrusion Detection via Static Analysis. In: IEEE Symposium on Security and Privacy (2001)
31. Wagner, D., Soto, P.: Mimicry Attacks on Host Based Intrusion Detection Systems. In: ACM CCS (2002)
32. Wespi, A., Dacier, M., Debar, H.: Intrusion detection using variable-length audit trail patterns. In: Debar, H., Mé, L., Wu, S.F. (eds.) RAID 2000. LNCS, vol. 1907, pp. 110–129. Springer, Heidelberg (2000)
33. Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced Policy Enforcement: a Practical Approach to Defeat a Wide Range of Attacks. In: USENIX Security Symposium (2006)

Secured Cloud Storage Scheme Using ECC Based Key Management in User Hierarchy

Atanu Basu[1], Indranil Sengupta[1], and Jamuna Kanta Sing[2]

[1] Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur 721302, India
{atanu,isg}@cse.iitkgp.ernet.in

[2] Department of Computer Science and Engineering
Jadavpur University, Kolkata 700032, India
jksing@ieee.org

Abstract. In our proposed scheme, the data owner outsources huge volume of data to a cloud storage provider and the end users request data to the data owner. The data owner encrypts the data before sending it to the cloud service provider and does over-encryption proposed by Vimercati et al. [4] to the outsourced encrypted data before sending it to the users. We incorporate an existing Elliptic Curve Cryptography (ECC) based key management scheme in user hierarchy proposed by Nikooghadam et al. [11] in our scheme which classifies users in security classes and efficiently helps to derive the secret keys of the lower order security classes. The cloud storage component of our scheme will not have to perform any extra work except storing data and this reduces the cost of the data owner as per pay-per-use pricing policy of the cloud service provider. Our scheme incurs low overhead for key generation as well as for its storage and the end users can use wireless mobile devices. The scheme is useful in owner-write-users-read applications and it is secured from the adversaries.

Keywords: Cloud storage, data owner, trusted dealer, ECC, adversary.

1 Introduction

In recent times, the Information Technology (IT) companies and research communities are putting their effort on cloud computing [1,2,3] for evolving suitable, trustworthy and usable platform for the users. The cloud service or cloud computing refers to providing scalable IT resources over the Internet instead of hosting and operating the resources locally but with the feeling that they are using the resources locally. The cloud computing is suitable for large amount of data storing and extensive data processing purposes where the data centers are spread over geographically distributed locations. The growing cost of in-house storage as well as management of large pool of data and software purchase as well as license renewal cost have opened the door for cloud service providers. This also helps better management of IT resources against any disaster and for rapidly growing IT companies cloud computing has also emerged as viable good solution for provisioning IT resources on demand. But, at the same time, outsourcing of data to third parties raise the questions like availability, confidentiality, integrity of the sensitive data or information of any company. Nowadays, Amazon S3, Microsoft's Azure are providing cloud storage service to customers with scalability and dynamic features.

The cloud service providers provide the following main services -

- **Infrastructure as a Service (IaaS)**. Here, the IT resources like operating system, middleware packages like Microsoft .Net and data storage facility on demand with administrative capability are provided. The users avail the cloud facility through virtualization, e.g. using software package like VMWARE.
- **Platform as a Service (PaaS)**. Here, facilities like development of applications with middleware packages, Web development tools etc are provided. The PaaS also provides the facilities like running of applications, hosting of Web services, data storage.
- **Application as a Service (AaaS)**. Here, some applications like e-mail usage with limited administrative capability and storage facility are provided.
- **Data as a Service (DaaS)**. Here, the large volume of data or database of a company is outsourced to the cloud service providers. The clients will access the data securely on demand.

The deployment models for cloud service are mainly three types –

- **Private cloud**. Here, the resources are kept behind the firewall of any company.
- **Public cloud**. Here, the resources are located in geographically distributed areas and the resources are accessed through Internet.
- **Hybrid cloud**. Here, though the data centers are located in different locations, the resources may be accessed through Internet with secure channel.
- **Community cloud**. Here, several organizations or companies having similar requirements seek to share IT resources to realize some of the benefits of cloud computing.

We focus our work in DaaS [4,5,6,7] for public cloud in owner-write-users-read application areas. Generally, in these applications large volume of data is created and updated by the owner and the readers with different access rights can efficiently as well as securely read the data. As an example, in a digital library, the subscribed readers can access the files of a digital library who are having different access rights whereas the owner of the digital library can create or update the contents of the digital library hosted on a secure cloud storage. In another owner-write-users-read example, the valuable data or information created or updated by various research groups of an organization which are located in different countries of the world can be outsourced to a cloud storage in encrypted form. The scientists of different countries with different security clearance can access data of different groups securely. The access rights of the scientists can be changed dynamically or can be revoked as usual.

The motivation of our work is to propose a cost effective secured cloud storage system suitable for owner-write-users-read applications where the end users should be able to use resource constrained wireless mobile devices securely.

In our proposed scheme, the owner outsources huge volume of data to a cloud storage by encrypting them. When a user requests data to the owner, the owner fetches the encrypted data from the cloud storage. Subsequently, the owner over-encrypts [4] the encrypted data by the shared key of that user which has been already pre-distributed to that user. It is required to enforce an user access policy combined with cryptography in this scenario so that the users who are in different hierarchical levels can access the data securely. The whole user set in our proposed scheme has been classified into security classes with hierarchical access rights. An elliptic curve cryptography (ECC) [8,9,10] based dynamic key management scheme [11] with access rights in hierarchy is used in our proposed scheme. This key management scheme helps to reduce computational, communication and storage overheads of our proposed scheme compared to other ECC based key management schemes. The end users with higher security clearance can access the same data securely assigned for members of the lower security classes by deriving their keys directly and the revoked users cannot access the updated data. Our scheme is protected from adversaries or attackers and the end users can use resource constrained wireless mobile devices. The proposed scheme is also cost effective as the data owner minimizes the access, processing and storage overheads to the cloud. The schematic diagram our scheme is shown in Figure 1.

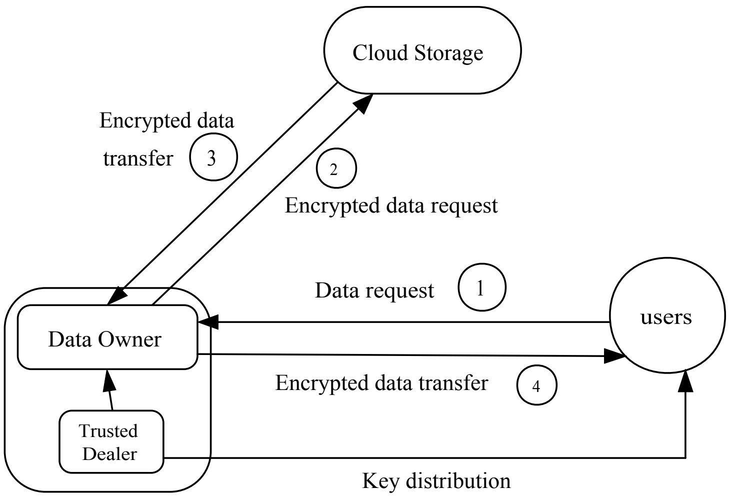


Fig. 1. Block diagram of our proposed scheme

2 Related Work

Similar to our work, Wang et al. [7] efficiently proposed a secured cloud storage scheme where the data owner stores huge volume of data in a cloud storage in encrypted form but here each data block of the whole data is encrypted with different key [12]. Any user requests to the data owner for the data and the data owner sends the access certificate to the end user as well as to the cloud service provider. The end user submits the access certificate to the cloud service provider for accessing the requested data. The cloud service provider does over-encryption [4] using one time key pad and for denial of over-encryption the data owner does lazy revocation [13]. The cloud service provider transfers the requested data to the user on matching both the received access certificates of the data owner and the end user. The user derives large number of keys from a key which exists as the root of the key hierarchy to decrypt blocks of encrypted data. But the scheme [7] depends on the factors like whether the cloud service provider performs the added responsibilities like over-encryption, validation of the access certificate of the end users and also putting the extra burden on the service provider will not be cost effective for the owner as per pay-per-use pricing policy of the cloud service provider. This scheme is suitable for resource constrained wireless mobile devices.

In Vimercati et al. [5] scheme, authorizations and encryption of the data are merged and outsourced to the cloud storage. This helps to allow enforcement of access control with the outsourced data and the data owner while formulating the access policy need not be involved with its enforcement. Two layers of encryption are used, one is BEL (Base Encryption Layer) which is used by data owner and another is SEL (Surface Layer Encryption) termed as over-encryption which is used by the cloud storage server. The end users receive two decryption keys - one for SEL decryption and another is for BEL decryption for decryption of the requested data. The over-encryption must be conducted to the data when there are changes of access rights of the users as well as updation of the data. The scheme is not collusion free and also not mentioned whether it is suitable for resource constrained wireless mobile devices.

3 Review of Key Management Schemes with Access Control in Hierarchy

The key management schemes [14,15,16,17,18,19,20,11,12] with access control in hierarchy propose derivation of key of any lower security class from a higher level security class but the key derivation of higher level security class from lower level security class is not possible. As an example, we consider the whole user set SC is classified into disjoint sets, e.g. $SC = \{SC_1, SC_2, \ldots, SC_n\}$ where n is the total number of security classes and the user sets can be arranged into hierarchical tree according to the security clearance as shown in Figure 2. We also consider the security keys corresponding to the security classes are $SK = \{SK_1, SK_2, \ldots, SK_n\}$. Again, we assume security classes are partially ordered by a binary relation '$\leq$' in such a way that if $SC_j \leq SC_i$ where $i, j \in n$ then the users in SC_i can access the information of the users who are in SC_j but reverse is not possible. In this hierarchical access structure, the users of SC_1 can access information of $\{SC_2, SC_3, SC_4, SC_5, SC_6, SC_7, SC_8\}$ and users of SC_2 can access information of $\{SC_5, SC_6\}$. If SC_1 wants to access data of those security classes either she will have to possess the cryptographic keys $\{SK_2, SK_3, SK_4, SK_5, SK_6, SK_7, SK_8\}$ with her own secret key SK_1 or will have to derive the other keys from SK_1. The derivation of secret keys of successor security classes of any security class will have to be done through iterations or directly with the help of public information or tokens.

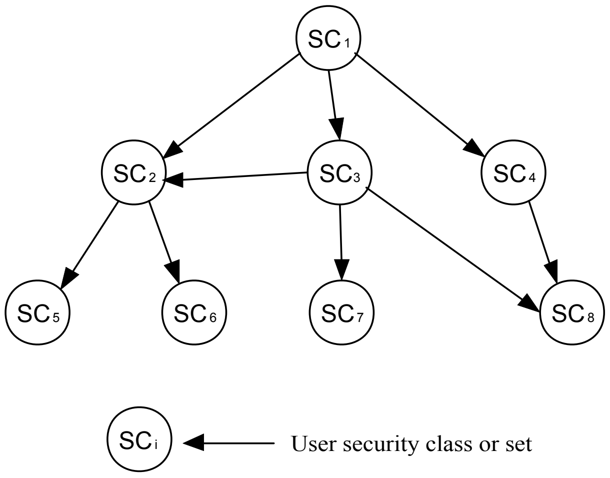


Fig. 2. Hierarchical tree for user classes

Akl and Taylor [14,15] proposed their scheme with much simplicity but it requires large storage overhead for storing public information for generation and derivation of keys of the lower level security classes. But their scheme is not dynamic in nature, e.g. new security classes cannot be added. The schemes [16,17,18] use iterative methods to derive the keys of lower security classes in the hierarchy. The schemes [17,18] use discrete logarithm method for generation of keys of the security classes and efforts are made to minimize the storing of public information with the incorporation of dynamic features, e.g. changing the keys of any security class, adding or deleting the security classes. The schemes [19,20] proposed their ECC based schemes with dynamic features efficiently. But, these schemes use polynomial methods to derive the keys which impose computational and storage overheads.

Nikooghadam et al. [11] proposed their ECC based scheme which derives the keys of security classes directly who are in lower hierarchical order with the help of public information without using any polynomial computation method. This scheme is cryptographically secure and efficient than other ECC based

schemes [19,20] and we incorporate this dynamic key management scheme with our proposed secured cloud storage scheme. This ECC based secret key distribution scheme in user hierarchy is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP). The problem is defined in Definition 1 below.
Definition 1 : Let A, B be two points on the elliptic curve of base point G with order n ($O = n.G$, where O is the point at infinity) and n is the prime number. Then the point $B = k.A$, where k in an integer and $k < n$. Find the value k where the points A and B are known.

It is computationally infeasible or hard to find the value k till now where A and B are known. The problem is termed as ECDLP [8,9].
The description of the scheme [11] is given concisely -

3.1 Setup Phase

The following steps are followed -

Step - 1 : The TD chooses a secure elliptic curve (C) over prime field, GF(p), where p is a prime number and the base point G of order q where $q \geq 163$ bits.

Step - 2 : After that, the TD chooses its private key, d_{TD} where $d_{TD} \in [1,\ q-1]$ and corresponding public key P_{TD} where $P_{TD} = d_{TD}.G$. The TD also selects unique private key d_i where $d_i \in [1,\ q-1]$ and the corresponding public key P_i where $P_i = d_i.G$ for the SC_i where $1 \leq i \leq n$. The private keys of the SC_i are distributed securely to the members of the SC_i through a secure channel, e.g. postage system.

Step - 3 : The TD computes $Z_i = k_i.G$ where k_i is a random integer chosen from $[1,\ q-1]$ for each security class SC_i and the TD also computes key $SK_i = H(Z_i)$ where H is the one way hash function which converts x-coordinate of the point Z_i on the elliptic curve to secret key SK_i.

Step - 4 : For the security classes which satisfies $SC_j \leq SC_i$ where $1 \leq j \leq n$, the TD computes the point $Y_{i,j} = k_j.P_i$ and transfers the points $Y_{i,j}$s to the corresponding classes through secure channel.

Step - 5 : The TD publishes p, q, G, P_i, P_{TD}, the hash function H and keeps her private key d_{TD} as well as the keys SK_is, k_is for all the security classes in secured place but removes private keys d_is of the security classes.

3.2 Key Generation Phase

The following steps are followed -

Step - 1 : Each security class computes inversion of her private key d_i, i.e. d_i^{-1} and stores it securely.

Step - 2 : Each security class SC_i computes $Z_i = d_i^{-1}.Y_{i,i}$ for self and $Z_j = d_i^{-1}.Y_{i,j}$ for the security classes SC_j ($SC_j \leq SC_i$, $1 \leq j \leq n$) as $Z_j = k_j.Y_{i,j}$. It is proved that,
$Y_{i,j} = k_j.P_i = k_j.(d_i.\ G)$
$d_i^{-1}.Y_{i,j} = d_i^{-1}.(k_j.(d_i.\ G)) = k_j.G = Z_j$
$\therefore Z_j = d_i^{-1}.Y_{i,j}$

Step - 3 : The security class SC_i computes the secret key $SK_i = H(Z_i)$ where H is the one way hash function which converts x-coordinate of the point Z_i on the elliptic curve to security key SK_i.
As an example, the security class SC_3 (Figure - 2) can access the information of the classes $\{SC_2, SC_5, SC_6, SC_7, SC_8\}$ who are in lower hierarchical order and she will have to generate the keys of those classes, i.e, $\{SK_2, SK_5, SK_6, SK_7, SK_8\}$. First the related $Y_{i,j}$s will have to transfer to the SC_3 by the TD.
$SC_3 : Y_{3,2} = k_2.P_3$, $Y_{3,5} = k_5.P_3$, $Y_{3,6} = k_6.P_3$, $Y_{3,7} = k_7.P_3$, $Y_{3,8} = k_8.P_3$, $Y_{3,3} = k_3.P_3$.
Now, the security keys of the lower order security classes can be derived by SC_3 as
$SK_2 = H(Z_2) = H(d_3^{-1}.Y_{3,2})$, $SK_5 = H(Z_5) = H(d_3^{-1}.Y_{3,5})$,
$SK_6 = H(Z_6) = H(d_3^{-1}.Y_{3,6})$, $SK_7 = H(Z_7) = H(d_3^{-1}.Y_{3,7})$,
$SK_8 = H(Z_8) = H(d_3^{-1}.Y_{3,8})$ and $SK_3 = H(Z_3) = H(d_3^{-1}.Y_{3,3})$ for self.

3.3 Changing Secret Key of a Security Class

Any security class SC_j may need to change its secret key SK_j. For this purpose, the TD selects k_j^* from $[1, q-1]$ and computes $SK_j^* = H(k_j^*.G)$. For each security classes which satisfies $SC_j \leq SC_i$, the TD computes the points $Y_{i,j}^* = k_j^*.P_i$ and sends the values as described in Section 3.1.

The other dynamic properties like addition of new security classes, removing existing security classes, creating new relationship between security classes and revoking existing relationships can be done efficiently through this scheme.

4 Proposed Cloud Storage Scheme

In this owner-write-users-read model, we discuss how the end users securely access the required data and the revoked users cannot access the updated information. As the cloud storage service provider may follow pay-per-use pricing policy model, we will also discuss how the access to cloud storage is minimized.

4.1 System and Network Model

The system model of our scheme consists of three components (Figure - 1) -

Data Owner (DO) : The DO encrypts the data and sends those to the cloud storage provider. It is also responsible for classification of data and user security classes. The DO serves data as requested by the valid users. We consider that the trusted dealer (TD) and the DO are in the same unit. The TD generates and distributes secret keys to the users and the DO. We assume that this system is not compromised by any adversary and is trusted by all the users as well as the DO.

Cloud Service provider (CS) : We assume that the CS can store huge amount of data as required by the data owner. We also assume that the CS may not be trusted but it will not modify or tamper any data and faithfully transfers any data as requested by the DO. We also assume that the CS will not try to open any file by colluding with any member of a security class.

End Users : The valid end users U_v request data from the DO.

The DO and the CS are connected through the dedicated secure channel, e.g. secure leased line. The readers may be connected through the wired or wireless medium through the public channel to the DO. The readers in our scheme may use resource constrained wireless mobile devices.

We consider various types of active and passive adversaries or attackers who try to acquire or modify data of the DO, the CS and the U_v or may capture data from the wireless transmission media.

4.2 Detail Description of the Cloud Storage Model

4.2.1 Setup Phase

The following steps are followed by the TD and the DO -

Step - 1 : All the users $U = \{U_1, U_2, \ldots, U_N\}$ registers to the TD where N is the total number of users. The users are classified into different disjoint security classes according to their hierarchy as shown in Figure - 2, i.e. $SC = \{SC_1, SC_2, \ldots, SC_n\}$.

Step - 2 : The DO classifies the whole data into different disjoint data sets, i.e. $DS = \{DS_1, DS_2, \ldots, DS_n\}$.

Step - 3 : The TD generates secret key SK_i for each security class SC_i where $1 \leq i \leq n$, i.e. $SK = \{SK_1, SK_2, \ldots, SK_n\}$ and the parameters $Y_{i,j}$s for each security class SC_i $(SC_j \leq SC_i,\ 1 \leq j \leq n)$ as described in Section 3.1 and 3.2.

Step - 4 : The TD randomly selects shared secret key SK_v^{pw} for the users $1 \leq v \leq N$ where $SK_v^{pw} \in [1, q-1]$, i.e. $SK^{pw} = \{SK_1^{pw}, SK_2^{pw}, \ldots, SK_N^{pw}\}$.

Step - 5 : The parameters $Y_{i,j}$s are either publicly declared by the TD or transferred to each SC_i through secure channel for the generation of secret keys of the SC_j $(SC_j \leq SC_i\ ,\ 1 \leq j \leq n)$ as described in Section 3.1.

Step - 6 : The TD transfers the whole secret key set SK and the shared secret key set SK^{pw} to the data owner. The TD also transfers shared secret keys SK_v^{pw}s to the users through secure channel.

Step - 7 : The DO creates two access control matrices. The first access control matrix (Table - 1) describes relationship between users (U_v) and security classes (SC_i). The second access control matrix (Table - 2) describes relationship between the security classes (SC_i) and the data sets (DS_i). As shown in the matrices, the parameter '1' defines existence of relationship while the parameter '0' defines non-existence of relationship between the entities.

Step - 8 : The DO encrypts the data of each DS_i with the corresponding key SK_i of the security class SC_i and sends the encrypted data to the CS.

Table 1. U_v vs. SC_i

	SC_1	SC_2	SC_i	SC_n
U_1	1	1	0	0
U_2	0	0	1	1
U_v	1	1	1	0
U_N	1	0	0	1

Table 2. SC_i vs. DS_i

	DS_1	DS_2	DS_i	DS_n
SC_1	1	1	0	0
SC_2	0	0	1	1
SC_i	1	1	0	0
SC_n	0	0	1	1

The key store of DO, U_v, SC_i and CS are shown in Figure - 3.

4.2.2 Data Access Mechanism of the Users

The following steps are followed when a user tries to access data of the data owner -

Step - 1 : When a U_v tries to access data of DS_i, U_v sends request to the DO for the data as
$U_i \rightarrow$ DO : $E_{SK_v^{pw}}[U_v, RD, RI]$
where the information $[U_v, RD, RI]$ is encrypted with the shared key SK_v^{pw}. The parameter request index (RI) is incremented each time for next requested data (RD) of U_v to avert replay attack.

Step - 2 : The DO decrypts the request sent by the U_v by the shared key SK_v^{pw} and checks the validity of the user as well as the request. The DO checks in which SC_i the U_v belongs from the access control matrix (Table - 1) and whether the corresponding SC_i has access to the DS_i from another access control matrix (Table - 2) where the requested data exists.

Step - 3 : After the DO satisfies with the request of U_v, the DO fetches the requested encrypted data ($E_{SK_i}[RD]$) from the CS and the DO over-encrypts the requested data with the shared key SK_v^{pw} of that user as
DO $\rightarrow U_v$: $E_{SK_v^{pw}}[E_{SK_i}[RD]]$.

Step - 4 : The user U_v after receiving the data, decrypts it with the shared key SK_v^{pw} and susequently decrypts the data with the secret key SK_i. Now, U_v gets the requested data (RD).

Any revoked user may retain the SK_i of the security class and she can sniff or eavesdrop the updated data destined for the U_v. The over-encryption helps to protect the updated data from the revoked users and is also useful when any end user switches from one security class to another security class, e.g. any member from security class SC_5 switches to security class SC_8 as shown in Figure 2.

The U_vs may also check the integrity of the received data using the schemes described in [21,22,23].

5 Performance Analysis

The computational and storage overheads of the proposed scheme are due to secret keys generation, number of encryption as well as decryption of the data

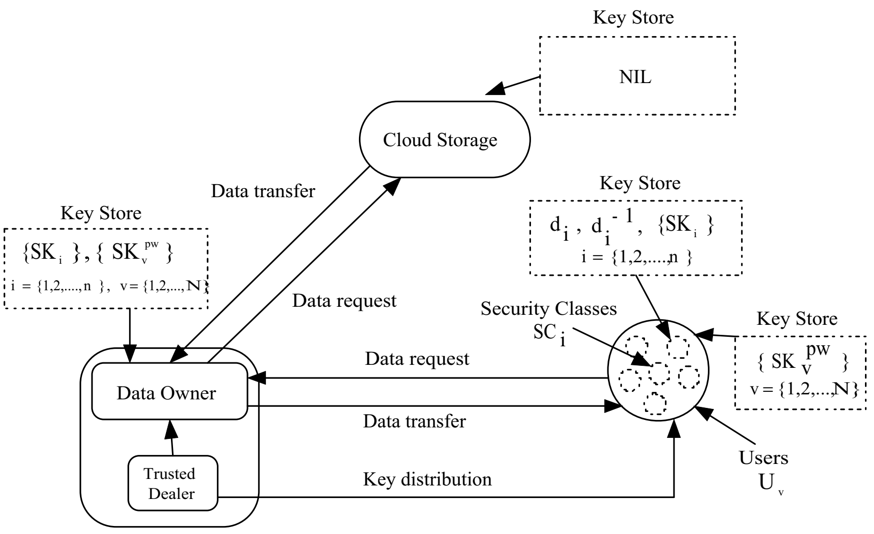


Fig. 3. Key stores of the DO, CS and SC_i

Table 3. Notations

T_{INV}	Time to execute modular inversion
T_{EC_MUL}	Time to execute ECC modular multiplication
T_{HASH}	Time to execute hash function
v_i	The number of successor security classes of a security class

requested by the end users to be performed by the data owner, the cloud storage and the end users. We consider notations for performance analysis in Table - 3.

5.1 Computational Overhead

As discussed in Section - 3, the computational overhead in the DO is due to generation of public keys P_is, the secret keys SK_is for n number of security classes and the parameters $Y_{i,j}$s which will be transferred to the security classes for generation of secret keys by the security classes. Then, the computational overhead in the DO -

$$T_{DO} = n.T_{EC\_MUL} + n.T_{EC\_MUL} + n.T_{HASH} + \sum_{i=1}^{n}(v_i + 1).T_{EC\_MUL}.$$

The first term $n.T_{EC\_MUL}$ of the above expression is due to generation of the P_is of the n security classes.

The second term $n.T_{EC\_MUL}$ is due to generation of n number of Z_i of the security classes (Section 3.1, Step - 3).

The third term $n.T_{HASH}$ is due to converting the ECC points Z_i to security keys (SK_i) of the security classes.

The fourth term $\sum_{i=1}^{n}(v_i + 1).T_{EC\_MUL}$ signifies computational overhead due to generation of parameters $Y_{i,j}$s (Section 3.1, Step - 4) as each security class requires $(v_i + 1).T_{EC\_MUL}$ operations.

$$\begin{aligned}\therefore T_{DO} &= 2.n.T_{EC\_MUL} + n.T_{HASH} + \textstyle\sum_{i=1}^{n}(v_i + 1).T_{EC\_MUL}.\\ &= (\textstyle\sum_{i=1}^{n}v_i + 3.n).T_{EC\_MUL} + n.T_{HASH}.\end{aligned}$$

Similarly, the total computational overhead of the n security classes is

$$T_{SC} = n.T_{INV} + \textstyle\sum_{i=1}^{n}(v_i + 1).T_{EC\_MUL} + \sum_{i=1}^{n}(v_i + 1).T_{HASH}.$$

The first term of the above expression $n.T_{INV}$ signifies the computational overhead due to inversion (T_{INV}) of n number of private keys $d_i s$, i.e. d_i^{-1} of n security classes (Section 3.2, Step - 2).

The second term of the above expression $\sum_{i=1}^{n}(v_i + 1).T_{EC\_MUL}$ indicates evaluation of the ECC points Z_is by the security classes (Section 3.2, Step - 3). The third term $\sum_{i=1}^{n}(v_i + 1).T_{HASH}$ indicates computation of the security keys SK_is by the security classes with the one way hash function H.

$$\therefore T_{SC} = n.T_{INV} + \textstyle\sum_{i=1}^{n}(v_i + 1).(T_{EC\_MUL} + T_{HASH}).$$

The computational overhead in a system of a user, T_U as a member of a security class (SC_i) -

$$T_U = T_{INV} + (v_i + 1).(T_{EC\_MUL} + T_{HASH}).$$

5.2 Storage Overhead

The DO will have to store public keys of the n security classes, secret keys (SK_i) of the n security classes, shared secret keys (SK_i^{pw}) of the N users. So, the storage overhead of the DO

$S_{DO} = [2.163.n + 163.n + N.163]$ bits

The first term is for storing n number of P_is of the SC_i (x & y coordinate of the ECC curve), the second term is for storing n number of SK_i and the third term is for storing SK_i^{pw} of the N users.

Any user as a member of a security class will have to store

$S_U = [(v_i + 1).2.163 + 2.163 + 163]$ bits $= [326.v_i + 815]$ bits.

The first term is for storing $(v_i + 1)$ number of $Y_{i,j}$s (x & y coordinate), the second term is for storing the private key d_i and d_i^{-1}, the third term is for storing the shared private key SK_i^{pw} used for over-encryption.

5.3 Security Analysis

The security of our scheme depends on ECDLP (Definition - 1) and we assume that the key stores of DO and U_v remain secured throughout the operation of the scheme.

The CS of our scheme cannot be compromised by any active or passive adversary as the large volume of data which are exported to the CS by the DO remains encrypted throughout the operation of our scheme. As the CS contains only encrypted data, she herself cannot decrypt or modify any data as in our proposed scheme, the CS cannot have any secret key to decrypt any data.

The DO always fetches data in encrypted form from the CS which is requested by the U_v. The requested data is never decrypted by the DO except updation of the data and over-encryption is done over the requested encrypted data to send it to the U_v. So, any active or passive adversary cannot compromise the DO.

The U_v always receive any data in encrypted form. Any revoked user from the same security class cannot eavesdrop any updated data destined for the U_v as the data for each U_v is over-encrypted by each U_v's unique shared secret key by the DO.

6 Comparison with Other Scheme

We compare our scheme with Wang et al. [7]. The computational overhead of Wang et al. [7] scheme comes from two aspects - key derivation using hash functions and over-encryption using one time key pad. This scheme [7] has not considered overhead due to decryption time and we have excluded the overhead due to execution time required for encryption and decryption of both the schemes. Every data block needs different key in Wang et al. [7] scheme. In this scheme, size of data block is considered as 4 KB. The height of the key hierarchy is considered as 42 and generation of one encryption key requires one hash operation. Suppose, an end user U_v wants to access 1 GB = 250,000 blocks of data from the server and 1 GB data consists of 250 chunks considering average number of one chunk consists of 1000 blocks where each chunk needs one key (chunk key) in its hierarchy. The generation of other block keys from those keys (250 chunk keys) require to conduct 8000 hash operations and for updated data blocks need 10750 hash operations to access 1 GB data. So, the data owner requires to perform total 18,750 hash operations to generate the required keys. For an end user, total 250 x 2000 = 50,0000 hash operations are required to access 1 GB data with updated data blocks. For over-encryption, the cloud storage server and the end user will have to perform 4.8 x 10^9 machine cycles considering all the systems (the data owner, the cloud storage server and the end users) use 1-GHz CPU system. It has been assumed that one hash computation requires 20 machine cycles to process one byte then 4.8 x10^9 machine cycles is equivalent to 24 x 10^7 hash operations. Then, if one end user wants to access 1 GB data, the data owner will have to compute total 18,750 hash operations, the cloud storage server will have to compute 24 x 10^7 hash operations and the end user will have to compute [50,0000 + 24 x 10^7] hash operations. This amounts to approx. 11 seconds for an end user to access 1 GB data.

For our proposed scheme, the computational complexity depends on over how many data sets (DS_i) the requested data has been distributed. If 1 GB requested data lies on one data set, then the DO will have to generate one secret key (Z_i = $H(Z_i)$), one $Y_{i,j}$ and one $Y_{i,i}$, i.e. total 3 numbers of T_{EC-MUL} and one hash operation will have to be performed. Similarly, if data is spread over more than one DS_i, the total number of operations to be performed is [v_i.(three T_{EC_MUL} operations and one hash operation)]. For accessing 1 GB data, the U_v will have to perform one T_{INV}, one T_{EC_MUL} and one hash operation, if the data is spread over same DS_i. Again, for accessing the same amount of data, the U_v will have to perform [v_i.(one T_{INV}, one T_{EC_MUL} and one hash operation)], if the data is spread over v_i number of DS_i. For over-encryption, no extra overheads are added to the DO, the CS and the U_v as keys are not required to generate for that purpose.

So, it is evident that our scheme requires less computational and storage overhead than Wang et al. [7] scheme in case any user wants to access 1 GB data. Wang et al.'s scheme also requires large number of computations if large number of end users access the server at the same time while for our proposed scheme computational overhead arises if the data is spread over number of data sets. The Table - 4 shows comparison of computations which is required for accessing 1 GB data from the cloud storage of Wang et al. [7] scheme with our proposed scheme.

Table 4. Comparison of Computational overhead for accessing 1 GB data

	Wang at el. [7]	**Our Proposed Scheme**
Data Owner	18,750 hash	v_i.(3.T_{EC_MUL} + one hash)
Cloud Storage	24 x 10^7 hash	Nil
End User	[50,0000 + 24 x 10^7] hash	v_i.(1.T_{INV} + 1.T_{EC_MUL} + one hash)

Hasegawa et al. [24] implemented an Elliptic Curve Cryptosystem on a 16-bit microcomputer M16C (10MHz) designed by Mitsubishi Electric Corporation suitable for using in embeddeed systems and the processing times for scalar multiplication of a randomly given point, a modulo inversion of a given 160-bit data and SHA-1 are shown in the Table - 5.

Table 5. Processing time for various ECC modules [24]

Module	**Processing Time**
Elliptic Scalar Multiplication (Random Point)	480 msec
Inversion Modulo 160-bit Integer	130 msec
SHA-1 (time for processing one block)	2 msec

Now, we utilize the processing times of the Table - 5 and assume that the requested 1 GB data is spread over in maximum 5 number of DS_i. Then, any end user of our proposed scheme will take 3060 msec for accessing the data and the time taken for any end user in Wang et al. [7] scheme will be [48.1 x 10^7] msec for accessing the same data. So, our scheme is suitable for end users to use resource constrained wireless mobile devices as the scheme requires less computational and storage overhead to access data.

7 Conclusion

In our proposed scheme, it has been shown that the data requested by the users is only served by the data owner which the data owner already sent it in encrypted form to the cloud storage. All the required secret keys are generated and distributed by the trusted dealer. The cloud storage only stores huge volume of data and serves data to the data owner when requires. The encryption and over-encryption of the data are done by the data owner only but it may increase slight bottleneck in the data owner. It has been shown that the incorporation of the ECC based key management scheme in user hierarchy enhances the efficiency of the proposed cloud storage scheme. Our scheme is also efficient in terms of pay-per-use pricing policy as the cloud storage incurs no burden and the end users of our scheme may use wireless mobile devices. The adversaries cannot acquire any information from the data owner, the cloud storage and the end users.

References

1. CSA (Cloud Security Alliance): Security Guidance for Critical Areas of Focus in Cloud Computing V2.1, http://www.cloudsecurityalliance.org/guidance (December 2009)
2. Mell, P., Grance, T.: The NIST Definition of Cloud Computing Version 15. Information Technology Laboratory, NIST (National Institute of Standards and Technology) (October 2009), http://csrc.nist.gov/groups/SNS/cloud-computing
3. Kamara, S., Lauter, K.: Cryptographic Cloud Storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshop. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
4. Vimercati, S.D.C.D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: management of access control evolution on outsourced data. In: International Conference on Very Large Databases, September 23-28, pp. 123–134 (2007)
5. Vimercati, S.D.C.D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: A data outsourcing architecture combining cryptography and access control. In: ACM Workshop on Computer Security Architecture, November 02, pp. 63–69 (2007)
6. Damiani, E., Vimercati, S.D.C.D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: An Experimental Evaluation of Multi-Key Strategies for Data Outsourcing. In: New Approaches for Security, Privacy and Trust in Complex Environments. IFIP International Federation for Information Processing, vol. 232, pp. 385–396. Springer, Heidelberg (2007)
7. Wang, W., Li, Z., Owens, R., Bhargava, B.: Secure and efficient access to outsourced data. In: ACM workshop on Cloud Computing Security, pp. 55–66 (2009)
8. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
9. SEC 1: Elliptic Curve Cryptography, Standards for Efficient Cryptography 1 (SEC1), Working Draft, Version 1.9, (August 22, 2008)
10. Vanstone, S.A.: Elliptic curve cryptosystem - The Answer to Strong, Fast Publickey Cryptography for Securing Constrained Environments. Information Security Technical Report 12(2), 78–87 (1997)

11. Nikooghadam, M., Zakerolhosseini, A., Moghaddam, M.E.: Efficient utilization of elliptic curve cryptosystem for hierarchical access control. The Journal of Systems and Software 83(10), 1917–1929 (2010)
12. Atallah, M.J., Blanton, M., Fazio, N., Frikken, K.B.: Dynamic and ecient key management for access hierarchies. ACM Trans. Inf. Syst. Secur. 12(3), 1–43 (2009)
13. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: USENIX Conference on File and Storage Technologies, pp. 29–42 (2003)
14. Akl, S.G., Taylor, P.D.: Cryptographic solution to a multilevel security problem. In: Proceeding Advances in Cryptology, pp. 237–249 (1982)
15. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Transaction on Computer Systems 1(3), 239–248 (1983)
16. Chang, C.C., Hwang, R.J., Wu, T.C.: Cryptographic key assignment scheme for access control in a hierarchy. Information Systems 17(3), 243–247 (1992)
17. Shen, V.R.L., Chen, T.: A novel key management scheme based on discrete logarithms and polynomial interpolations. Computers & Security 21(2), 164–171 (2002)
18. Chang, C.C., Lin, I.C., Tsai, H.M., Wang, H.H.: A key assignment scheme for controlling access in partially ordered user hierarchies. In: 18th IEEE International Conference on Advanced Information Networking and Applications (AINA 2004), Fukuoka, Japan, vol. 2, pp. 376–379 (March 2004)
19. Jeng, F.G., Wang, C.M.: An efficient key-management scheme for hierarchical access control based on elliptic curve cryptosystem. The Journal of Systems and Software, 1161–1167 (2006)
20. Chung, Y.F., Lee, H.H., Lai, F., Chen, T.S.: Access control in user hierarchy based on elliptic curve cryptosystem. Information Sciences 178, 230–243 (2008)
21. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: International Conference on Very large Data Bases (VLDB 2007), pp. 782–793. ACM (2007)
22. Goodrich, M.T., Papamanthou, C., Tamassia, R., Triandopoulos, N.: Athos: Efficient Authentication of Outsourced File Systems. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 80–96. Springer, Heidelberg (2008)
23. Bowers, K.D., Juels, A., Oprea, A.: HAIL: A High-availability and Integrity Layer for Cloud Storage. In: 16th ACM Conference on Computer and Communications Security, pp. 187–198. ACM (2009)
24. Hasegawa, T., Nakajima, J., Matsui, M.: A Practical Implementation of Elliptic Curve Cryptosystems over GF(p) on a 16-Bit Microcomputer. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 182–194. Springer, Heidelberg (1998)

Reversible Image Watermarking through Coordinate Logic Operation Based Prediction

Ruchira Naskar and Rajat Subhra Chakraborty

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur–721302, India
{ruchira,rschakraborty}@cse.iitkgp.ernet.in

Abstract. Reversible digital watermarking techniques enable the recovery of the original "cover image" from an watermarked image in a distortion–free way. Reversible watermarking techniques find application in military and medical imagery, where integrity of the cover image is of utmost importance. In this paper we propose a reversible digital image watermarking algorithm which predicts a pixel grayscale value exploiting its correlation with its neighboring pixels, using *coordinate logic operations*, and embeds watermark bits into the prediction errors. We have compared the performance of our scheme with other reversible watermarking schemes. Results indicate that the cover image distortion produced by the proposed algorithm is lower than other state–of–the–art reversible watermarking algorithms.

Keywords: Coordinate logic operations, Correlation, Digital watermarking, Pixel prediction, Reversible data hiding.

1 Introduction

Digital watermarking is the act of hiding information in multimedia data (images, audio or video), for the purposes of content protection or authentication [1]. In digital image watermarking, the secret information (usually in the form of a bitstream), the *watermark*, is embedded into an image (*cover image*), in such a way, that distortion of the cover image due to watermarking is almost negligible perceptually. *Fragile watermarking* algorithms are used for authentication of the cover image. A fragile watermark is destroyed, even in case of minimal modification of the cover image. In fragile watermarking algorithms, the watermark is generally a secure keyed hash of the entire cover image. Thus if an intruder modifies even a single bit of the cover image, the extracted watermark does not match the hash of the modified cover image, and the authentication fails at the receiver side. *Reversible watermarking algorithms* belong to the class of fragile watermarking algorithms. Additionally, in reversible watermarking, the image restored after the watermark extraction, is identical to the original cover image, pixel by pixel, bit by bit. Reversible watermarking finds widespread use in domains dealing with highly sensitive data such as military and medical imagery, where distortion–free recovery of the original image after watermark extraction is of utmost importance.

In general reversible watermarking of digital images is carried out by exploiting the high spatial correlation among neighboring pixels. A feature of the cover image is selected, which is modified to embed the watermark bits. For example, the grayscale values of pixels, the difference of adjacent pixels' grayscale values, the quantization or interpolation errors (after the pixels are quantized or interpolated respectively), are some of the features selected by various authors.

In this paper we have introduced a reversible watermarking algorithm for grayscale images. Our algorithm exploits spatial correlation among neighboring pixels of an image to predict the grayscale values of the pixels. We have used *coordinate logic operations* on a set of pixels in close proximity to predict the value of their common neighboring pixel. Difference of pixel values with their predicted values are then modified for embedding the watermark. We have applied our technique on a suite of standard benchmark images, and found that our technique achieves high embedding capacity at low distortion, compared to other state–of–the–art reversible watermarking techniques.

The rest of the paper is organized as follows. In Section 2, we provide an overview of coordinate logic operations and their application. In Section 3, we describe our proposed watermarking algorithm in detail. Section 4 provides the experimental results and comparison of our algorithm with other state-of-art reversible image watermarking algorithms. We point to future research directions and conclude in Section 5.

2 Coordinate Logic Operations

Coordinate logic operations. (CLOs) [2] are logic operations such as *AND, OR, XOR* etc. applied to corresponding binary bits at the same position in two binary bit sequences. Let a and b be two unsigned n–bit integers such that their binary representations are $a_1, a_2, \cdots, a_n$ and $b_1, b_2, \cdots, b_n$ respectively. $a_i, b_i \in \{0, 1\}\ \ \forall i \in [1..n]$. A CLO ($\bullet$) on a and b is a "bitwise operation" represented as:

$$c = a \bullet b \tag{1}$$

where c is another unsigned n–bit integer whose binary representation is $c_1, c_2, \cdots, c_n$. $c_i \in \{0, 1\}\ \ \forall i \in [1..n]$. Here $\bullet$ is a CLO such as coordinate logic AND ($CAND$), coordinate logic OR (COR), coordinate logic XOR ($CXOR$). Then,

$$c_i = a_i \circ b_i, \ \ \forall i \in [1..n]$$

where $\circ$ is a normal logic operation such as *AND, OR, XOR* respectively.

An example will make the idea clear. Let $a = 11001011_2$ and $b = 01001101_2$ be two unsigned 8–bit integers. Then $a\ CAND\ b = 01001001_2 = 73$. Similarly, $a\ COR\ b = 11001111_2 = 207$.

Application of a CLO to more than 2 unsigned n–bit integers, $A_1, A_2, \cdots, A_m$ ($m > 2$) is represented as:

$$C = \bullet(A_1, A_2, \cdots, A_m) = (A_1 \bullet (A_2 \bullet \cdots (A_{m-1} \bullet A_m))) \tag{2}$$

where each of $A_1 \cdots A_m$ and C is an unsigned n–bit integer. For example, let $A_1 = 11001011_2$, $A_2 = 01001101_2$ and $A_3 = 10001001_2$ be three unsigned 8–bit integers. Here $m = 3$. *CAND* operation applied to A_1, A_2, A_3 results in

$$\begin{aligned}
&CAND\ (A_1, A_2, A_3)\\
&= A_1\ CAND\ (\ A_2\ CAND\ A_3\ )\\
&= 00001001_2.\\
&= 9
\end{aligned}$$

Similarly, $COR\ (A_1, A_2, A_3) = 11001111_2 = 207$.

The following theorem [2] states those properties of *CAND* and *COR* operations, which are useful for pixel prediction.

Theorem 1. Let $C = CAND\ (A_1, A_2, \cdots, A_m)$ and $D = COR\ (A_1, A_2, \cdots, A_m)$. Then,

$$\begin{array}{l} 0 \le C \le \min(A_1, A_2, \cdots, A_m), \\ \max(A_1, A_2, \cdots, A_m) \le D \le (2^n - 1), \end{array} \tag{3}$$

where each of $A_1, A_2, \cdots, A_m, C$ and D are unsigned n–bit integers.

The above property can be used to predict the value of a pixel P from the value of four of its neighbors N_1, N_2, N_3, N_4, by application of a function f defined as follows:

$$\begin{aligned} P &= f(N_1, N_2, N_3, N_4) \\ &= CAND(COR(N_1, N_2, N_3), COR(N_1, N_2, N_4), \\ &\qquad COR(N_1, N_3, N_4), COR(N_2, N_3, N_4)) \end{aligned} \tag{4}$$

Note here that each of $N_1 \cdots N_4$, in an n–bit image, is an unsigned n–bit integer, representing one pixel of the image. Due to the property stated in Theorem 1, the predicted value P is also an unsigned n–bit integer, thus a valid pixel of the image, and gives a considerably good estimate of the original pixel.

3 Proposed Algorithm

The proposed algorithm utilizes usually high spatial correlation among neighboring pixel values in grayscale images. It predicts the grayscale value of a pixel from those of its neighboring pixels and the watermark bits are embedded into the prediction errors. Depending on the order of prediction, the pixels are divided into four classes as follows:

1. The pixels of the cover image whose values remain unchanged during the watermarking process are termed *base pixels.*
2. From the base pixels, the *first set of predicted pixel values* are derived.
3. Further, the *second* and *third sets of predicted pixel values* are derived from the base pixels and first set of predicted pixel values.

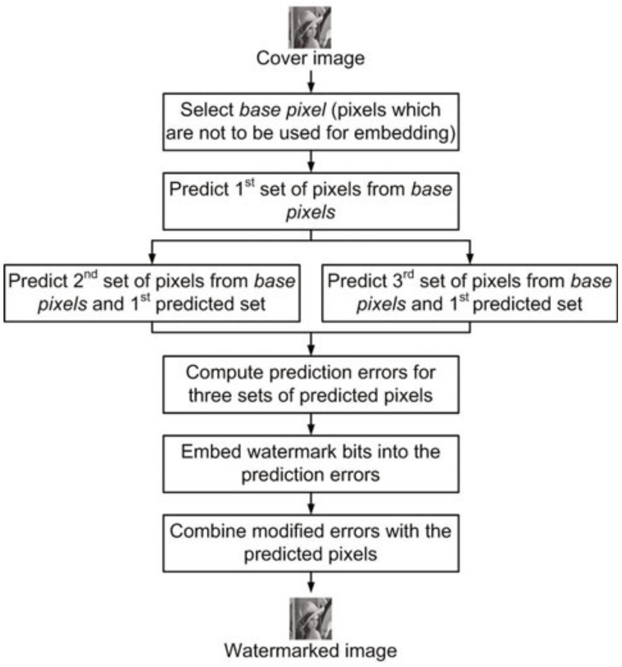

Cover image
Select *base pixel* (pixels which are not to be used for embedding)
Predict 1st set of pixels from *base pixels*
Predict 2nd set of pixels from *base pixels* and 1st predicted set
Predict 3rd set of pixels from *base pixels* and 1st predicted set
Compute prediction errors for three sets of predicted pixels
Embed watermark bits into the prediction errors
Combine modified errors with the predicted pixels
Watermarked image

Fig. 1. Watermark embedding algorithm

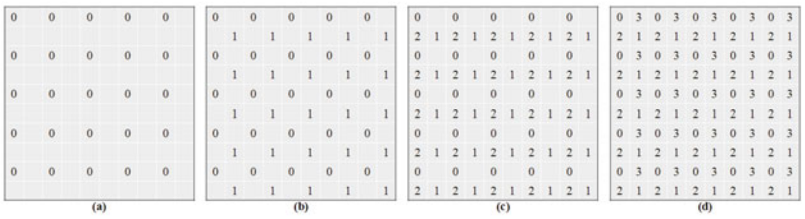


Fig. 2. Locations of (a) base pixels ('0's), (b) predicted first set of pixels ('1's), (c) predicted second set of pixels ('2's), (d) predicted third set of pixels ('3's)

3.1 Watermark Embedding Algorithm

Fig. 1 shows the flowchart of our watermark embedding algorithm. Our watermark embedding algorithm consists of *four* broad steps: (a) selection of base pixels; (b) predicting other pixels from the base pixels; (c) computing the prediction errors, and (d) embedding watermark bits into the errors. We next describe the above steps in detail.

Selection of Base Pixels. One out of every four pixels in the original cover image is chosen as a *base pixel* in our algorithm, such that they are uniformly distributed throughout the image. The positions of the base pixels we selected in an image, are marked with '0's in Fig. 2.

Predicting Three Sets of Pixels. The first set of pixels (marked with '1's in Fig. 2) are predicted from the base pixels whereas the second and third sets of pixels (marked with '2's and '3's respectively in Fig. 2) are predicted from the base pixels as well as the predicted first set of pixels. All predictions are done utilizing CLOs according to the definition of the function f (Definition 1).

Predicted value of each first–set pixel depends on the four base pixels surrounding it on its four corners. Therefore those four base pixels constitute the set of four neighbors $\{N_1 \cdots N_4\}$ for a first–set pixel. The prediction formula for a first–set pixel $p(i,j)$ is given by:

$$\xi(p(i,j)) = f(p(i-1,j-1), p(i-1,j+1), p(i+1,j-1), p(i+1,j+1)) \quad (5)$$

where i and j are the row and column numbers of the pixel to be predicted, respectively.

Next the second set of pixels are predicted. Each second–set pixel is surrounded on its top and bottom by two base pixels, and on its left and right by two first–set pixels. In this case, $\{N_1 \cdots N_4\} = \{p(i-1,j), \xi(p(i,j-1)), \xi(p(i,j+1)), p(i+1,j)\ \}$ and the prediction formula is:

$$\xi(p(i,j)) = f(p(i-1,j), \xi(p(i,j-1)), \xi(p(i,j+1)), p(i+1,j)) \quad (6)$$

For each third–set pixel prediction, we use the two base pixels located on its left and right, and the two first–set pixels on its top and bottom. Here also, $\{N_1 \cdots N_4\} = \{\xi(p(i-1,j)), p(i,j-1), p(i,j+1), \xi(p(i+1,j))\ \}$. The prediction formula is given by:

$$\xi(p(i,j)) = f(\xi(p(i-1,j)), p(i,j-1), p(i,j+1), \xi(p(i+1,j))) \quad (7)$$

Fig. 2(a)–(d) shows the order of prediction of the three pixel sets.

Computing Prediction Errors. Prediction error is given by the difference between an original pixel value p and its predicted value $\xi(p)$. For each predicted pixel we compute prediction error using the following integer transformation:

$$e = \xi(p) - p \quad (8)$$

Due to high correlation of adjacent pixels, in practice, usually the errors are small integers, close to zero.

Embedding Watermark Bits. Prediction errors which are close to zero are used to embed the watermark bits, leading to achievement of high embedding capacity, since the number of errors close to zero is usually large. To define *closeness to zero*, we adopt an *error threshold* k (≥ 0). Only those pixels with prediction errors $|e| \leq k$ are used for embedding watermark bits. A watermark bit is embedded into an error, by multiplying the error by 2 and adding the watermark bit to

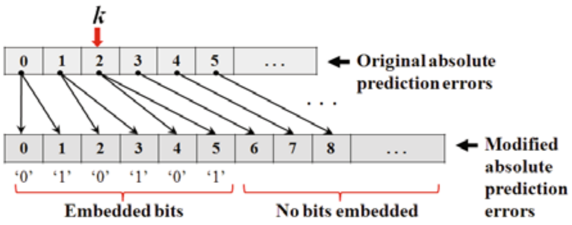


Fig. 3. Modification of absolute values of prediction errors during embedding

the result. For pixels with $|e| > k$, a constant shift of magnitude $(k+1)$ is applied to the absolute prediction error values to avoid overlap with pixels in which watermark bits have been embedded. Procedure 1 (***EMBED_WATERMARK***) is followed to embed watermark bits into the prediction errors. Fig. 3 shows how prediction errors are modified by the procedure ***EMBED_WATERMARK***, for embedding watermark bits.

Combining Modified Errors with Predicted Pixels. Each predicted pixel, combined with its corresponding modified (watermark bit embedded) prediction error, produces a watermarked pixel. We combine a predicted pixel with a modified error using the following integer transformation:

$$p_{wm} = \xi(p) - \phi(e), \tag{9}$$

where p_{wm} is the watermarked pixel and $\phi(e)$ is the corresponding modified (watermark bit embedded) prediction error. Note that, transformation (9) is the reverse of transformation (8).

Procedure 1. ***EMBED_WATERMARK***

```
/* Embed watermark bits into the prediction errors */
Input: Original pixel prediction error ($e$), error threshold ($k$), watermark bit to be
    embedded ($b$)
Output: Modified prediction errors $\phi(e)$
 1: if  $e < 0$  then
 2:    $sgn(e) \leftarrow -1$
 3: else
 4:    $sgn(e) \leftarrow +1$
 5: end if
 6: if  $(|e| > k)$  then
 7:    /* Apply constant shift to the absolute error value */
 8:    $\phi(|e|) \leftarrow |e| + (k+1)$
 9: else
10:    /* $b \in \{0,1\}$ is the next watermark bit to be embedded */
11:    $\phi(|e|) \leftarrow 2 * |e| + b$
12: end if
13: $\phi(e) \leftarrow sgn(e) * \phi(|e|)$
14: return $\phi(e)$
```

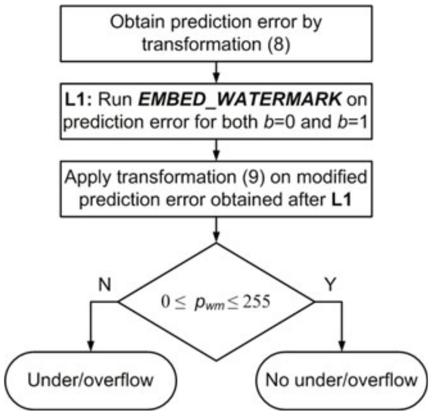

Obtain prediction error by transformation (8)
L1: Run EMBED_WATERMARK on prediction error for both b=0 and b=1
Apply transformation (9) on modified prediction error obtained after L1
0 ≤ p_wm ≤ 255
N
Y
Under/overflow
No under/overflow

Fig. 4. Testing for under/overflow during extraction

Transformation (9) may produce some watermarked pixels falling outside the unsigned 8–bit integer range $[0, 255]$ causing an underflow ($p_{wm} < 0$) or an overflow ($p_{wm} > 255$). Handling of these situations has been presented in Section 3.3. Before that we present our watermark extraction algorithm.

3.2 Watermark Extraction Algorithm

For extraction, we select base pixels in the watermarked image, and predict the first, second and third sets of pixels subsequently, following the same procedure described in Section 3.1. Locations of the base pixels and the three sets of predicted pixels are the same as shown in Fig.2. Next, the following forward transformation (10) is applied to each watermarked non–base pixel p_{wm} and its prediction $\xi(p_{wm})$, to compute the prediction error $\phi(e)$:

$$\phi(e) = \xi(p_{wm}) - p_{wm} \tag{10}$$

Watermark bits are extracted from the prediction errors ($\phi(e)$), and the original errors (e) are restored, following Procedure 2 (***EXTRACT_WATERMARK***). By the term "original errors" we refer to the prediction pixel errors, we achieved originally, before watermark embedding.

We apply the following transformation (11) (reverse of transformation (10)) to each $\{\xi(p_{wm}), e\}$ pair to restore the original cover image:

$$p = \xi(p_{wm}) - e \tag{11}$$

Note that the values of $\xi(p)$ and $\xi(p_{wm})$ are equal since the base pixels are not modified during the process of embedding the watermark.

3.3 Handling of Under/Overflow

If the application of transformation (9) to a particular $\{\xi(p), \phi(e)\}$ pair produces $p_{wm} \notin [0, 255]$, an *underflow* ($p_{wm} < 0$) or *overflow* ($p_{wm} > 255$) is said to have

Procedure 2. ***EXTRACT_WATERMARK***

```
/* Extract watermark bits and restore original prediction errors */
Input: Watermark bit embedded prediction error ($\phi(e)$), error threshold ($k$)
Output: Original prediction errors $e$
 1: if  $\phi(e) < 0$  then
 2:     $sgn(\phi(e)) \leftarrow -1$
 3: else
 4:     $sgn(\phi(e)) \leftarrow +1$
 5: end if
 6: if  $(|\phi(e)| > (2 * k + 1)$  then
 7:     $|e| \leftarrow |\phi(e)| - (k + 1)$
 8: else
 9:     /* $b \in \{0, 1\}$ is the next watermark bit extracted*/
10:     $b = mod(|\phi(e)|, 2)$
11:     $|e| \leftarrow \frac{|\phi(e)| - b}{2}$
12: end if
13: $e \leftarrow sgn(\phi(e)) * |e|$
14: return $e$
```

occurred. We simply do not embed into a prediction error ($\phi(e)$), which may cause such an under/overflow, and move on to the next.

While extraction, we test each prediction error to find out whether it can cause such under/overflow. To perform the test we follow the steps shown in Fig. 4. A prediction error found capable of causing under/overflow during extraction, indicates one of the two possibilities:

1. It was found to be capable of causing under/overflow during embedding, and hence was not used for embedding.
2. Previously it was capable of undergoing embedding without causing an under/overflow, so was used for embedding, but after embedding it has lost its embedding capability.

For error–free extraction, we need to correctly infer which of the above two possible cases has actually occurred. This differentiation is accomplished by the use of a binary bit string, termed as the *location map*. For each occurrence of the first case, we assign a '0' to the location map and for each occurrence of the second case we assign a '1' to the location map. If none of the above cases occurs, the location map is an empty bit string. During extraction, if any prediction error is found to be capable of causing under/overflow, we check the next location map bit. If we encounter a '0' in the location map, we do not use the corresponding prediction error for extraction and keep it unchanged. If we encounter a '1' in the location map, we apply Procedure ***EXTRACT_WATERMARK*** on the corresponding prediction error to extract watermark bits and restore the original prediction error. For our test images, the size of location map required is considerably small. We can further reduce the size of location map by lossless compression, using *runlength encoding* [3] or any other lossless compression algorithm.

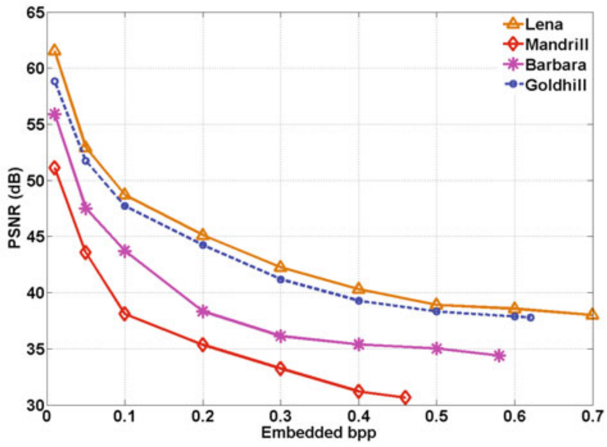


Fig. 5. Plot of Peak Signal–to–Noise Ratio (PSNR) vs. watermark bits embedded per pixel

Table 1. Maximum Embedding Capacity and corresponding PSNR for 512 $\times$ 512 Images, with $k \in [0.10]$

Images	**Maximum Embedding Capacity**		**PSNR (dB)**
	(bpp)	**(bits)**	
Lena	0.7	183499	38.01
Mandrill	0.46	120587	30.65
Barbara	0.58	152043	34.39
Goldhill	0.62	162529	37.81

For insertion of the location map bits we use the LSB positions of the base pixels beginning from the last base pixel. This insertion is done by replacing the LSBs of n_{loc_map} number of base pixel, where n_{loc_map} is the number of location map bits. The end of n_{loc_map} bits is marked with a special end–of–message symbol. Before replacement, the base pixel LSBs are concatenated at the beginning of the watermark and embedded into the predicted pixels errors.

4 Results and Discussion

The proposed algorithm was implemented in MATLAB and tested on 512×512 pixels standard test images: *Lena*, *Mandrill*, *Barbara* and *Goldhill*. We tested our algorithm to investigate its performance with respect to the following properties:

– Maximum embedding capacity achievable
– Distortion of watermarked image as compared to original cover image.

Maximum embedding capacity of an image was evaluated by the number of pure watermark bits (not including the overhead bits) that can be embedded into the

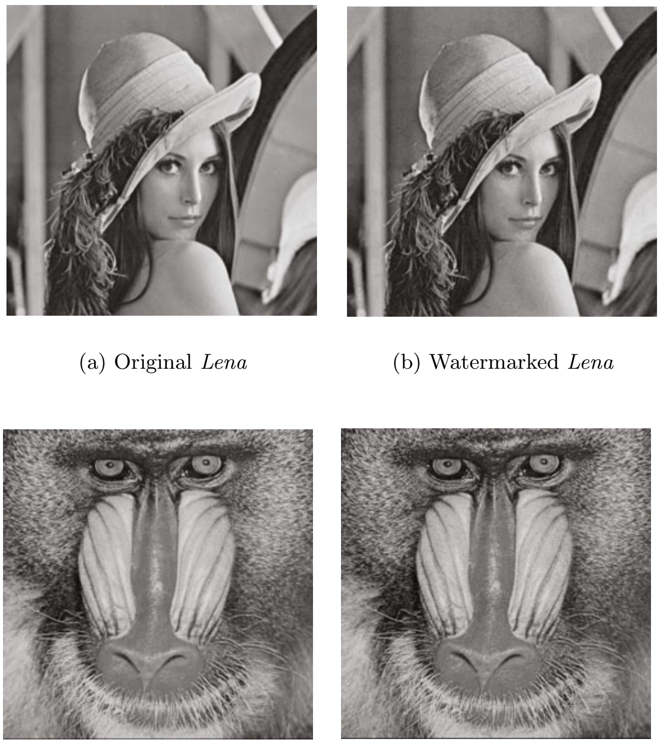


Fig. 6. Proposed reversible watermarking: (*Left*) Original Images; (*Right*) Watermarked Images

entire cover image as well as average number of bits that can be embedded per pixel, measured in units of bpp (*bits–per–pixel*). Distortion of the watermarked image was estimated in terms of *Peak–signal–to–noise–ratio* (PSNR). To calculate the PSNR, first the *mean square error* (MSE) was calculated as:

$$MSE = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\left(X_{org}(i,j) - X_{wm}(i,j)\right)^2}{m \cdot n} \tag{12}$$

where $X_{org}(i,j)$ is the (i,j)–th pixel of the original image, and $X_{wm}(i,j)$ is the (i,j)–th pixel of the watermarked image, and m and n are the dimensions of the image (here each is 512). Then, PSNR was calculated as:

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right)\text{dB} = 10\log_{10}\left(\frac{255^2}{MSE}\right)\text{dB} \tag{13}$$

where MAX_I is the maximum possible pixel value of the image, which is 255 in this case because of the 8–bit grayscale nature of the image.

(a) Original *Barbara*

(b) Watermarked *Barbara*

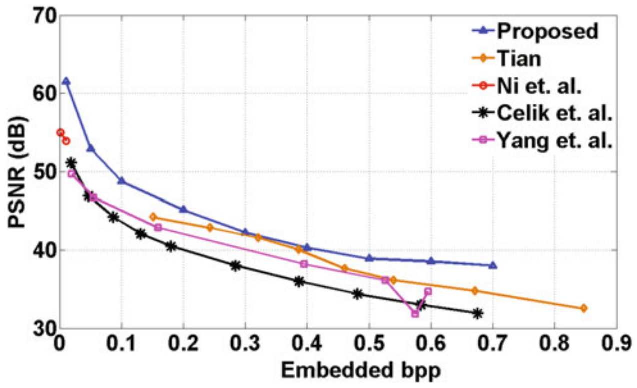
(c) Original *Goldhill*

(d) Watermarked *Goldhill*

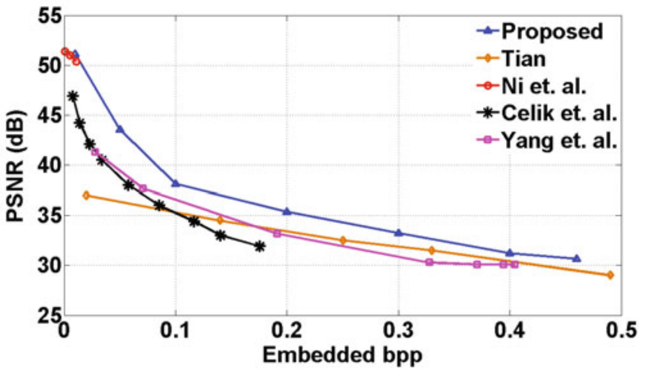
Fig. 7. Proposed reversible watermarking: (*Left*) Original Images; (*Right*) Watermarked Images

We have varied the value of the error threshold parameter k from 0 to 10, in our experiments. Within this range of k, the maximum embedding capacity and corresponding cover image distortion (in form of PSNR), achieved by the proposed algorithm, for each test image has been presented in Table 1. Note that, the embedding capacity can be further increased, by increasing the value of the error threshold. Fig. 5 shows the variation of PSNR (in dB) with embedding capacity (in bpp), for the four test images. The original test images and watermarked images are shown in Figs. 6 and 7. The images in Figs. 6, 7 are watermarked upto their maximum embedding capacities achieved by varying the value of the error threshold parameter k from 0 to 10.

We compared our algorithm in terms of watermark embedding capacity (bpp) and distortion characteristics (PSNR), with other state–of–the–art reversible watermarking algorithms, such as those proposed by Tian [5], Celik et. al. [7], Ni et. al. [10] and Yang et. al. [12]. In these techniques, higher embedding capacity is achieved by applying *multi–layer embedding*, whereas in our algorithm, higher embedding capacity is achieved by increasing the value of error threshold k.

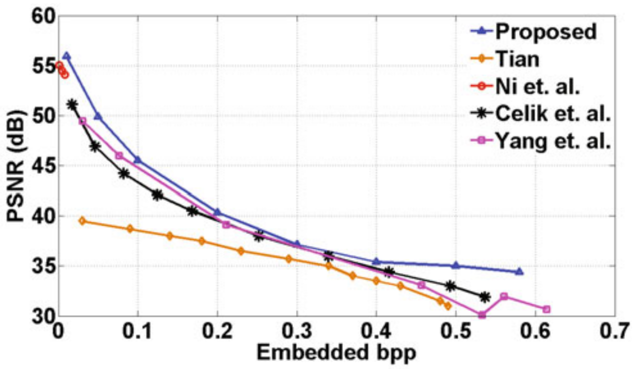


(a) Comparison Results for *Lena*

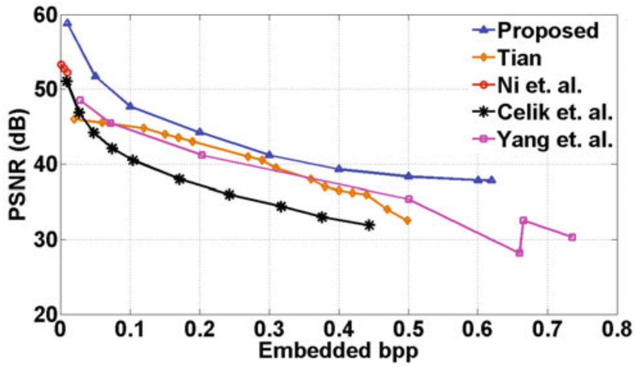


(b) Comparison Results for *Mandrill*

Fig. 8. Capacity vs Distortion Comparison Results for images *Lena* and *Mandrill*



(a) Comparison Results for *Barbara*



(b) Comparison Results for *Goldhill*

Fig. 9. Capacity vs Distortion Comparison Results for images *Barbara* and *Goldhill*

Figs. 8 and 9 show the comparison results, where the proposed algorithm is found to achieve lowest distortion compared to the other algorithms, for equal number of watermark bits embedded.

5 Conclusions

Reversible watermarking is a digital watermarking technique which allows distortion–free recovery of the original content after watermark extraction. In this paper, we have developed a novel reversible watermarking scheme for grayscale images, based on coordinate logic operation based pixel prediction. The main observation that is utilized to embed the watermark is that the absolute values of the prediction errors are small due to high correlation among neighboring image pixels. Watermark embedding capacity achieved by the proposed scheme is high at relatively low distortion of the watermarked cover image, compared to other state–of–the–art reversible watermarking algorithms.

References

1. Cox, I.J., Miller, M.L., Bloom, J.A., Fridrich, J., Kalker, T.: Digital Watermarking and Steganography. Morgan Kaufmann Publishers (2008)
2. Mitra, S., Sicuranza, J.: Nonlinear Image Processing. Academic Press, San Diego (2001)
3. Bhaskaran, V., Konstantinides, K.: Image and Video Compression Standards: Algorithms and Applications, 2nd edn. Kluwer, Norwell (1995)
4. Feng, J.B., Lin, I.C., Tsai, C.S., Chu, Y.P.: Reversible watermarking: current status and key issues. International Journal of Network Security 2(3), 161–171 (2006)
5. Tian, J.: Reversible data embedding using a difference expansion. IEEE Transactions on Circuits Systems and Video Technology 13(8), 890–896 (2003)
6. Tian, J.: Reversible watermarking by difference expansion. In: Proceedings of Workshop on Multimedia and Security, pp. 19–22 (December 2002)
7. Celik, M.U., Sharma, G., Tekalp, A.M., Saber, E.: Reversible data hiding. In: Proceedings of International Conference on Image Processing, pp. III-157–III-160 (September 2002)
8. Celik, M.U., Sharma, G., Tekalp, A.M., Saber, E.: Localized lossless authentication watermark (LAW), International Society for Optical Engineering, California, USA, vol. 5020, pp. 689–698 (January 2003)
9. Fridrich, J., Goljan, M., Du, R.: Lossless data embedding – new paradigm in digital watermarking. EURASIP Journal of Signal Processing 2002(2), 185–196 (2002)
10. Ni, Z., Shi, Y.Q., Ansari, N., Su, W.: Reversible data hiding. IEEE Transactions on Circuits and Systems for Video Technology 16(3), 354–362 (2006)
11. Ni, Z., Shi, Y.Q., Ansari, N., Wei, S.: Reversible data hiding. In: Proceedings of International Symposium on Circuits and Systems, vol. 2, pp. II-912–II-915 (May 2003)
12. Yang, B., Schmucker, M., Funk, W., Busch, C., Sun, S.: Integer DCT-based reversible Watermarking Technique for images using companding technique. In: Proceedings of SPIE, vol. 5306, pp. 405–415 (2004)
13. Plonka, G., Tasche, M.: Integer DCT-II by lifting steps. International Series in Numerical Mathematics 145, 235–252 (2003)

Some Combinatorial Results towards State Recovery Attack on RC4[⋆]

Apurba Das[1], Subhamoy Maitra[1], Goutam Paul[2], and Santanu Sarkar[1]

[1] Applied Statistics Unit, Indian Statistical Institute,
Kolkata 700 108, India
{contactadasbesu,sarkar.santanu.bir}@gmail.com, subho@isical.ac.in
[2] Department of Computer Science and Engineering, Jadavpur University,
Kolkata 700 032, India
goutam.paul@ieee.org

Abstract. A stream cipher has an unobservable internal state that is updated in every step and a keystream output (bit or word) is generated at every state transition. State recovery attack on stream cipher attempts to recover the hidden internal state by observing the keystream. RC4 is a very widely used commercial stream cipher that has a huge internal state. No known state recovery attack on RC4 is feasible in practice and the best so far has a complexity of 2^{241} (Maximov et al., CRYPTO 2008). In this paper, we take a different approach to the problem. RC4 has a secret index j of size one byte. We perform a combinatorial analysis of the complexity of RC4 state recovery under the assumption that the values of j are known for several rounds. This assumption of knowledge of j is reasonable under some attack models, such as fault analysis, cache analysis, side channel attacks etc. Our objective is not to devise an unconditional full state recovery attack on RC4, but to investigate how much information of j leaks how much information of the internal state. In the process, we reveal a nice combinatorial structure of RC4 evolution and establish certain interesting results related to the complexity of state recovery.

Keywords: Cryptanalysis, RC4, State Recovery Attack, Stream Cipher.

1 Introduction

RC4 is one of the most popular stream ciphers with the following structure. It requires an array S of size N (typically, 256), which contains a permutation of the integers $\{0, \ldots, N-1\}$, two indices i, j and the secret key array K. Given a secret key k of l bytes (typically 5 to 32), the array K of size N is such that $K[y] = k[y \bmod l]$ for any y, $0 \leq y \leq N-1$.

[⋆] This paper is based on the M. Tech. (CS) dissertation work of the first author under the supervision of second author at Indian Statistical Institute, Kolkata.

The permutation S is initialized as the identity permutation. Then RC4 proceeds in two phases: the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA uses the secret key to scramble the permutation and the PRGA uses the scrambled permutation to generate the keystream bytes that are bitwise XOR-ed with the plaintext bytes in the sender end (during encryption) and with the ciphertext bytes at the receiver end (during decryption).

Below we describe the KSA and the PRGA briefly. All additions performed are addition modulo N, unless otherwise specified.

```
KSA
Initialization:
        For $i = 0, \ldots, N-1$
                $S[i] = i$;
        $j = 0$;

Scrambling:
        For $i = 0, \ldots, N-1$
                $j = (j + S[i] + K[i])$;
                Swap($S[i], S[j]$);
```

```
PRGA
Initialization:
        $i = j = 0$;

Keystream Generation Loop:
        $i = i + 1$;
        $j = j + S[i]$;
        Swap($S[i]$, $S[j]$);
        $t = S[i] + S[j]$;
        Output $z = S[t]$;
```

RC4 can be completely broken if one can reconstruct the permutation S^G by observing the keystream output bytes. Such attacks are called *state recovery attacks*.

The RC4 state consists of two 8-bit indices i and j and a permutation of 256 possible 8-bit elements. Thus, the size of the state space is $2^8! \times (2^8)^2 \approx 2^{1700}$, making the exhaustive search completely infeasible.

In [1], it has been estimated that this kind of attack would require around 2^{779} complexity. Later in [6], an improved idea has been presented that estimates a complexity of 2^{731}. A much improved result [3] in this area shows that the permutation can be recovered in around 2^{241} complexity, rendering RC4 insecure when the key length is more than 30 bytes. Fortunately, this result does not affect RC4 for the typical secret key size of 5 to 16 bytes.

In this paper, we revisit the problem of state recovery from a combinatorial view point. We model the problem under different assumptions and investigate how the time complexity of performing full state recovery differs from one model to another.

Let S_t be the permutation, z_t be the keystream output byte and i_t, j_t be the indices after t many rounds of RC4 PRGA, $t \geq 1$. We also denote the initial values of these variables before the PRGA starts by S_0, i_0, j_0 (note that z_0 does not exist).

2 Previous Works on State Recovery

The works [1,4] independently discovered for the first time that a branch and bound strategy reduces the complexity for recovering the internal state much below that of exhaustive search.

The basic idea of [1] is as follows. At any point of time, there are four unknowns, namely, $j_r^G, S_r^G[i_r^G], S_r^G[j_r^G], S_r^{-1}[z_r]$. One can simulate the PRGA and guess these unknown values in order to continue when necessary. The recursion steps backward if a contradiction is reached, due to the previously wrong guesses. If some M (out of N) many permutation entries are a-priori known, the complexity is reduced further. For $N = 256$, the complete attack requires a complexity of around 2^{779}. The time complexity of the attack for various values of N and M are provided in Tables D.1 and D.2 in [2, Appendix D.4].

In [4], the cycle structures in RC4 are analyzed in detail and a "tracking" attack is developed that recovers the RC4 state, if a significant fraction of the full cycle of keystream bits is generated. For example, the state of a 5 bit RC4-like cipher can be obtained from a portion of the keystream using 2^{42} steps, while the nominal key-space of the system is 2^{160}.

The work [5] showed that Knudsen's attack [1] requires 2^{220} search complexity if 112 entries of the permutation are known and presents an improvement whereby state recovery with the same complexity requires prior knowledge of only 73 permutation entries in certain cases.

In [6], an improvement over [1] is presented using a tree representation of RC4. At time-step r, the nodes are distributed at $r + 1$ levels. Nodes at level h, $0 < h \leq r$, refer to the set of all possible positions in S_{r-h}^G where z_r can be found. The nodes are connected by the branches which represent the conditions to pass from one node to another. In order to find the internal state, such a tree of general conditions is searched by hill-climbing strategy. This approach reduces the time complexity of the full RC4 state recovery from 2^{779} to 2^{731}.

The best known result for state recovery appears in [3] that shows that the permutation can be recovered in around 2^{241} complexity. This establishes that RC4 is not secure when the key length is more than 30 bytes (240 bits). The basic idea of cryptanalysis in [3] is as follows. Corresponding to a window of $w + 1$ keystream output bytes, one may assume that all the j^G's are known, i.e., $j_r^G, j_{r+1}^G, \ldots, j_{r+w}^G$ are known. Thus w many $S_r^G[i_r^G]$ will be available from $j_{r+1}^G - j_r^G$. Then w many equations of the form $S_r^{G^{-1}}[z_r] = S_r^G[i_r^G] + S_r^G[j_r^G]$ will be found where each equation contains only two unknowns (instead of four unknowns $j^G, S^G[i^G], S^G[j^G], S^{G^{-1}}[z]$ as in [1]). Some precomputation is performed to identify a certain position in the keystream where the internal state is compliant to a specific pattern. A *d-order pattern* is a tuple $A = \{i, j, U, V\}$, where U and V are two vectors from Z_N^d with pairwise distinct elements. At time step r, the internal state is *compliant with* A if $i_r^G = i$, $j_r^G = j$, and d cells of S_r^G with indices from U have corresponding values from V. A pattern A is called w-generative if for any internal state compliant with A, the next w clockings allow to derive w equations of the form $S_r^{G^{-1}}[z_r] = S_r^G[i_r^G] + S_r^G[j_r^G]$, i.e., if consecutive w values of j^G are known. The strategy is to look for d-order w-generative patterns with small d and large w. Whenever the observed keystream indicates such patterns of the internal state, iterative recovery of the unknowns is done and the window w is dynamically expanded. A general time complexity estimate is performed in [3], and simulation results for scaled-down version of

RC4 (i.e. smaller N) are reported. The authors claim that the success rate of the full attack is at least 98%.

A very recent work [7] revisits the method [3] and presents an iterative probabilistic reconstruction and discusses how one can practically approach the complexity of [3].

3 State Recovery with Known j: Theoretical Analysis

Our initial study on state recovery assumes that the index j is known for each round in the RC4 PRGA. If the index j is known at each round of the PRGA, then the value at updated location i of the S array before the current round is known. Therefore, after the swap operation in the current round of PRGA, the value at updated location j in the S array can be determined with probability 1. But that does not ensure that the value at updated location i would be determined after the swap operation, because of the fact that the value at the updated location j may not be known before the swap operation.

Therefore, the only problem here is to deterministically compute the value of the updated location j before the swap operation. For this, we use an auxiliary integer array *guess* of size N initially marked $EMPTY$. We use this array to simulate the hidden permutation S.

Our goal is to gradually fill the $EMPTY$ locations of the array *guess* by the correct values of the permutation S. In the process, we perform swaps in the array *guess* in tandem with the swaps in S so that if the array *guess* becomes completely filled at some round r, then we can obtain $S_r[u]$ directly from the values $guess[u]$ for all u in $[0, N-1]$.

3.1 Without Using the Keystream Bytes

First, we attempt to recover the internal state without using any information about the keystream bytes z_t. Suppose, we observe the evolution of the cipher from round t onwards. At round $t+1$, the value of $S_t[i_{t+1}]$ is known. Therefore, at the end of the $(t+1)$-th round, the value of $S_{t+1}[j_{t+1}]$ will be known deterministically. Then that value will be placed in the array *guess* at location j_{t+1}. Before this update of array *guess*, if the value at location j_{t+1} in *guess* was not $EMPTY$, then that value is to be placed at location i_{t+1} of the array *guess*, otherwise the value at location i_{t+1} of the array *guess* should be updated to $EMPTY$.

If we repeat the above procedure for several rounds, the number of known entries in the array *guess* increases and eventually, at some round $t+m$, we derive $N-1$ entries of S. Since S is a permutation over $\{0, 1, \ldots, N-1\}$, knowledge of the values in any $N-1$ locations reveal the remaining value.

The above discussion is summarized in the form of Algorithm 1.

The complexity of the above algorithm can be expressed in terms of the number m of rounds that needs to be iterated to fill the array *guess*. The following theorem gives the expected value of m.

```
**Input**: $\{(i_{t+r}, j_{t+r}) : r = 0, 1, \ldots, M-1\}$.
**Output**: Permutation array $S_{t+m}$ for some $m \in [0, M-1]$.

1  $numKnown \leftarrow 0$;
2  for $u$ from $0$ to $N-1$ do
3      $guess[u] \leftarrow EMPTY$;
   end
4  $m \leftarrow 0$ ;
5  repeat
6      $guess[i_{t+m+1}] \leftarrow guess[j_{t+m+1}]$;
7      $guess[j_{t+m+1}] \leftarrow j_{t+m+1} - j_{t+m}$;
8      $m \leftarrow m + 1$;
9      $numKnown \leftarrow$Number of non-empty entries in the array $guess$;
   until $numKnown = N-1$ OR $m = M-1$ ;
10 if $numKnown = N-1$ then
11     Fill the remaining single EMPTY location of the array $guess$;
12     for $u$ from $0$ to $N-1$ do
13         $S_{t+m}[u] \leftarrow guess[u]$;
       end
   end
```

Algorithm 1. The algorithm for state recovery when j is known

Theorem 1. *The expected number of rounds of Algorithm 1 to recover S completely is $N \cdot \sum_{k=2}^{N} \frac{1}{k}$.*

Proof. When k entries are filled, the probability that one more entry would be filled in the next step is equal to the probability that the difference in the consecutive j-values (that is a uniformly random number between 0 to $N-1$) computed in Step 7 is distinct from the already present k values. This probability is clearly $p_k = \frac{N-k}{N}$.

Let X_k denote the number of steps required to fill a new entry in *guess*, when k entries of *guess* are filled. So the total number of steps required to fill $N-1$ entries is given by $X = \sum_{k=0}^{N-2} X_k$. Each X_k follows a geometric distribution with probability p_k. Hence, $E(X_k) = \frac{1}{p_k} = \frac{N}{N-k}$. By linearity of expectation,

$$E(X) = \sum_{k=1}^{N-2} E(X_k) = \sum_{k=0}^{N-2} \frac{N}{N-k} = N \cdot \sum_{k=2}^{N} \frac{1}{k}.$$

□

Substituting $N = 256$ in the expression for $E(X)$, we get the theoretical expectation of the number m of rounds required as 1312. If $M < 1312$, then it is expected that we would have a partially recovered state. We experiment by fixing different values of M. For each M, we run RC4 with 100 randomly chosen secret keys and calculate the average number of permutations bytes recovered. The results are presented in Table 1.

Table 1. No. of rounds vs. average no. of bytes recovered for Algorithm 1

Rounds M	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300
#Bytes Recovered	84	140	179	203	220	232	240	245	248	251	253	254	255

3.2 Using the Keystream Bytes

In the above strategy, information of the keystream bytes z_t has not been used. Knowledge of keystream is a standard assumption in *known plaintext attack model* of cryptanalysis. If we use the keystream information, it is likely that complete state recovery would be possible in smaller number of rounds.

```
    Input: $(i_t, j_t), \{(i_{t+r}, j_{t+r}, z_{t+r} : r = 1, \ldots, M-1)\}$.
    Output: Permutation array $S_{t+m}$ for some $m \in [0, M-1]$.
1   $numKnown \leftarrow 0$;
2   for $u$ from $0$ to $N-1$ do
3       $guess[u] \leftarrow EMPTY$;
    end
4   $m \leftarrow 0$;
5   repeat
6       $guess[i_{t+m+1}] \leftarrow guess[j_{t+m+1}]$;
7       $guess[j_{t+m+1}] \leftarrow j_{t+m+1} - j_{t+m}$;
8       if $(guess[i_{t+m+1}] \neq EMPTY)$ AND
        $guess[guess[i_{t+m+1}] + guess[j_{t+m+1}]] = EMPTY$ then
9           $guess[guess[i_{t+m+1}] + guess[j_{t+m+1}]] \leftarrow z_{t+m+1}$;
        end
10      if $guess[i_{t+m+1}] = EMPTY$ AND $z_{t+m+1} = guess[v]$ for some $v$ then
11          $guess[i_{t+m+1}] \leftarrow v - guess[j_{t+m+1}]$;
        end
12      $m \leftarrow m + 1$;
13      $numKnown \leftarrow$ Number of non-empty entries in the array $guess$;
    until $numKnown = N-1$ OR $m = M-1$ ;
14  if $numKnown = N-1$ then
15      Fill the remaining single EMPTY location of the array $guess$;
16      for $u$ from $0$ to $N-1$ do
17          $S_{t+m}[u] \leftarrow guess[u]$;
        end
    end
```

Algorithm 2. The algorithm for state recovery when j, z are known

We can use the keystream bytes to recover the state more efficiently in two ways. Assume that at round t, we know i_t, j_t and at round $t+1$, we know

$i_{t+1}, j_{t+1}, z_{t+1}$. First, we update the contents of the locations i_{t+1} and j_{t+1} of *guess*. Since $z_{t+1} = S_{t+1}[S_{t+1}[i_{t+1}] + S_{t+1}[j_{t+1}]]$, we check whether z_{t+1} is already present at some location v in the array *guess* after the update of the locations i_{t+1} and j_{t+1}. If so, then $S_{t+1}[i_{t+1}]$ is found from $v - S_{t+1}[j_{t+1}]$ and is placed in $guess[i_{t+1}]$. If however, z_{t+1} is not present but $S_{t+1}[i_{t+1}]$ is known, then we can fill z_{t+1} in location $guess[i_{t+1}] + guess_{[}j_{t+1}]$ of *guess*. The detailed procedure is explained in Algorithm 2.

The following result gives an estimate of the number of rounds that need to be iterated for full state recovery.

Theorem 2. *For Algorithm 2, let X_k denote the number of additional rounds required to fill the entire guess array, when k locations are already filled. Then*

$$E(X_k) = 1 + (1 - p_k)\Big(q_k E(X_{k+1}) + (1 - q_k)E(X_k)\Big)$$
$$+p_k\Big(q_{k+1}E(X_{k+2}) + (1 - q_{k+1})E(X_{k+1})\Big),$$

where $p_k = \frac{N-k}{N}$ and $q_k = 2p_k(1 - p_k)$.

Proof. We call that a *success* has occurred in a step of Algorithm 2, if a new entry of *guess* is filled in that step. Note that the Conditions 8 and 10 cannot hold together.

We consider two different cases. When k entries are filled, Step 7 may give a success with probability $p_k = \frac{N-k}{N}$ or a failure with probability $1 - p_k$.

Case I: Failure in Step 7. After a failure in Step 7, which happens with probability $(1 - p_k)$, we would have k entries filled. So, the probability that there would be a new success between Steps 8 and 12 is when either Step 9 gives a success (with probability p_k) and Step 11 gives a failure (with probability $1 - p_k$) or vice versa. Hence, after a failure in Step 7, the probability that there would be one more success between Steps 8 and 12 is given by

$$q_k = p_k(1 - p_k) + (1 - p_k)p_k = 2p_k(1 - p_k),$$

and if there is a success, we would have $k + 1$ entries filled. However, if there is a failure between Steps 8 and 12, which happens with probability $1 - q_k$, then after Step 12, we would have k entries filled. Thus, the contribution of this part to $E(X_k)$ is given by

$$(1 - p_k)\Big(q_k E(X_{k+1}) + (1 - q_k)E(X_k)\Big).$$

Case II: Success in Step 7. After a success in Step 7, we have $k + 1$ entries filled. So, the probability that there would be one more success between Steps 8 and 12 is when either Step 9 gives a success (with probability p_{k+1}) and Step 11 gives a failure (with probability $1 - p_{k+1}$) or vice versa. Hence, after a success in Step 7, the probability that there would be one more success between Steps 8 and 12 is given by

$$q_{k+1} = p_{k+1}(1 - p_{k+1}) + (1 - p_{k+1})p_{k+1} = 2p_{k+1}(1 - p_{k+1}),$$

and if there is a success, we would have $K + 2$ entries filled. However, if there is a failure between Steps 8 and 12, which happens with probability $1 - q_{k+1}$, then after Step 12, we would have $k + 1$ entries filled. Hence, the contribution of this part to $E(X_k)$ is given by

$$p_k \Big(q_{k+1} E(X_{k+2}) + (1 - q_{k+1}) E(X_{k+1}) \Big).$$

In addition to the above two contributions, we need to add 1 to $E(X_k)$, as we have analyzed the situations after one more additional round. So,

$$\begin{aligned} E(X_k) = 1 + (1 - p_k) \Big(q_k E(X_{k+1}) + (1 - q_k) E(X_k) \Big) \\ + p_k \Big(q_{k+1} E(X_{k+2}) + (1 - q_{k+1}) E(X_{k+1}) \Big). \end{aligned}$$

□

Corollary 1. *The expected number of rounds required to completely recover the RC4 state using Algorithm 2 is given by $E(X_0)$, where $E(X_{N-1}) = E(X_N) = 0$.*

Experimental results show that the number m of rounds required to fill the array A using the improved algorithm is around 550, which is close to the theoretical value 531 obtained computing $E[X_0]$ as stated in Corollary 1. Table 2 shows the experimental results generated in the same method as in Section 3.1.

Table 2. No. of rounds vs. average no. of bytes recovered for Algorithm 2

Rounds M	100	150	200	250	300	350	400	450	500	550
#Bytes Recovered	112	163	203	229	245	252	255	255.6	255.9	255.99

4 Heuristics for Further Improvement

The number of rounds (which is equal to the number of known j values) required in Algorithm 2 can be reduced further by applying some heuristics that we describe in this section.

In Algorithms 1 and 2, information of the new entries filled at any round r could not be used in any earlier round $t < r$. We introduce a concept of *backward pass* on the auxiliary array *guess*. Suppose the algorithm begin execution from round t. After each subsequent round r, we revert everything back to the initial round t and in the process use the new entries to check if the array *guess* can be populated further. After we reach round t, we again perform a *forward pass* up to the current round r to further populate the array *guess* as much as possible. The improved strategy is presented in Algorithm 3.

Algorithm 3 uses two subroutines. The subroutine $backtrack(r, t)$ presented in Algorithm 4 performs a backward pass, tracing all state information back from the current round r to a previous round $t < r$. On the other hand, the subroutine $processForward(r, t)$, presented in Algorithm 5 evolves the state information in the forward direction from a past round r to the current round $t > r$. Unlike the previous two algorithms, an additional two dimensional array *acc* is used, whose r-th row contains the triplet (i_r, j_r, z_r).

```
   Input: $(i_t, j_t), \{(i_{t+r}, j_{t+r}, z_{t+r} : r = 1, \ldots, M-1)\}$.
   Output: Permutation array $S_{t+m}$ for some $m \in [0, M-1]$.
1  $numKnown \leftarrow 0$;
2  $m \leftarrow 0$;
3  for $u$ from $0$ to $N-1$ do
4      $guess[u] \leftarrow EMPTY$;
   end
5  $acc[0][0] \leftarrow i_t$;
6  $acc[0][1] \leftarrow j_t$;
7  for $u$ from $1$ to $M-1$ do
8      $acc[u][0] \leftarrow i_{t+u}$;
9      $acc[u][1] \leftarrow j_{t+u}$;
10     $acc[u][2] \leftarrow z_{t+u}$;
   end
11 repeat
12     $i_{t+m+1} \leftarrow acc[t+m+1][0]$;
13     $j_{t+m+1} \leftarrow acc[t+m+1][1]$;
14     $z_{t+m+1} \leftarrow acc[t+m+1][2]$;
15     if $guess[i_{t+m+1}] = EMPTY$ then
16         $guess[i_{t+m+1}] \leftarrow j_{t+m+1} - j_{t+m}$;
       end
17     $backtrack(t+m, t)$;
18     $processForward(t, t+m+1)$;
19     $m \leftarrow m+1$;
20     $numKnown \leftarrow$ Number of non-empty entries in the array $guess$;
   until $numKnown = N-1$ OR $m = M-1$ ;
21 if $numKnown = N-1$ then
22     Fill the remaining single EMPTY location of the array $guess$;
23     for $u$ from $0$ to $N-1$ do
24         $S_{t+m}[u] \leftarrow guess[u]$;
       end
   end
```

Algorithm 3. The algorithm for state recovery with backward and forward passes

```
   Subroutine $backtrack(r, t)$
1  repeat
2      $i_r \leftarrow acc[r][0]$;
3      $j_r \leftarrow acc[r][1]$;
4      $swap(guess[i_r], guess[j_r])$;
5      $r \leftarrow r-1$;
   until $r = t$ ;
```

Algorithm 4. Subroutine *backtrack*

```
Subroutine $processForward(r, t)$
1  repeat
2      $i_r = acc[r][0]$;
3      $j_r = acc[r][1]$;
4      $z_r = acc[r][2]$;
5      $swap(guess[i_r], guess[j_r])$;
6      if $guess[i_r] \neq EMPTY$ then
7          $temp \leftarrow guess[i_r] + guess[j_r]$;
8          if $guess[temp] = EMPTY$ then
9              $guess[temp] \leftarrow z_r$;
           end
       end
10     if $guess[i_r] = EMPTY\ AND\ z_r = guess[v]$ then
11         $guess[i_r] \leftarrow v - guess[j_r]$;
       end
12     $r \leftarrow r + 1$;
   until $r = t$ ;
```

Algorithm 5. Subroutine *processForward*

4.1 Experimental Results

Theoretical analysis of Algorithm 3 is a challenging task. Since the theoretical analysis is yet open, we present some experimental evidences to support the improvements achieved. Experimental result showing the average number of bytes recovered (over 100 random simulations of RC4) against the number of rounds used is shown in Table 3.

Table 3. No. of rounds vs. average no. of bytes recovered for Algorithm 3

Rounds M	100	150	200	250
#Bytes Recovered	146	218	240	255

In Figure 1, we plot the number of $S[i]$'s recovered vs. no. of j's known (on the left) and the ratio of the numbers of $S[i]$'s recovered and j's known (on the right). It is interesting to note that though the number of bytes recovered increases with number of known j's, the relationship between the two is not linear. When a few j's or a lot of j's are known, less number of bytes are recovered, compared to when moderate number (around 128) of j's are known. The reason behind this is as follows. When a few j's are known, the probability that more than one entry would be filled is very low (due to Theorem 2). Also, when many j's are known, most of the entries of $guess$ are already filled, so the probability that a new entry computed is different from the already known ones is very low. Therefore, maximum information gain is achieved in between these two extreme cases. From our experiments, we find the maximum gain corresponding to the case when 116 many selected j values are known. A potential future work would be to guess such 116 j values and then devise a strategy to reconstruct the full state.

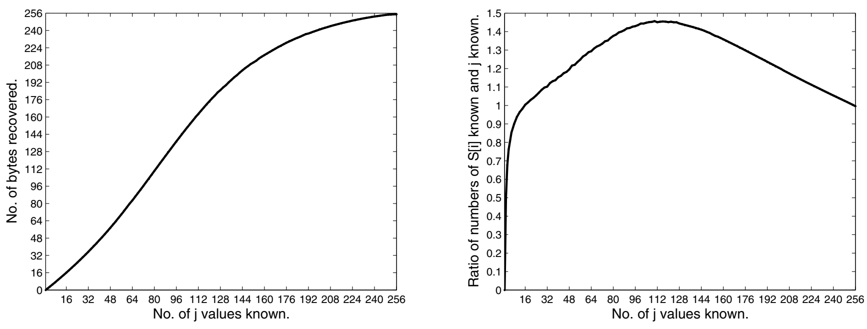


Fig. 1. Relationship between no. of permutation bytes recovered and no. of j's known for Algorithm 3

5 Conclusion

We show how the knowledge of the secret index j leaks information about the internal state of RC4 PRGA. Though our analysis does not immediately lead to a state recovery attack on RC4, it certainly gives insight into the interplay between the state variables and their dependencies. Full state recovery attack on RC4 in practically achievable complexity is still an open problem. Currently the best known state recovery attack requires 2^{241} complexity [3]. We believe our work may be extended further to investigate the possibility of RC4 state recovery in complexity less than 2^{241}.

References

1. Knudsen, L.R., Meier, W., Preneel, B., Rijmen, V., Verdoolaege, S.: Analysis Methods for (Alleged) RC4. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 327–341. Springer, Heidelberg (1998)
2. Mantin, I.: Analysis of the stream cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel (2001)
3. Maximov, A., Khovratovich, D.: New State Recovery Attack on RC4. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, Heidelberg (2008)
4. Mister, S., Tavares, S.E.: Cryptanalysis of RC4-like Ciphers. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 131–143. Springer, Heidelberg (1999)
5. Shiraishi, Y., Ohigashi, T., Morii, M.: An Improved Internal-state Reconstruction Method of a Stream Cipher RC4. In: Hamza, M.H. (ed.) Proceedings of Communication, Network, and Information Security, Track 440-088, New York, USA, December 10-12, pp. 440–488 (2003)
6. Tomasevic, V., Bojanic, S., Nieto-Taladriz, O.: Finding an internal state of RC4 stream cipher. Information Sciences 177, 1715–1727 (2007)
7. Golic, J., Morgari, G.: Iterative Probabilistic Reconstruction of RC4 Internal States. IACR Eprint Server, eprint.iacr.org, number 2008/348 August 8 (2008)

Distributed Signcryption from Pairings

Indivar Gupta and P.K. Saxena

SAG, DRDO, Metcalfe House Complex, Delhi-110054, India
{indivargupta,pksaxena}@sag.drdo.in

Abstract. The distributed signcryption scheme was proposed by Mu and Varadharajan in their paper presented at INDOCRYPT 2000. Since then some more schemes have been proposed for distributed signcryption and extended for group signcryption.

In 2007, Li et al [15] proposed signcryption scheme with key privacy. In this paper, we extend this scheme and propose a scheme for distributed signcryption based on pairings. Further, we extend distributed signcryption protocol to group signcryption. Finally, the security analysis of the protocols has been carried out based on difficulty of Diffie-Hellman problem in Gap Diffie-Hellman groups.

1 Introduction

Encryption is a means to provide 'confidentiality' to data but that is not sufficient to achieve another important security goal namely "authenticity'. Many schemes (protocols) are being proposed so that both of these goals are met when put to use. Traditionally, a two step approach 'sign-then-encrypt', is followed where a message is first digitally signed and then encrypted. In 1997, Zheng [23] introduced the concept of 'signcryption' where he proposed to accomplish the two tasks namely 'digitally signing' and 'encryption' in a single logical step thus reducing computational cost as well as communication overheads. Since then, several new efficient schemes for signcryption have been proposed.

Though signcryption can be used for communication between two individuals or parties, it is more meaningful for communication among many parties /groups/ organizations, where people can be authorized to send and receive message in a hierarchical or some other manner depending upon categorization of information. Towards this objective, Mu and Varadharajan [19] proposed a scheme in 2000 called "distributed signcryption' (scheme referred as *MVS-DSC* here onwards), where a signcrypted message sent by any party (within or outside the group) can be designcrypted by any member of the receiving group. They have further extended this concept enabling a member of a designated group to signcrypt the message on behalf of the group and send it to the receiving group, where the message can be de-signcrypted by any member of that group. This modified group-to-group scheme of Mu and Varadharajan is called 'group signcryption' (referred as *MVS-GSC* here onwards). Both these schemes: MVS-DSC & MVS-GSC have certain weaknesses. These scheme become quite cumbersome with increase in the size of the designated group as the computational complexity and communication overheads increase considerably. In [13], Kwak and Moon proposed distributed signcryption scheme (henceforth referred as KMS-DSC) and its extension to a group signcryption (referred as KMS-GSC henceforth). They also discuss 'join protocol' which allows new members to join the group during initialization. In this protocol, members could choose their own secret key instead of the group manager. But these schemes also had the same weaknesses as those in MVS-DSC & MVS-GSC. In [10], Gupta et al proposed a scheme for distributed signcryption overcoming some such weaknesses (scheme referred as GPSS here onwards). But in all of the schemes mentioned above, authors did not include formal security notions and security proofs. Kwak and Moon [13], however, gave some heuristic arguments for security analysis of their schemes (KMS-DSC & KMS-GSC). Bao, Cao and Quin [2] showed that KMS-DSC actually does not provide sender ID confidentiality since verification step requires sender ID and thus proving that there is no significant advantage of KMS-DSC over MVS-DSC. They also claimed that KMS-GSC does not provide unforgeability and traceability property. Kwak et al proposed a secure extension of KMS-GSC in [14], and gave some informal proofs for unforgeability and coalition-resistance along with heuristic arguments for other notions. None of these schemes have been proved secure through formal security notions.

In this paper, we propose a new scheme for distributed signcryption from pairings and extend our method to group signcryption. We also give formal security model for confidentiality and unforgeability. Here, computational cost and communication overheads are independent of the number of members in the group. Our schemes are extensions of signcryption scheme proposed by Li et al [15] and security of these protocols is based on the difficulty of Diffie-Hellman problem in *Gap Diffie-Hellman Groups* as done in [15] with proofs given in the random oracle model. Since the signcryption scheme proposed by Li et al is efficient and provably secure, we have selected their scheme for extension to distributed signcryption though the concept can be applied on other pairing based signcryption schemes also.

This paper has been organized as follows. In Section 2, we present the mathematical background required for subsequent discussion. In Section 3, we give the signcryption scheme as proposed by Li et al [15] for completeness. In Section 4, we propose our generic method for distributed signcryption scheme along with security notions and then extend it for group signcryption. Our both schemes are presented in Section 5. Security analysis of both schemes has been discussed in Section 6 with comparative results and conclusion in Section 7.

2 Mathematical Background

2.1 Overview of Pairings

Let $\mathfrak{K}$ denote security parameter. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of the same prime order q with $q \geq 2^{\mathfrak{K}}$, where $\mathbb{G}_1$ is written in additive notation with identity $\mathcal{O}$ and $\mathbb{G}_2$ is written in multiplicative notation with identity 1. A function $e: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is called a bilinear map if it satisfies the followings properties:

(a) **Bilinearty:** $e(aP, bQ) = e(P, Q)^{ab}$ $\forall$ $P, Q \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}$.
(b) **Non-Degeneracy:** For any point $P(\neq \mathcal{O}) \in \mathbb{G}_1$, $\exists Q \in \mathbb{G}_1$ such that $e(P, Q) \neq 1$.

For practical applications, we need bilinear maps satisfying the following additional property:

(c) **Computability:** there exists an efficient algorithm to compute $e(P, Q)$ $\forall$ $P, Q \in \mathbb{G}_1$.

The groups $\mathbb{G}_1$, & $\mathbb{G}_2$ are called *bilinear groups* (represented as a pair $(\mathbb{G}_1, \mathbb{G}_2)$) if there exists a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ satisfying (a), (b) and (c) properties.

Bilinear maps are also referred to as 'pairing'. For cryptographic applications, group $\mathbb{G}_1$ & $\mathbb{G}_2$ in 'pairings' are to be chosen carefully. Weil and Tate pairing introduced in [3,5,7,8,11,18] are bilinear maps defined over elliptic curve subgroups. In these cases, the group $\mathbb{G}_1$ is a cyclic elliptic curve subgroup while $\mathbb{G}_2$ is a cyclic subgroup of a multiplicative group of a finite field. Supersingular elliptic curves [5] are popular for the construction of pairing friendly groups but are not so popular anymore.

Modified Weil and Tate pairing provide admissible maps for designing cryptographic systems. More details on pairings, optimization of pairing computation on supersingular elliptic curves and other computational issues can be seen in [3,5,6,8,9,18,20].

2.2 Intractable Problems

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups where $\mathbb{G}_1$ is an additive group with prime order q and $\mathbb{G}_2$ a multiplicative group of the same order. Let P be an arbitrary generator of $\mathbb{G}_1$ and $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map. Then the intractable problems can be formalized in the elliptic curve additive notation as follows:

Computational Diffie-Hellman Problem (CDHP): Given P, aP and bP in $\mathbb{G}_1$ for some $a, b \in \mathbb{Z}_q$, finding $abP \in \mathbb{G}_1$ is known as CDHP in $\mathbb{G}_1$.
Decisional Diffie-Hellman Problem (DDHP) [6] : Given elements P, aP, bP and cP in $\mathbb{G}_1$ for some $a, b, c \in \mathbb{Z}_q$, deciding whether $ab = c \bmod q$ or not is called DDHP problem. If $ab = c$ then the quadruples of the form (P, aP, bP, cP) in $\mathbb{G}_1$ is called the DDH quadruple. Let us denote by $O_{\mathbb{G}_1}^{DDH}$ a decision Diffie-Hellman oracle that solves DDHP i.e. which answers whether a given quadruple is a Diffie-Hellman quadruple or not.
Gap Diffie-Hellman Problem (GDHP) [20]: Given P, aP, bP in $\mathbb{G}_1$ for some $a, b \in \mathbb{Z}_q$, computing $cP = abP$ with the help of $O_{\mathbb{G}_1}^{DDH}$ is called GDHP.

The groups in which a DDHP is easy while the CDHP is believed to be hard are called the *Gap Diffie-Hellman Groups*. More details on Gap Diffie-Hellman Groups can be found in [12,20].

3 Signcryption Scheme with Key Privacy Proposed by Li et al

Libert and Quisquater [16,17] proposed a method for signcryption with key privacy from Gap Diffie-Hellman Groups (henceforth scheme referred as *LQS*). Yang et al [22] analyzed LQS and showed that this scheme can not achieve security claimed. They also proposed an improved signcryption scheme with key privacy (*let us call it YWD-ISC*) and gave security proof. In 2006, Tan [21] pointed out that the YBD-ISC also did not achieve claimed security but he did not suggest any solution to fix the problem. In 2007, Li et al [15] proposed an efficient signcryption scheme (referred as LYWDC-ESC) with key privacy modifying the *YWD-ISC* and removing its weaknesses. The security of this scheme relies upon intractability of the computational Diffie-Hellman problem in Gap Diffie-Hellman groups. The scheme consists of five PPT algorithms: Setup, Key-Gen, Signcryption, DeSigncryption and Verify as described below.

Setup: This algorithm generates system's common public parameters which are shared by both the sender as well as the receiver. Given the security parameters $\mathfrak{K}$ and L, this algorithm outputs cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ (of prime order $q \geq 2^{\mathfrak{K}}$ where each element of $\mathbb{G}_1$ can be represented in L bits), a generator P of $\mathbb{G}_1$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Algorithm also generates hash functions $H_1 : \{0,1\}^{\mu} \times \mathbb{G}_1^3 \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_1^3 \rightarrow \{0,1\}^{\mu+2L}$, where μ denotes the size of the plaintext (i.e. the message space is $M = \{0,1\}^{\mu}$). The common parameters are then $params = \{q, \mathbb{G}_1, \mathbb{G}_2, P, e, H_1, H_2, \mu\}$.

KeyGen: This algorithm takes a random $x_u \in \mathbb{Z}_q^*$ selected by the user 'u' as his private key and outputs the corresponding public key $Y_u = x_u P \in \mathbb{G}_1$. (We will denote the sender and the receiver by $u = S$ and $u = R$ and their key-pairs by (x_S, Y_S) and (x_R, Y_R)) respectively.

Signcryption: Given a plaintext $m \in \{0,1\}^{\mu}$ intended to R, the sender S uses the following procedure:

1. Pick a random $r \in \mathbb{Z}_q^*$ and compute $U = rP \in \mathbb{G}_1$.
2. Compute $V = x_S H_1(m, U, Y_R, rY_R) \in \mathbb{G}_1$.
3. Compute $Z = (m||Y_S||V) \oplus H_2(U, Y_R, rY_R) \in \{0,1\}^{(\mu+2L)}$($\oplus$ denotes the bitwise exclusive OR). The ciphertext is given by $\sigma =< U, Z >\in \mathbb{G}_1 \times \{0,1\}^{\mu+2L}$.

Designcryption: The receiver R follows the the following procedure to recover the plaintext m from the ciphertext $\sigma =< U, Z >$.

1. Compute $D = x_R U$
2. Compute $(m||Y_S||V) = Z \oplus H_2(U, Y_R, D) \in \{0,1\}^{\mu+2L}$ and reject σ if Y_S is not a point on the curve on which $\mathbb{G}_1$ is defined, otherwise go to next step.
3. Compute $H = H_1(m, U, Y_R, rY_R) \in \mathbb{G}_1$ and then check if $e(H, Y_S) = e(V, P)$. If the condition holds, output $(m, (U, Y_R, V, D), Y_S)$; otherwise reject the ciphertext.

Verify: For the message signature pair $(m, (U, Y_R, V, D))$ and a signing key Y_S, the algorithm checks if $e(Y_S, H_1(m, U, Y_R, D)) = e(V, P)$. If the condition holds, it outputs **true**; otherwise it outputs **false**.

It is important to note that **Verify** step need not be executed in the decryption process. This step can be used to verify the signature publicly.

4 Distributed Signcryption and Group Signcryption

As we know, in case of public key cryptography, public keys are used for encryption whereas corresponding private keys are used for decryption. Similar is the case with signcryption. When it comes to distributed / group signcryption, the whole designated group is assigned a common 'public key' called 'group public key' (*gpk*), whereas the corresponding 'secret keys' are different for individual members of the designated group and the 'group manager' ($\mathcal{GM}$). The secret parameters assigned to the group are referred as 'group secret parameters' (*gsp*) and are kept secret by the $\mathcal{GM}$. If the group consists of a single member i.e. the group manager ($\mathcal{GM}$ himself), the group is referred as the *'single-user-group'* (i.e. $gsp = sk$, $gpk = pk$).

In the case of *single-user-group*, there is only one secret key (*skp*= *sk*) with $\mathcal{GM}$ which corresponds to the public key $gpk = pk$. In case the group consists of a $\mathcal{GM}$ and n members, the $\mathcal{GM}$ selects $2 \times (n+1)$ integers from $\mathbb{Z}_q^*$ (q a prime) either randomly or through some function to compute *gpk*, *gsp* and secret keys of individual members. The $\mathcal{GM}$ keeps its *gsp* secret and issues members' secret keys to group members. The *gpk* is common to all group members of the designated group.

In case of distributed signcryption, the sender is a *single person* whereas the receiver is a *designated group*. Group signcryption deals with the communication between two or more groups.

4.1 Definition and Security Models for Distributed Signcryption from Pairings

We first give a generic scheme for distributed signcryption from pairings and then propose formal security notions for *confidentiality* and *unforgeability* of distributed signcryption.

Definition 1. *A distributed signcryption scheme from pairings* (DSC^e_{pairg}) *consists of the following six probabilistic polynomial time (PPT) algorithms.*

Setup: *Based on the input of unary string* $1^{\mathfrak{K}_p}$, *where* $\mathfrak{K}_p$ *is a security parameter, it generates the common system parameters referred as* 'params'.
InzGroup: *It takes* params *and the number of members, whose group is to be initialized, (including group manager) as inputs and outputs the parameters required to initialize the designated group.*
KeyGen: *This algorithm consists of two modules: sender's key generation algorithm* KGA_S *and receiver's key generation algorithm* KGA_R.

1. KGA_S: *This algorithm takes* params *as an input and generates key for sender and returns secret / public key pair* (sk_U, pk_U). *In this process* KGA_R *remains idle.*
2. KGA_R: *Given the input(*params, n*),* KGA_R *generates group public / secret key parameters* (gpk_G, gsp_G) *and member's secret key* (msk_G) *for each member of the receiving designated group* G. *This algorithm calls InzGroup as subroutine during the execution. In this process* KGA_S *remains idle.*

DisSignCrypt: *It is an algorithm which takes* (m, sk_U, gpk_G) *as an input and outputs a ciphertext* $\boldsymbol{c}$, *where* m *is drawn from message space* $\mathcal{M}$, sk_U *is the user secret key and* gpk_G *is the group public key of the receiving group. The ciphertext* $\boldsymbol{c}$ *consists of the signcrypted message* c, *signature* s *on the message* m, *and other data for verification.*
DisDeSignCrypt: *It is an algorithm which takes* ($\boldsymbol{c}$, msk_G) *as an input and outputs either a tuple* (m, s, pk_U) *or the special symbol* $\perp$ *which indicates the ciphertext is invalid.*
Verify: *This algorithm takes* (m, s, pk_U) *as an input and outputs* $`true'$ *or* $`false'$ *depending upon the validity of the signature 's'.*

Definition 2. (Completeness) DSC^{e}_{pairg} *is said to have completeness property if for any message* $m \in M$, $(sk_U, pk_U) \leftarrow \mathbf{KGA_S}(params)$,
$\big((gsp_G, gpk_g), msk^l_G \mid \forall\ l\big) \leftarrow \mathbf{KGA_R}(params, n)$, *we have:*
$\boldsymbol{c} \leftarrow \mathbf{DisSignCrypt}(m, sk_U, gpk_{G_R})$
$(m, s, pk_U) \leftarrow \mathbf{DisDeSignCrypt}(\boldsymbol{c}, msk^l_G)$
and true $\leftarrow \mathbf{Verify}(m, s, pk_U)$ *for each value of* l *where* msk^l_G *is the secrete key of* l^{th} *member of the group* G.

Definition 3. (Confidentiality) *We say that a* DSC^{e}_{pairg} *ensures message confidentiality against chosen-ciphertext attacks i.e. DSC-IND-CCA if no probabilistic polynomial time adversary has a non-negligible advantage in the following game:*

1. *The challenger* $\mathcal{CH}$ *generates two pairs* (sk_U, pk_U) *and* (msk_G, gpk_G). *The* $\mathcal{CH}$ *keeps* sk_U *and* msk_G *secret, while* gpk_G *and* pk_U *are sent to the adversary* $\mathcal{A}$.
2. *In the first stage,* $\mathcal{A}$ *performs a series of queries of the following kinds:*
   DisSignCrypt queries: $\mathcal{A}$ *submits message* (m, gpk_{G_R}) *to* $\mathcal{CH}$ *for the result of DisSignCrypt* (m, sk_U, gpk_R) (gpk_{G_R} *may be different from* gpk_G) *where* $m \in \mathcal{M}$.
   DisDeSignCrypt queries: $\mathcal{A}$ *submits a ciphertext* $\boldsymbol{c}$ *to* $\mathcal{CH}$ *for the result of DisDeSignCrypt* $(\boldsymbol{c}, msk_G)$. *The result contains a tuple of the form* (m, s, pk_S) *in case the de-signcryption is successful and the 's' is valid under the recovered* pk_S. *Otherwise, a symbol* $\perp$ *is returned as a result of DisDeSignCrypt for rejection.*
   These queries can be asked adaptively. After a number of queries, $\mathcal{A}$ *outputs two plaintexts* m_0, $m_1 \in \mathcal{M}$ *of equal length (with the condition that* m_0 *and* m_1 *have not been output of DisDeSignCrypt queries call from* msk_G) *and a chosen private key of the form* sk_S. $\mathcal{CH}$ *flips a coins* $b \in_R \{0, 1\}$ *to compute a distributed signcrypted message* $\boldsymbol{c}^\star$ = *DisSignCrypt* (m_b, sk_S, gpk_G) *of* m_b *with the sender's private key* sk_S *under the attacked group public key* gpk_G. *The ciphertext* $\boldsymbol{c}^\star$ *is sent to* $\mathcal{A}$ *as a challenge.*
3. $\mathcal{A}$ *performs a number of new queries as shown in the first stage with a restriction that* $\mathcal{A}$ *may not ask the DisDeSignCrypt query of the challenged ciphertext* $\boldsymbol{c}^\star$. *At the end of the game,* $\mathcal{A}$ *outputs a bit* b' *and wins if* $b' = b$.

$\mathcal{A}$*'s advantage is defined to be* $Adv^{DSC-IND-CCA}(\mathcal{A}) = 2Pr[b' = b] - 1$.

Definition 4. (Unforgeability) *DSC^{e}_{pairg} is said to be existentially unforgeable against chosen-message attack i.e. DSC-EUF-CMA if no probabilistic polynomial time forger $\mathcal{F}$ has a non-negligible advantage in the following game:*

1. *The challenger $\mathcal{CH}$ generates two pairs (sk_U, pk_U) and (msk_G, gpk_G). The $\mathcal{CH}$ keeps sk_U and msk_G secret, while gpk_G and pk_U are sent to the forger $\mathcal{F}$.*
2. *$\mathcal{F}$ makes a series of DisSignCrypt(m, sk_U, gpk_{G_R}) and DisDeSignCrypt $(\boldsymbol{c}, msk_G)$ adaptive queries exactly in the same way as in step 2 of DSC-IND-CCA game in Definition 3.*
3. *$\mathcal{F}$ produces a ciphertext $\boldsymbol{c}^{\star}$ and a valid key pair $(msk_{G^{\star}_R}, gpk_{G^{\star}_R})$ and wins the game if (i) DisDeSignCrypt $(\boldsymbol{c}^{\star}, msk_{G^{\star}_R})$ returns a tuple $(m^{\star}, s, pk_U)$ where s is a valid signature on the message under the recovered sender's public key pk_U; and (ii) the ciphertext $\boldsymbol{c}^{\star}$ is not the output of any of the DisSignCrypt queries made during the game.*

4.2 Definition and Security Models for Group Signcryption from Pairings

Definition 5. *A group signcryption scheme from pairings (GSC^{e}_{pairg}) consists of the following six probabilistic polynomial time (PPT) algorithms.*

Setup: *Same as in the Definition 1.*
InzGroup: *Same as in the Definition 1.*
KeyGen: *It takes $(params, n)$ as an input and outputs (gpk_G, gsp_G) and member secret key (msk_G) for each member of the designated group using InzGroup as subroutine.*
DisSignCrypt: *It is an algorithm which takes (m, msk_{G_S}, gpk_{G_S}, gpk_{G_R}) as an input and outputs a ciphertext $\boldsymbol{c}$ where gpk_{G_S} and gpk_{G_R} are group public keys of the sending / receiving group and msk_{G_S} is the secret key of any member of the sending group G_S (who is communicating).*
DisDeSignCrypt: *It is an algorithm which takes ($\boldsymbol{c}$, msk_{G_R}) as input and outputs either a tuple $(m, s, Apnd_m(gpk^1_{G_S}))$, or the special symbol $\perp$ for rejection. Here, msk_{G_R} is the secret key of any member of the receiving group G_R and $Apnd_m(gpk^1_{G_S})$ is the information appended with the message which has been derived from one part of the group public key of the sending group.*
Verify: *This algorithm takes $(m, s, Apnd_m(gpk^1_{G_S}))$ as an input and outputs 'true' or '$false$' depending on the validity of the signature 's'.*

Definition 6. (Completeness) *GSC^{e}_{pairg} is said to have completeness property if for any message $m \in \mathcal{M}$,*
$\big((gsp_{G_S}, gpk_{G_S}), msk_{G^{l'}_S} \mid \forall\, l'\big) \overset{Sen.}{\leftarrow} \textbf{KeyGen}(params, n)$,
$\big((gsp_{G_R}, gpk_{G_R}), msk_{G^{l}_R} \mid \forall\, l\big) \overset{Rec.}{\leftarrow} \textbf{KeyGen}(params, n)$, *we have:*
$\boldsymbol{c} \leftarrow \textbf{DisSignCrypt}\ (m, gpk_{G_S}, gpk_{G_R}, msk^{l'}_{G_S})$
$(m, s, Apnd_m(gpk^1_{G_S})) \leftarrow \textbf{DisDeSignCrypt}(\boldsymbol{c}, msk_{G_R})$
& *true* $\leftarrow \textbf{Verify}(m, s, Apnd_m(gpk^1_{G_S}))$

for each value of l and l' where $msk^{l'}_{G_S}$ is the secret key of l^{th} member of the group G_S and $msk^{l}_{G_R}$ is the secrete key of l^{th} member of the group G_R.

Definition 7. (Confidentiality) *We say that GSC^{e}_{pairg} ensures message confidentiality against chosen-ciphertext attacks i.e. GSC-IND-CCA if no probabilistic polynomial time adversary has a non-negligible advantage in the following game:*

1. *A challenger $\mathcal{CH}$ generates two pairs (msk_{G_S}, gpk_{G_S}) and (msk_{G_R}, gpk_{G_R}). The $\mathcal{CH}$ keeps msk_{G_S} and msk_{G_R} secret, while the pair (gpk_{G_S}, gpk_{G_R}) is given to the adversary $\mathcal{A}$.*
2. *In the first stage, $\mathcal{A}$ performs a series of adaptive queries of the following kinds:*
   DisSignCrypt queries: $\mathcal{A}$ submits a message $m \in \mathcal{M}$ and an arbitrary group public key $gpk_{G_{R'}}$ of the receiving group $G_{R'}$ (which may be different from gpk_{G_R}) to $\mathcal{CH}$ for the result of DisSignCrypt$(m, gpk_{G_S}, msk_{G_S}, gpk_{G_{R'}})$.
   DisDeSignCrypt queries: $\mathcal{A}$ submits a ciphertext $\boldsymbol{c}$ to $\mathcal{CH}$ for the result of DisDeSignCrypt $(\boldsymbol{c}, msk_{G_R})$, which contains either a tuple or the symbol $\perp$.
   After a number of queries, A outputs two plaintexts m_0, $m_1 \in \mathcal{M}$ as in the Definition 3 and a chosen key-pair of the form $(gpk_{G^{\star}_S}, msk_{G^{\star}_S})$. The challenger $\mathcal{CH}$ flips a coin $b \in_R \{0, 1\}$. It computes and sends:
   $\boldsymbol{c}^{\star} = DisSignCrypt(m_b, gpk_{G^{\star}_S}, msk_{G^{\star}_S}, gpk_{G_R})$ *to $\mathcal{A}$.*
3. *$\mathcal{A}$ performs a number of new queries as described in the first stage with a restriction that $\mathcal{A}$ may not ask the DisDeSignCrypt query of the challenged ciphertext $\boldsymbol{c}^{\star}$. At the end of the game, $\mathcal{A}$ outputs a bit b' and wins if $b' = b$.*

$\mathcal{A}$'s advantage is defined to be $Adv^{GSC-IND-CCA}(\mathcal{A}) = 2Pr[b' = b] - 1$.

Definition 8. (Unforgeability) *We say that GSC^{e}_{pairg} is existentially unforgeable against chosen-message attack (i.e. GSC-EUF-CMA) if no probabilistic polynomial time forger $\mathcal{F}$ has a non-negligible advantage in the following game:*

1. *The challenger $\mathcal{CH}$ generates two pairs (msk_{G_S}, gpk_{G_S}) and (msk_{G_R}, gpk_{G_R}). The $\mathcal{CH}$ keeps msk_{G_S} and msk_{G_R} secret while pair (gpk_{G_S}, gpk_{G_R}) is given to the forger $\mathcal{F}$.*
2. *The forger $\mathcal{F}$ makes series of DisSignCrypt $(m, gpk_{G_S}, msk_{G_S}, gpk_{G'_R})$ and DisDeSignCrypt $(\boldsymbol{c}, msk_{G_R})$ adaptive queries exactly in the same way as done in the step 2 of GSC-IND-CCA game (Definition 7).*
3. *$\mathcal{F}$ produces a ciphertext $\boldsymbol{c}^{\star}$ and a valid key pair $(msk_{G^{\star}_R}, gpk_{G^{\star}_R})$ and wins the game if (i) DisDeSignCrypt $(\boldsymbol{c}^{\star}, msk_{G^{\star}_R})$ returns a tuple $(m^{\star}, s, Apnd_m (gpk^{1}_{G_S}))$ such that true $\leftarrow$ Verify$((m^{\star}, s, Apnd_m(gpk^{1}_{G_S}))$; and (ii) ciphertext $\boldsymbol{c}^{\star}$ is not the output of any DisSignCrypt Oracle during the game.*

5 DSC^{e}_{pairg} and GSC^{e}_{pairg} *Schemes*

5.1 Distributed Signcryption Scheme Based on Pairings: DSC^{e}_{pairg}

The scheme ensures message confidentiality and signature unforgeability and is described below.

Setup: Same as described in Section 3.
InzGroup: For construction of the group, we follow the approach as proposed by Gupta, Pillai and Saxena [10]. We assume that there are $n+1$ members in the group including $\mathcal{GM}$ who is responsible for constructing the gpk and updating the keys of the group members. Let $\mathbb{G}_1$ be a cyclic group of prime order q and P be an arbitrary generator of $\mathbb{G}_1$. In order to initialized the group, $\mathcal{GM}$ carries out the following steps:
step 1. $\mathcal{GM}$ selects a set of pairs of non zero integers $(x_i, y_i)_{i=0}^{i=n} \in_R \mathbb{Z}_q$ and computes Lagrange's interpolation polynomial $f(x) = \sum_{i=0}^{n} \beta_i x^i \mod q$ passing through these points $(x_i, y_i)_{i=0}^{n}$. Clearly, the polynomial $f(x)$ satisfies the condition: $f(x_i) = y_i \mod q,\ 0 \le i \le n$. We can rewrite the polynomial in the form:

$$f(x) = \beta_0 + \sum_{j=1}^{n} \beta_j \sum_{k=1}^{n} x^k - \sum_{j=1}^{n} \sum_{k=1, j\neq k}^{n} \beta_j x^k \mod q$$

or

$$f(x) = \beta_0 + \beta\delta_1(x) - \delta_2(x) \mod q.$$

where $\beta = \sum_{j=1}^{n} \beta_j$, $\delta_1(x) = \sum_{j=1}^{n} x^j$ and $\delta_2(x) = \sum_{j=1}^{n} \sum_{k=1,j\neq k}^{n} \beta_j x^k$.
Step 2. The $\mathcal{GM}$ constructs new polynomials of the form $F_i(x) = f(x) - y_i \mod q$ for $i = 0, 1, \ldots, n$ (the polynomial $F_i(x)$ vanishes at x_i for each value of $0 \le i \le n$) and defines the functions $F_i^*(x, P) = F_i(x) \cdot P$ on the elements of $\mathbb{G}_1$. Clearly, $F_i^*(x_i, P) = \mathcal{O} \mod q$ for $0 \le i \le n$.
Step 3. The $\mathcal{GM}$ keeps β_i secret for all i and computes $\beta_0 \cdot P$ and $\beta \cdot P$. Then he/she selects a random number $\gamma \in_R \mathbb{Z}_q^*$ and computes parameters ρ_l $(l = 0, 1 \ldots, n)$ such that $\rho_l = \overline{\gamma}\delta_2(x_l) \mod q$ where $\overline{\gamma}$ is the multiplicative inverse of γ. Finally, the $\mathcal{GM}$ selects random $k \in_R \mathbb{Z}_q$, computes $\gamma \cdot P, kP$ and keeps it secret.
Step 4. The $\mathcal{GM}$ defines gpk by the quadruple $(\beta_0 P, \beta P, \gamma P, kP)$ and the secret key of the l^{th} member of the group by the tuple (x_l, y_l, ρ_l) (for all $0 \le l \le n$).
KeyGen: We let the user U be a sender while the receivers need $\mathcal{GM}$ to generate their keys. The user U picks a random $x_U \in \mathbb{Z}_q^*$ and sets his public key to $Y_U = x_U P \in \mathbb{G}_2$. His private key is x_U. The $\mathcal{GM}$ generates $gpk = (\beta_0 P, \beta P, \gamma P, kP)$ and individual members' secret keys (x_l, y_l, ρ_l) for all $l = 0, 1, \cdots, n$. $\mathcal{GM}$ then distributes members' secret keys securely to the members of the group. Let us denote the sender and the receiver by $U = S$ and $U = R$ respectively. We also denote key pair of the sender by (x_S, Y_S) and key pair of the receiver by $((x_l, y_l, \rho_l), gpk)$, where $gpk = (\beta_0 P, \beta P, \gamma P, kP)$.
DisSignCrypt: To signcrypt a message $m \in \{0,1\}^\mu$ intended to be sent to a member of the receiving group (say R) , the sender S carries out the following steps:

1. Pick $k_1 \in \mathbb{Z}_q^*$ randomly and compute $U = k_1 P \in \mathbb{G}_1$, $U_0 = k_1 kP + k_1\beta_0 P \in \mathbb{G}_1$, $U_1 = k_1\beta P \in \mathbb{G}_1$, $U_2 = k_1\gamma P \in \mathbb{G}_1$.
2. Compute $U_3 = x_S H_1(m, U, kP, k_1 kP) \in \mathbb{G}_1$.
3. Compute $Z = (m||Y_S||U_3) \oplus H_2(U, kP, k_1 kP) \in \{0,1\}^{\mu+2L}$ and construct the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, Z >\in \mathbb{G}_1^4 \times \{0,1\}^{\mu+2L}$.
4. Send $\boldsymbol{c}$ to R.

DisDeSignCrypt: On receiving the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, Z >\in \mathbb{G}_1^4 \times \{0,1\}^{\mu+2L}$, the receiver R carries out the following steps:

1. Compute $k_1 kP = U_0 + w_l U_1 - \rho_l U_2 - y_l U \in \mathbb{G}_1$ using member's secret key where $w_l = \sum_{i=1}^{i=n} x_l^i$.

2. Compute $(m||Y_S||U_3) = Z \oplus H_2(U, kP, k_1kP)$ and return $\perp$ if $Y_S \notin \mathbb{G}_1$.
3. Compute $H = H_1(m, U, kP, k_1kP) \in \mathbb{G}_1$ and then check if $e(H, Y_S) = e(U_3, P)$. If the condition holds, output $(m, (U, kP, k_1kP, U_3), Y_S)$, otherwise; reject the ciphertext.

Verify: For a message signature pair $(m, (U, kP, k_1kP, U_3))$ and the sender's public key Y_S, the algorithm checks if: $e(Y_S, H_1(m, U, kP, k_1kP)) = e(U_3, P)$. If the condition holds, it outputs ***'true'***; otherwise it outputs ***'false'***.
Note that the scheme works by virtue of the fact that:

$$\begin{aligned}
k_1kP &= U_0 + w_lU_1 - \rho_lU_2 - y_lU \\
&= k_1kP + k_1\beta_0P + (\sum_{i=1}^{i=n} x_l^i)k_1\beta P - \rho_lk_1\gamma P - y_lk_1P \\
&= k_1kP + k_1\big(\beta_0 + (\sum_{i=1}^{i=n} x_l^i)\beta - \rho_l\gamma - y_l\big)P \\
&= k_1kP + k_1F_i(x_l).P = k_1kP + k_1\mathcal{O}
\end{aligned}$$

Remark: From the Proposition 1 of [10], it can be easily proved that the constructed designated group in the above scheme cannot be modified illegally (i.e. the quadruple $(\beta_0P, \beta P, \gamma P, kP)$).

5.2 Group Signcryption Scheme Based on Pairings: GSC^e_{pairg}

In this section we present group signcryption from pairings (GSC^e_{pairg}) which is an extension of the distributed signcryption scheme DSC^e_{pairg}. Apart from the message confidentiality and signature unforgeability, proposed GSC^e_{pairg} also preserves anonymity of the member who DisSigncrypts the message. This implies that except the group manager, no one is able to find out the identity of the sender. Let $\mathcal{G_S}$ and $\mathcal{G_R}$ be two designated groups who want to communicate with each other. The scheme works as follows.

Setup: Same as described in Section 3 except that the function H_1 in this scheme is defined as: $H_1 : \{0,1\}^\mu \times \mathbb{G}_1^5 \rightarrow \mathbb{G}_1$.
InzGroup Same as described in Section 5.1
KeyGen: The group manager of the group $\mathcal{G_S}$ generates $gpk = (\beta_0'P, \beta'P, \gamma'P, k'P)$ and individual member's secret keys $(x'_{l'}, y'_{l'}, \rho'_{l'})$, (for all $l' = 0, 1, \cdots, n'$). The group manager of the group $\mathcal{G_R}$ generates $gpk = (\beta_0P, \beta P, \gamma P, kP)$ and individual member's secret keys (x_l, y_l, ρ_l), (for all $l = 0, 1, \cdots, n$). Let $S_{l'}$ be a the l'^{th} member of the group $\mathcal{G_S}$ who signcrypts the message on behalf of the group and let R_l be the l^{th} member of the group $\mathcal{G_R}$ who De-signcrypts the message on behalf of the group. Thus the sender-key pair is $\big((x'_l, y'_l, \rho'_l), (\beta'_0P, \beta'P, \gamma'P, k'P)\big)$ and receiver key pair is $\big((x_l, y_l, \rho_l), (\beta_0P, \beta P, \gamma P, kP)\big)$.
DisSignCrypt: To signcrypt the message m intended to be sent to $\mathcal{R}$, a member of the receiving group $\mathcal{G_R}$, the sender $\mathcal{S}$, a member of $\mathcal{G_S}$ carries out following steps:

1. Pick random $k_1, k_2, k_3, x_S \in \mathbb{Z}_q^*$ and compute k_3P, $U = k_1P \in \mathbb{G}_1$, $U_1 = k_1\beta P$, $U_2 = k_1\gamma P$ (we assume that elements of $\mathbb{Z}_q^*$ can be represented in λ bits).

2. Compute $\omega'_{l'} = \sum_{j=1}^{j=n} {x'_{l'}}^{j}$. Then compute $\overline{U} = x_S k'P$, $\overline{U}_0 = k_2\beta'_0P + k_2P$, $\overline{U}_1 = k_2\omega'_{l'}\beta'P$ and $\overline{U}_2 = k_2\rho'_{l'}\gamma'P$.
3. Compute $\overline{U}_3 = k_2k_3y'_{l'}P$ and $U_0 = k_1kP + k_1\beta_0P + k_2k_3P - \overline{U}_3$.
4. Compute $U_3 = x_S H_1(m, k_3P, \overline{U}_0, U, kP, k_1kP) \in \mathbb{G}_1$.
5. Compute $Z = (m||\overline{U}||U_3) \oplus H_2(U, kP, k_1kP) \in \{0,1\}^{\mu+2L}$ The ciphertext is given by $\boldsymbol{c} =< U, U_0, U_1, U_2, \overline{U}_0, \overline{U}_1, \overline{U}_2, Z, k_3 >\in \mathbb{G}_1^7 \times \{0,1\}^{\mu+\lambda+2L}$. S sends ciphertext to R.

DisDeSignCrypt: On receiving a ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, \overline{U}_0, \overline{U}_1, \overline{U}_2, Z, k_3 >\in \mathbb{G}_1^7 \times \{0,1\}^{\mu+\lambda+2L}$, the receiver $\mathcal{R}$ carries out following steps:

1. Compute $\omega_l = \sum_{i=1}^{i=n} x_l^i$ and then compute $k_1kP = (U_0 + w_lU_1 - \rho_lU_2 - y_lU) - k_3(\overline{U}_0 + \overline{U}_1 - \overline{U}_2)$ using members secret key.
2. Compute $Z = (m||\overline{U}||U_3) \oplus H_2(U, kP, k_1kP)$ and return $\perp$ if $\overline{U} \notin \mathbb{G}_1$.
3. Compute k_3P & $H = H_1(m, k_3P, \overline{U}_0, U, kP, k_1kP) \in \mathbb{G}_1$ and then check if $e(H, \overline{U}) = e(U_3, k'P)$. If this condition does not hold, reject the ciphertext.

Verify: For given pair $(m, (k_3P, \overline{U}_0, U, kP, k_1kP, U_3))$ and given $\overline{U}$, the algorithm checks whether $e(\overline{U}, H_1(m, k_3P, \overline{U}_0, U, kP, k_1kP)) = e(U_3, k'P)$?. If the condition holds, it outputs **true**, otherwise, it outputs **false**.
Note that the scheme works by virtue of the fact that:

$$\begin{aligned}
k_1kP &= (U_0 + w_lU_1 - \rho_lU_2 - y_lU) - k_3(\overline{U}_0 + \overline{U}_1 - \overline{U}_2) \\
&= k_1kP + k_1\beta_0P + k_2k_3P - k_2k_3y'_{l'}P + (\sum_{i=1}^{i=n} x_l^i)k_1\beta P \\
&\quad - \rho_lk_1\gamma P - y_lk_1P - k_3(k_2\beta'_0P + k_2P + k_2\omega'_{l'}\beta'P - k_2\rho'_{l'}\gamma'P) \\
&= k_1kP + k_2k_3P + k_1\big(\beta_0 + (\sum_{i=1}^{i=n} x_l^i)\beta - \rho_l\gamma - y_l\big)P \\
&\qquad - k_2k_3\big(\beta'_0 + \sum_{j=1}^{j=n} {x'_{l'}}^{j} - \rho'_{l'}\gamma' - y'_{l'}\big)P - k_2k_3P \\
&= k_1kP + k_1\mathcal{O} + k_2k_3\mathcal{O}
\end{aligned}$$

6 Performance and Security Analysis

In this section, we analyze the complexity and also computational overheads of proposed distributed signcryption and its extension to a group signcryption. We then discuss its security aspects.

6.1 Complexity Analysis and Expansion Factor

The computational cost (**CC**) of proposed schemes can be calculated in terms of scalar multiplications (SM) in group $\mathbb{G}_1$, and computation of pairings (PC) $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_1$. Computational overheads can be determined in term of the expansion factor (η the ratio of number of bits required to represent the ciphertext and the plaintext). In Table 1, we summarize computational cost and communication overheads of distributed signcryption and the group signcryption.

Table 1. Computational Cost and Expansion Factor of Proposed Methods

Proposed Methods	Approximate Computational Cost (DisSignCrypt & DisDeSignCrypt)	Verification Cost	Approximate Expansion Factor η
DSC^e_{pairg}	$C_{DSC} + C_{DDSC} = (6SM + 3SM + 2PC)$	2 PC	$1 + \frac{6L}{\mu}$
GSC^e_{pairg}	$C_{GSC} + C_{GDSC} = (13SM + 5SM + 2PC)$	2PC	$1 + \frac{9L + \lceil \log q \lceil}{\mu}$

[C_{DSC}: CC for distributed signcryption, C_{DDSC}: CC for distributed de-signcryption, C_{GSC}: CC for group signcryption, C_{GDSC}: CC for group de-signcryption]
Note 1: As compared to the computational cost of scalar multiplications in $\mathbb{G}_1$ and computation of pairings, the cost of other associated operations like field multiplication, integer multiplication, addition of two elements of $\mathbb{G}_1$ are significantly small. We can neglect these.

6.2 Security Analysis

As both the schemes DSC^e_{pairg} & GSC^e_{pairg} are extensions of LYWDC-ESC scheme, their security also rely on hardness of GDHP (in GDH groups $(\mathbb{G}_1, \mathbb{G}_2)$) and therefore security proofs can be given using the same approach as given in [15,16,22]. However, we use security notions proposed in section 4 and give security proof on the random oracle model [4] following approach as given by Yang et al and Li et al [15,22].

Security Analysis of DSC^e_{pairg}

Theorem 6.2.1. Let $\mathfrak{K}_\mathfrak{p}$ be a security parameter. If there exists a PPT adversary $\mathcal{A}$ that has at least $\epsilon(\mathfrak{K}_\mathfrak{p})$ advantage against the DSC-IND-CCA security of the proposed DSC^e_{pairg} scheme when asking q_{h_i} queries to random oracle H_i for $(i = 1, 2)$, q_{DSC} queries to DisSignCrypt oracle and q_{DDSC} to DisDeSignCrypt oracle then there exists a PPT algorithm $\mathcal{B}$ which can solve the Gap Diffie-Hellman Problem with probability at least $(1 - q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} - q_{h_1}q_{DDSC}/2^{2\mathfrak{K}_\mathfrak{p}})\epsilon(\mathfrak{K}_\mathfrak{p}))$. If the running time of $\mathcal{A}$ is t then the running time of $\mathcal{B}$ is $t' < t + (4q_{DDSC} + 2q_{h_1} + 2q_{h_2})t_e$, where t_e is the time required for one pairing computation.

Proof. See Appendix A.

Theorem 6.2.2. Let $\mathfrak{K}_\mathfrak{p}$ be a security parameter. If there exists a PPT adversary $\mathcal{F}$ that has at least $\epsilon(\mathfrak{K}_\mathfrak{p})$ non negligible advantage against the DSC-EUF-CMA security of the proposed DSC^e_{pairg} scheme when making q_{h_i} queries to random oracle H_i for $(i = 1, 2)$, q_{DSC} queries to DisSignCrypt oracle and q_{DDSC} to DisDeSignCrypt oracle then there exists a PPT algorithm $\mathcal{B}$ which can solve the Gap Diffie-Hellman Problem with the probability at least $(1 - q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} - q_{h_1}q_{DDSC}/2^{2\mathfrak{K}_\mathfrak{p}})\epsilon(\mathfrak{K}_\mathfrak{p})$. If the running time of $\mathcal{A}$ is t then the running time of $\mathcal{B}$ is $t' < t + (2q_{h_1} + 2q_{h_2} + 4q_{DDSC})t_e$ where t_e is the time required for one pairing computation.

Proof. See Appendix B.

Security Analysis GSC^e_{pairg}

Theorem 6.2.3. Let $\mathfrak{K}_\mathfrak{p}$ be a security parameter. If there exists a PPT adversary $\mathcal{A}$ that has at least $\epsilon(\mathfrak{K}_\mathfrak{p})$ advantage against the GSC-IND-CCA security of the proposed DSC^e_{pairg} scheme when asking q_{h_i} queries to random oracle H_i for $(i = 1, 2)$, q_{DSC} queries to DisSignCrypt oracle and q_{DDSC} to DisDeSignCrypt oracle then there exists a PPT algorithm $\mathcal{B}$ which can solve the Gap Diffie-Hellman Problem with probability at least $p_1(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p})\epsilon(\mathfrak{K}_\mathfrak{p})$. If the running time of $\mathcal{A}$ is t then the running time of $\mathcal{B} = t' < t + (4q_{DDSC} + 2q_{h_1} + 2q_{h_2})t_e$, where t_e is the time required for one pairing computation. Here $p_1(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p})$ is the least probability that the simulation does not fail i.e. $p_1(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p}) \leq Pr$[simulation does not fail].

Proof. Proof of theorem can be done using the same approach as given in theorem 6.2.1. [It is to be noted that the probability that the simulation does not fail is a function of $q_{h_1}, q_{h_2}, q_{DDSC}$ and $\mathfrak{K}_\mathfrak{p}$. Therefore we denoted it by $p_1(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p})$.]

Theorem 6.2.4. Let $\mathfrak{K}_\mathfrak{p}$ be a security parameter. If there exists a PPT adversary $\mathcal{F}$ that has at least $\epsilon(\mathfrak{K}_\mathfrak{p})$ non negligible advantage against the GSC-EUF-CMA security of the proposed GSC^e_{pairg} scheme when making q_{h_i} queries to random oracle H_i for $(i = 1, 2)$, q_{DSC} queries to DisSignCrypt oracle and q_{DDSC} to DisDeSignCrypt oracle, then there exists a PPT algorithm $\mathcal{B}$ which can solve the Gap Diffie-Hellman Problem with probability at least $p_2(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p})\epsilon(\mathfrak{K}_\mathfrak{p})$. If the running time of $\mathcal{A}$ is t then the running time of $\mathcal{B} = t' < t + (2q_{h_1} + 2q_{h_2} + 4q_{DDSC})t_e$, where t_e is time required for one pairing computation. Here probability $p_2(q_{h_1}, q_{h_2}, q_{DDSC}, \mathfrak{K}_\mathfrak{p}) \leq Pr$[simulation does not fail].

Proof. Proof of the theorem can be done using the same approach as given in theorem 6.2.2.

Anonymity: It is clear that the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, \overline{U}_0, \overline{U}_1, \overline{U}_2, Z, k_3 >$ does not contain any information about the group's identity or the sender's identity. Information of both the identities is encrypted through public key encryption of symmetric encryption as described in the DisSigncrypt algorithm. (Particularly, the sender's identity is hidden in $\overline{U}_1$, $\overline{U}_2$ and $\overline{U}_3$).

The sender's identity is also not required for DisDeSignCrypt and to Verify the message. However, the verification algorithm requires one part of the sender's group public key. But it does not leave any information about the sender. It leaks information of the sending group and only group manager can check the identity of the sender. Thus GSC^e_{pairg} ensures anonymity property.

6.3 Comparative Study

It may be noted that no separate encryption/decryption algorithm is required in DSC^e_{pairg} and GSC^e_{pairg} schemes (it is done through xor operation with key derived from hash function). However, schemes proposed in [10,13,19] have a separate encryption/decryption algorithm. Comparative study of proposed scheme and existing distributed signcryption / group signcryption schemes is given in the Table 2.

Table 2. Comparative Study of Distributed Signcryption

Schemes	CC and CO	Scheme Based On	Security Analysis
MVS (DSC and GSC)	$O(\mid G \mid)$	$\mathbb{Z}_p^\star$	Not Given
KMS (DSC and GSC)	$O(\mid G \mid)$	$\mathbb{Z}_p^\star$	Heuristic arguments
GPS	Constant	HEC	Heuristic arguments
Proposed DSC^e_{pairg} & GSC^e_{pairg}	Constant	Pairing	DSC-IND-CCA, DSC-EUF-CMA

[Constant: Independent of the number of members in the designated groups, $O(\mid G \mid)$: As function of $\mid G \mid$ where G = set of users, CC: Computational Complexity, CO: Communication Overhead]

7 Conclusion

We proposed a method for distributed signcryption from pairings which is an extension of the signcryption scheme with key privacy proposed by Li, Yang, Wong, Deng and Chow. We further extended the distributed signcryption into group signcryption. Proofs of security have been given for the proposed distributed signcryption and group signcryption schemes along with performance analysis. Comparative study of proposed schemes with other existing schemes was also performed. Further work on other security notions (like key privacy etc.) for our distributed signcryption and group signcryption is being carried out.

Acknowledgments. The authors would like to extend their gratitude to Prof. Rana Barua, ISI Kolkata for useful interactions during the visit of the author to ISI Kolkata. Authors are also grateful to Prof C. E. Veni Madhavan, Prof A. Tripathi, Dr. S S Bedi, Dr. Meena Kumari and N R Pillai for the technical discussions and suggestions.

References

1. An, J.H., Dodis, Y., Rabin, T.: On the Security of Joint Signature and Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002)
2. Bao, H., Cao, Z., Qian, H.: On the Security of a Group Signcryption Scheme from Distributed Signcryption Scheme. In: Desmedt, Y.G., Wang, H., Mu, Y., Li, Y. (eds.) CANS 2005. LNCS, vol. 3810, pp. 26–34. Springer, Heidelberg (2005)

3. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
4. Bellare, M., Rogaway, P.: Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the First ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
5. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in Elliptic curves in cryptography. Cambridge University Press, Cambridge (2005)
6. Boneh, D.: The Decision Diffie-Hellman Problem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 48–63. Springer, Heidelberg (1998)
7. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
8. Frey, G., Müller, M., Rück, H.G.: The Tate Pairing and Discrete Logarithm Applied to Elliptic Curve Cryptosystems. IEEE Trans. of Information Theory 45(5), 1717–1719 (1999)
9. Galbraith, S.D., Harrison, K., Soldera, D.: Implementing the Tate Pairing. In: Fieker, C., Kohel, D.R. (eds.) ANTS 2002. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
10. Gupta, I., Pillai, N.R., Saxena, P.K.: Distributed Signcryption Scheme on Hyper-elliptic Curve. In: Proceedings of the Fourth IASTED International Conference on Communication, Network and Information security: CNIS 2007, pp. 33–39. Acta Press, Calgary (2007)
11. Joux, A.: A One Round Protocol for Tripartite Diffie-Hellman. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 385–394. Springer, Heidelberg (2000)
12. Joux, A., Nguyen, K.: Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups. Journal of cryptology 16(4), 239–247 (2003)
13. Kwak, D.J., Moon, S.J.: Efficient Distributed Signcryption Scheme as Group Signcryption. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 403–417. Springer, Heidelberg (2003)
14. Kwak, D., Moon, S., Wang, G., Deng, R.H.: A Secure Extension of the Kwak-Moon Group Signcryption Scheme. Computer & Security 25, 435–444 (2006)
15. Li, C.K., Yang, G., Wong, D.S., Deng, X., Chow, S.S.M.: An Efficient Signcryption Scheme with Key Privacy. In: López, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 78–93. Springer, Heidelberg (2007)
16. Libert, B., Quisquater, J.-J.: Efficient Signcryption with Key Privacy from Gap Diffie-Hellman Groups. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 187–200. Springer, Heidelberg (2004)
17. Libert, B.: New Secure Applications of Bilinear Maps in Cryptography. PhD Thesis, Microelectronics Laboratory Laboratory, Université Catholique de Louvain (January 2006)
18. Miller, V.S.: The Weil Pairing and Its Efficient Calculation. Journal of Cryptology 17, 235–261 (2004)
19. Mu, Y., Varadharajan, V.: Distributed Signcryption. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 155–164. Springer, Heidelberg (2000)

20. Okamoto, T., Pointcheval, D.: The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
21. Tan, C.-H.: Analysis of Improved Signcryption Scheme with Key Privacy. Information Processing Letter 99(4), 135–138 (2006)
22. Yang, G., Wong, D.S., Deng, X.: Analysis and Improvement of a Signcryption Scheme with Key Privacy. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 218–232. Springer, Heidelberg (2005)
23. Zheng, Y.: Digital Signcryption or How to Achieve Cost (Signature & Encryption) << Cost(Signature) + Cost(Encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997)

A Proof of Theorem 6.2.1

To prove the theorem, we have to construct an algorithm that solves GDHP, assuming that adversary has non negligible advantage against the $DSC-IND-CCA$. The algorithm $\mathcal{B}$ starts with the common parameter generation subroutine and then uses $\mathcal{A}$ as a subroutine to solve GDHP.

We assume that $\mathcal{B}$ is given (aP, bP), a random instant of GDHP. Our aim is to compute abP with the help of DDHP in $(\mathbb{G}_1, \mathbb{G}_2)$. Let $gpk_G = (\beta_0^\star P, \beta^\star P, \gamma^\star P, bP)$ be a challenged public key of any group G. The algorithm $\mathcal{B}$ runs the subroutine $\mathcal{A}$ with the challenged $gpk_G = (\beta_0^\star P, \beta^\star P, \gamma^\star P, bP)$. $\mathcal{A}$ adaptively performs the hash queries, the DisSignCrypt queries and the DisDeSignCrypt queries. $\mathcal{B}$ maintains the two lists L_1 and L_2 for handling the queries to keep track of the answers given to random oracles H_1 and H_2 respectively. All the queries are performed as follows.

H_1-queries: When a hash query H_1 is made on the input (m, P_1, P_2, P_3), where $m \in \{0,1\}^\mu$ and $P_1, P_2, P_3 \in \mathbb{G}_1$, $\mathcal{B}$ first checks if $e(P_1, P_2) = e(P, P_3)$ i.e. (P, P_1, P_2, P_3) is valid DDH quadruple.

If it is, then $\mathcal{B}$ checks if the list L_1 contains the query tuple (m, P_1, P_2, P_3). If it contains, the existing result in L_1 is returned. If the tuple (m, P_1, P_2, P_3) is not in L_1 but $(m, P_1, P_2, \top)$ is in L_1, then $\top$ is replaced by P_3 and the value corresponding to the query tuple $(m, P_1, P_2, \top)$ is used as the value to be returned, where $\top$ is a special symbol. Otherwise $\mathcal{B}$ selects random $r' \in \mathbb{Z}_q$, and returns $r'P$. Finally, $\mathcal{B}$ updates the list L_1 to answer the future queries.

If (P, P_1, P_2, P_3) is not a DDH quadruple, then $\mathcal{B}$ checks if the list L_1 contains the query tuple. If yes, then $\mathcal{B}$ returns existing value, otherwise $\mathcal{B}$ selects random $r \in \mathbb{Z}_q$ and returns rP. The list L_1 is updated to answer the future queries.

H_2-queries: Hash queries to H_2 can be handled in a similar manner as H_1 queries.

DisSignCrypt-queries: $\mathcal{A}$ selects the plaintext m, and the group public key $gpk_{G_R} = (\beta_0 P, \beta P, \gamma P, kP)$ of the receiver and submits it for DisSignCrypt queries. On receiving (m, gpk_{G_R}), $\mathcal{B}$ does the following:

- $\mathcal{B}$ first checks if $gpk_{G_R} \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$ and $gpk_G \neq gpk_{G_R}$. If not, then $\mathcal{B}$ returns the symbol $\perp$ for rejection. Otherwise, $\mathcal{B}$ selects random $k_1, k_2 \in \mathbb{Z}_q$

and applies the same procedure as described in the section 5.1 to compute U, U_0, U_1, U_2.
- $\mathcal{B}$ simulates hash functions H_1 and H_2 to obtain $r_1P \leftarrow H_1(m, U, kP, k_1kP)$ and $H_2(U, kP, k_1kP)$.
- $\mathcal{B}$ computes $U_3 = r_1Y_u$ and $Z = (m\|Y_u\|r_1Y_u) \oplus H_2(U, kP, k_1kP)$ as described in the section 5.1 and returns the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, Z >$.

DisDeSignCrypt queries: $\mathcal{A}$ submits $\boldsymbol{c}$ for DisDeSignCrypt query. On receiving the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, Z >$, $\mathcal{B}$ computes λ using msk and caries out the following steps.

Step 1. $\mathcal{B}$ checks if the list L_2 contains a tuple of the form (U, bP, λ) such that (P, U, bP, λ) is a valid DDH quadruple.

- If yes, then the existing value will be used as the value for $H_2(P, U, bP, \lambda)$.
- If not, then $\mathcal{B}$ inserts a new entry into the list L_2 by saving $(U, bP, \top)$ as query tuple and the value randomly drawn from the range of H_2 is returned as the oracle value.

It may be noted that the special symbol $\top$ is used as a marker for denoting that the real value should be the solution of the CDH problem instance (U, bP). This step ensures that the values of $H_2(U, bP, \lambda)$ is fixed before $\boldsymbol{c}$ is de-signcrypted.

Step 2. $\mathcal{B}$ computes $(m\|Y_S\|U_3) = Z \oplus H_2(U, bP, \lambda)$ and $\mathcal{B}$ checks if the list L_1 contains a tuple of the form (m, U, bP, λ) such that (P, U, bP, λ) is a valid DDH quadruple or $\lambda = \top$.

- If L_1 contains (m, U, bP, λ), then $\mathcal{B}$ uses existing result as a value of $H_1(m, U, bP, \lambda)$ and uses λ to update the the corresponding entry in L_2. If the tuple $(m, U, bP, \top)$ is in L_1 then the existing value will be used. $\mathcal{B}$ updates list L_1 replacing $\top$ by λ if $\mathcal{B}$ obtains value of λ from L_2.
- If L_1 does not contain (m, U, bP, λ), then $\mathcal{B}$ inserts new entry into the list L_2 by saving (m, U, bP, λ) as query tuple. $\mathcal{B}$ selects random $r_2 \in \mathbb{Z}_q$ and returns r_2P as oracle value of $H_1(m, U, bP, \lambda)$.

Step 3. Then $\mathcal{B}$ checks if the (Verify) condition $e(P, U_3) = e(H_1(m, U, bP, \lambda), Y_S)$ holds.

- If yes and (P, U, bP, λ) is valid DDH quadruple i.e. $\big(e(P, \lambda) = e(U, bP)\big)$, then $\mathcal{B}$ returns the message signature pair $(m, (U, bP, \lambda, U_3)$ and the sender identity Y_S. If (Verify) condition holds but H_1 is simulated on the tuple $(m, U, bP, \top)$ in 'Step 2' i.e. $\lambda = \top$, then $\mathcal{B}$ halts with failure.
- Otherwise, $\mathcal{B}$ returns symbol $\perp$ for rejection of the ciphertext.

At the end of the first stage, $\mathcal{A}$ provides $\mathcal{B}$ two equal length (μ-bit) plaintexts m_0 and m_1, which have never been queried to DisSignCrypt, together with an arbitrary senders private key $x_S^\star \in \mathbb{Z}_q^\star$ (say) and requires a challenge ciphertext built under the challenged group public key $gpk_G = (P_{\beta_0^\star}, P_{\beta^\star}, P_{\gamma^\star}, P_{\mathfrak{b}})$. Suppose that the public key associated with $x_S^\star$ is $Y_S^\star = x_S^\star P$.

$\mathcal{B}$ randomly picks $\tilde{b} \in_R \{0,1\}$, binary strings $r^\star \in \mathbb{Z}_q$ & $Z^\star \in \{0,1\}^{\mu+2L}$ and sets $U^\star = aP$. Then $\mathcal{B}$ carries out the following steps.

– $\mathcal{B}$ selects three random multiples of aP and assign values to $U_0^\star$, $U_1^\star$ and $U_2^\star$.
– $\mathcal{B}$ assigns the value $r^\star P$ to $H_1(m_{\tilde{b}}, aP, bP, \top)$ and updates list L_1 by saving $(m_{\tilde{b}}, aP,$
$bP, \top)$ as query tuples.
– $\mathcal{B}$ sets $U_3^\star = r^\star Y_S^\star$ and computes $H_2(aP, bP, \top) = Z^\star \oplus (m_{\tilde{b}} \| Y_S^\star \| r^\star Y_S^\star)$.
– $\mathcal{B}$ returns the ciphertext $\boldsymbol{c}^\star =< U^\star, U_0^\star, U_1^\star, U_2^\star, Z^\star >$

Once $\mathcal{A}$ receives the challenged ciphertext, $\mathcal{A}$ performs a second series of queries. These queries are handled as in the first stage. But $\mathcal{A}$ is not allowed to DisDeSignCrypt query of the challenged ciphertext. If $\mathcal{A}$ queries H_1 or H_2 with (m, aP, bP, λ) / (aP, bP, λ) such that (P, aP, bP, λ) is a DDH quadruple, then $\mathcal{B}$ outputs λ and halts. If $\mathcal{A}$ halts without making this query, $\mathcal{B}$ outputs a random point in $\mathbb{G}_1$ and halts.

As we have assumed that the adversary $\mathcal{A}$ has non-negligible advantage ϵ over $DSC-IND-CCA$, we have to find probability of success of $\mathcal{B}$ in solving GDHP.

Let Evt denote an event that $\mathcal{A}$ asked the H_1 query on the tuple $(., aP, bP, abP)$ for any message '.' or H_2 query on the tuple (aP, bP, abP) during the simulation of H_1 and H_2. Let $\neg Evt$ be the event that neither $(., aP, bP, abP)$ is queried on H_1 nor (aP, bP, abP) is queried on H_2. We assume that simulation of the attacks environment is perfect i.e an attack where $\mathcal{A}$ interacts with oracles. We claim that for $\neg Evt$, $\mathcal{A}$ does not have any advantage in winning the game over random guessing.

Let $U_3^{\tilde{b}} = x_S^\star H_1(m_{\tilde{b}}, aP, bP, abP)$. Then $\boldsymbol{c}^\star = (U^\star, U_0^\star, U_1^\star, U_2^\star, Z^\star)$ is the signcryption of $m_{\tilde{b}}$ if we have $(m_{\tilde{b}} \| Y_S^\star \| U_3^b) = Z^\star \oplus H_2(aP, bP, abP)$. Clearly, in $\neg Evt$, $\mathcal{B}$ does not leak any information to $\mathcal{A}$ since H_1 and H_2 are not queried with (aP, bP, abP) and $(m_{\tilde{b}}, aP, bP, abP)$. Note that, $\mathcal{A}$ does not have any advantage in determining whether the oracle returns of H_1 on query tuple $(m_{\tilde{b}}, aP, bP, abP)$ and H_2 on query tuple (aP, bP, abP). Hence $Pr[\mathcal{A}$wins the game$|\neg Evt] = Pr[b = b'|\neg Evt] = \frac{1}{2}$. From the assumption that attacker wins the game if $b = b'$ and attacker advantage over the game is defined as $Adv^{DSC-IND-CCA}\mathcal{A} = 2 \times Pr[b = b'] - 1 \Rightarrow Pr[b = b'] = \frac{(\epsilon(\mathfrak{K}_p)+1)}{2}$, we have

$$\begin{aligned} Pr[b = b'] &= Pr[b = b'|\neg Evt]Pr[\neg Evt] + Pr[b = b'|Evt]Pr[Evt] \\ &\leq \frac{1}{2}(1 - Pr[Evt]) + Pr[Evt] \end{aligned}$$

This implies that $\frac{(\epsilon(\mathfrak{K}_p)+1)}{2} \leq \frac{1}{2}(1 + Pr[Evt])$ and hence $Pr[Evt] \geq \epsilon$.

We now consider the case when the simulation fails. We need to handle only two possibilities (i) $\mathcal{B}$ fails or rejects the ciphertext and (ii) simulation is not perfect. $\mathcal{B}$ may fail at step 3 of DisDeSignCrypt oracle simulated above when $(U, bP, \top)$ is in L_2 and $(m, U, bP, \top)$ in L_1, while $e(P, U_3) = e(H_1(m, U, bP, \lambda), Y_S)$ where $m \| Y_S \| U_3 = Z \oplus H_2(U, bP, \top)$. This implies that $\mathcal{A}$

has never queried H_1 on (m, U, bP, λ) nor H_2 on (U, bP, λ) such that $e(P, \lambda) = e(U, bP)$, while $Z_3 = W_3 \oplus U_3$ where $Z = (Z_1, Z_2, Z_3) \in \{0,1\}^{\mu} \times \mathbb{G}_1^2$ and $(W_1, W_2, W_3) = H_2(U, bP, \top) \in \{0,1\}^{\mu} \times \mathbb{G}_1^2$. Since DisDeSignCrypt oracle does not leak any information of $H_1(m, U, bP, \top)$ and W_3 to $\mathcal{A}$ at step 3, the probability that the first event will occur (i.e. $Z_3 = W_3 \oplus U_3$) is smaller than $q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}}$. In the second event, there is a chance in which DisDeSignCrypt oracle rejects the valid ciphertext. Since hash function H_1 is used to decide the acceptance of the message. The probability to reject the valid ciphertext is not greater than $q_{h_1}q_{DDSC}/2^{2\mathfrak{K}_\mathfrak{p}}$. The probability when the simulation fails is not greater than $q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} + q_{h_1}q_{DDSC}/2^{2\mathfrak{K}_\mathfrak{p}}$. Therefore, the probability when the simulation does not fail is at least $(1 - q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} - q_{h_1}q_{DDSC}/2^{2\mathfrak{K}})$.
Hence $Pr[E \wedge \text{simulation does not fail}] \geq (1 - q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} - q_{h_1}q_{DDSC}/2^{2\mathfrak{K}})\epsilon(\mathfrak{K}_\mathfrak{p})$.

The bound on $\mathcal{B}$'s computational time can be derived from pairing computations. Since every DisDeSignCrypt oracle requires four pairing computations while H_1 and H_2 oracles requires 2 pairing computations each. Thus the maximum number of possibilities for which $\mathcal{B}$ computes pairing is $(2q_{h_1} + 2q_{h_2} + 4q_{DDSC})$. Hence $\mathcal{B}$ can solve DDHP in time $t' \leq t + (2q_{h_1} + 2q_{h_2} + 4q_{DDSC})t_e$.

B Proof of Theorem 6.2.2

We first assume that there exists a forger $\mathcal{F}$ who wins the game of DSC-EUF-CMA as defined in Definition 4. It would be shown that algorithm $\mathcal{B}$ can be constructed that solves GDHP.

We follow the same procedure as in theorem 6.2.1. We assume that (aP, bP) is any random instant of CGDHP. Let $pk_U = bP$ be the challenged public key of any group user U. Assume that $\mathcal{F}$ adaptively performs hash queries, DisSignCrypt & DisDeSignCrypt queries and $\mathcal{B}$ maintains the two lists L_1 and L_2 for handling the queries to keep track of the answers given to random oracles H_1 and H_2 respectively. All the queries are performed as follows.

H_1-queries: When a hash query H_1 is made on the input (m, P_1, P_2, P_3), $\mathcal{B}$ first checks if $e(P_1, P_2) = e(P, P_3)$ i.e (P, P_1, P_2, P_3) is DDH quadruple. If it is, then $\mathcal{B}$ checks if the list L_1 contains the query tuple (m, P_1, P_2, P_3). If it contains, the existing result in L_1 is returned. If the tuple (m, P_1, P_2, P_3) is not in L_1 but $(m, P_1, P_2, \top)$ is in L_1 then $\top$ (an special symbol) is replaced by P_3 and the value is returned as the value to be returned for the tuple (m, P_1, P_2, P_3). Otherwise, $\mathcal{B}$ selects random $r \in \mathbb{Z}_q$ and returns $r(aP)$. Then $\mathcal{B}$ updates the list L_1 to answer the future queries. If (P, P_1, P_2, P_3) is not a DDH quadruple, then $\mathcal{B}$ checks if the list L_1 contains the query tuple. If yes, then $\mathcal{B}$ returns existing value, otherwise $\mathcal{B}$ selects random $r \in \mathbb{Z}_q$ and returns $r(aP)$. The list L_1 is updated to answer the future queries.

H_2-queries: H_2 queries are performed in a similar manner as in Theorem 6.2.1.

DisSignCrypt-queries: $\mathcal{A}$ submits a message m and the group public key: $\big(m, gpk_{G_R} = (\beta_0 P, \beta P, \gamma P, kP)\big)$ of the receiver. $\mathcal{B}$ does the following.

- $\mathcal{B}$ randomly selects $r' \in \mathbb{Z}_q^\star$, and sets $U = r'P$. Then $\mathcal{B}$ follows the same procedure as in Theorem 6.2.1 to compute U_0, U_1, U_2, simulates hash functions H_1 and H_2 and obtains $r_1P \leftarrow H_1(m, U, kP, k_1kP)$ and $H_2(U, kP, k_1kP)$.
- $\mathcal{B}$ computes $U_3 = r_1Y_u = bH_1(m, U, kP, k_1kP)$ & $Z = (m \| Y_u \| r_1Y_u) \oplus H_2(U, kP, k_1kP)$ and returns the ciphertext $\boldsymbol{c} =< U, U_0, U_1, U_2, Z >$.

DisDeSignCrypt queries: $\mathcal{A}$ submits $\boldsymbol{c}$ for DisDeSignCrypt query. On receiving the ciphertext $\boldsymbol{c} = < U, U_0, U_1, U_2, Z >$, $\mathcal{B}$ computes λ using msk and carries out the following steps.

Step 1. $\mathcal{B}$ follows 'steps' of DisDeSignCrypt oracle as in theorem 6.2.1.
Step 2. $\mathcal{B}$ computes $(m \| Y_S \| U_3) = Z \oplus H_2(U, bP, \lambda)$ and checks if the list L_1 contains a tuple of the form (m, U, bP, λ) such that (P, U, bP, λ) is a valid DDH quadruple or $\lambda = \top$.

– If L_1 contains (m, U, bP, λ) or $(m, U, bP, \top)$, then $\mathcal{B}$ follows the same procedure as in Step 2 of DisDeSignCrypt oracle in theorem 6.2.1.
– If L_1 does not contain (m, U, bP, λ) , then $\mathcal{B}$ inserts a new entry into the list L_2 by saving (m, U, bP, λ) as query tuple. $\mathcal{B}$ selects random $r_1 \in \mathbb{Z}_q$ and returns $r_1(aP)$ as oracle value of $H_1(m, U, bP, \lambda)$.

Step 3. This step is the same as Step 1 of DisDeSignCrypt oracle in theorem 6.2.1.

At the end of the game, the forger $\mathcal{F}$ provides a ciphertext $\boldsymbol{c}^\star$ and a key pair $(msk_{G_R^\star}, *gpk_{G_R^\star})$ where $gpk_{G_R^\star} = (\beta_0^\star, \beta^\star P, \gamma^\star P, k^\star P)$. $\mathcal{B}$ performs DisDeSignCrypt operation as we have discussed above on the input of $(\boldsymbol{c}^\star, msk_{G_R^\star}, pk_U)$, where $pk_U = bP$ is the challenged public key. If the ciphertext is valid, then DisDeSignCrypt$(\boldsymbol{c}^\star, msk_{G_R^\star}, pk_U)$ returns valid message signature pair $(m^\star, s^\star)$ and the sender public key bP. This means that $e(Y_u, H_1(m^\star, U^\star, k^\star P, \lambda^\star)) = e(bP, r^\star aP) = e(P, U_3^\star)$. Thus $U_3^\star$ must be equal to $r^\star abP$, where $r^\star$ is randomly picked up from $\mathbb{Z}_q$. The value abP can be computed using $(r^\star)^{-1}V^\star$. This yields the solution of Diffie-Hellman problem.

It is clear from the DisDeSignCrypt query above that the list L_1 must contain an entry for $H_1(m^\star, U^\star, k^\star P, \lambda^\star)$ and the corresponding oracle value in the entry must be in the form $r^\star(aP)$ for some $r^\star \in \mathbb{Z}_q$, which can be retrieved from L_1. It may be noted that if the oracle value of $H_1(m^\star, U^\star, k^\star P, \lambda^\star)$ is generated in a signcryption query i.e. of the form $r_1'P$ for some $r_1'P \leftarrow \mathbb{Z}_q$, then the values of Z and other parameters would also have been determined in that signcryption query. This implies that the target ciphertext is an output of signcryption query, which contradicts the restriction of the game defined in Definition 4.

We have to find the probability of success of $\mathcal{B}$ in solving GDHP assuming that the forger $\mathcal{F}$ has non-negligible advantage $\epsilon(\mathfrak{K}_\mathfrak{p})$ over $EUF-DSC-CCA$. As in theorem 6.2.1, it can be shown that $Pr[\mathcal{F}\text{win the game}] \geq (\mathfrak{K}_\mathfrak{p})$. Now, we consider the case when the simulation fails. Since $\mathcal{F}$ fails in Step 3 of DisDeSignCrypt oracle, the probability when the simulation fails is $\leq q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} + q_{h_1}q_{DDSC}/2^{2\mathfrak{K}}$. Hence $Pr[\mathcal{F}$ win the game $\wedge$ simulation does not fail$] \geq (1 - q_{DDSC}/2^{\mathfrak{K}_\mathfrak{p}} - q_{h_1}q_{DDSC}/2^{2\mathfrak{K}})\epsilon(\mathfrak{K}_\mathfrak{p})$ and $\mathcal{B}$ can solve GDHP in time $t' \leq t + (2q_{h_1} + 2q_{h_2} + 4q_{DDSC})t_e$.

Formal Privacy Analysis of Communication Protocols for Identity Management*

Meilof Veeningen, Benne de Weger, and Nicola Zannone

Eindhoven University of Technology, The Netherlands
{m.veeningen,b.m.m.d.weger,n.zannone}@tue.nl

Abstract. Over the years, formal methods have been developed for the analysis of security and privacy aspects of communication in IT systems. However, existing methods are insufficient to deal with privacy, especially in identity management (IdM), as they fail to take into account whether personal information can be linked to its data subject. In this paper, we propose a general formal method to analyze privacy of communication protocols for IdM. To express privacy, we represent knowledge of personal information in a three-layer model. We show how to deduce knowledge from observed messages and how to verify a range of privacy properties. We validate the approach by applying it to an IdM case study.

1 Introduction

With the growth of social networking, e-business, e-Government, and ubiquitous computing, more and more personal information is being handled over the Internet. This has increased the need to design IT systems that preserve user privacy. Not only users may demand that the IT systems they interact with preserve their privacy, but also privacy regulations (such as the EU Data Protection Directive) impose stringent requirements on the collection, processing, and disclosure of personal information.

Identity management (IdM) [1,2,3] is an emerging technology for handling personal data in distributed systems. In such a system, a service provider (SP) retrieves user credentials from (possibly multiple) identity providers (IdPs) for the authentication of users, leading to an exchange of information which may also involve the user or third parties. This information exchange impacts the user's privacy: the design of the protocols used by parties to communicate determines how much personal information is learned by the various parties, and to what extent they can link these different pieces of information. This makes it important to compare these protocols in a precise way.

Over the years, formal methods have arisen as an important tool to analyze security of communication in IT systems [4,5,6,7]. The idea is to express communication protocols in a suitable formal model, and then verify whether such a model satisfies properties such as authentication properties [5] and secrecy properties [8]. Secrecy properties, in particular, can be used to express one aspect of privacy; namely, whether a certain piece of information is known by some party in a protocol. However, they can not be used to express another fundamental aspect of privacy; namely, to what extent a piece of personal information is linkable to the corresponding data subject (who, in general, may

* This work is partially funded by the Dutch Sentinel Mobile IDM project (#10522).

not even participate directly in the protocol). Recently, formal methods have been extended to address privacy issues. However, in some cases the properties defined and verified are specific to their respective settings such as e-voting [9]. In other cases [10,11], the focus is on linking messages rather than interpreting them as personal information about a data subject as needed for the analysis of IdM systems.

In our previous work [12], we captured privacy in a general formal model for knowledge of personal information. This model expresses to which entity different pieces of information belong, and what knowledge actors have about these items and their relations. Based on this representation, we formally defined and compared identity-related properties, e.g., anonymity, pseudonymity and identifiability. However, the model cannot capture how this knowledge follows from communication as it does not allow interpretation of personal information in terms of how it was learned or what it contains.

In this paper, we combine existing formal methods and our previous work [12] by presenting a framework for analyzing which identity-related properties are satisfied by a system, given the information observed from communication protocols. This provides the machinery to compare the privacy of communication protocols in various IdM architectures in a precise way. The contributions of this paper are as follows:

- We define a *three-layer model* of (personal) information, which captures that (i) personal information in different contexts may satisfy different privacy properties; and (ii) different pieces of information may have the same contents.
- We take a representative set of cryptographic primitives and show how existing *deductive methods* for these primitives can (with some modifications) operate within our three-layer model.
- We show how to represent an actor's knowledge of personal information in terms of which personal information he can *detect*, and which personal information he can *associate* to a data subject.
- We verify, by checking these associations, which *identity-related properties*, as defined in our previous work [12], hold in a particular situation.

We demonstrate our approach by applying it to the attribute aggregation infrastructure proposed in the TAS^3 project. This infrastructure aims to satisfy a number of privacy properties: we check whether these privacy properties indeed hold, report on some problems we found, and provide some recommendations to improve the system.

The structure of the paper is as follows. We first introduce the three-layer model of personal information (§2). We use it to analyze what personal information an actor can deduce (§3) and associate (§4) from observed messages, and show how identity-related properties are defined and verified in terms of the model (§5). We apply our approach to the TAS^3 attribute aggregation infrastructure (§6), and present conclusions and directions for future work (§7).

2 A Three-Layer Model of Personal Information

In this section, we introduce a model for the representation and analysis of personal information that may be known by various actors within a system. The model can be seen as a refinement of the model proposed in our previous work [12], and is used to define actor knowledge (§4) and privacy properties (§5).

2.1 Personal Information

A piece of personal information in the digital world is a *specific* string that has a *specific* meaning as personal information about a *specific* person. We distinguish between two types of digital personal information: identifiers and data items. Identifiers are unique within the system; for data items this is not necessarily the case. The sets of identifiers and data items are denoted $\mathcal{I}$ and $\mathcal{D}$, respectively. The set $\mathcal{E}$ of *entities* models the real-world persons whom the considered information is about.

The link between the information and its subject is captured by the *related* relation, denoted $\leftrightarrow$. This is an equivalence relation on entities, identifiers and data items, such that $o_1 \leftrightarrow o_2$ means that o_1 and o_2 are information about the same person. In particular, any identifier or data item is related to exactly one entity. Elements of the set $\mathcal{O} := \mathcal{E} \cup \mathcal{I} \cup \mathcal{D}$ are called *items of interest*.

These concepts, however, are insufficient to fully characterize the system dynamics. When interacting with a system, an actor may learn the same personal information several times without realizing that it is the same information. For example, consider two profiles of the same user that both contain “age=18”, and suppose an actor does not know that the profiles are related. Then, from a privacy point of view (e.g., to check linkability between information in profiles) it is important to differentiate in the actor’s knowledge between the age in the one profile and the age in the other profile.

In addition, an actor may be able to deduce information from the fact that different pieces of information have the same string contents. For example, if an actor encounters the same hash string in different contexts, and he knows the contents used in the first context, then he knows that these contents were also used in the second context.

2.2 Three-Layer Model

Because of the need to distinguish different instances of the same piece of information, but also to reason about message contents, we introduce a three-layer representation of personal information. The representation consists of the *object layer*, *information layer*, and *contents layer*. In the information layer, as described above, the information itself is represented, e.g., “the age of actor c”. In the object layer, information is described along with the context in which it has been observed, e.g., “the age of the data subject in instance 1 of protocol π”. In the contents layer, information is described in terms of the strings actually transmitted in a protocol, e.g., “age=18”.

In the object layer, we model the *context* in which an actor knows pieces of information. A context is a tuple (η, k), where η is a *domain* and k is a *profile* within that domain. The sets of domains and profiles depend on the application; we deliberately do not define these sets here but instead content ourselves with some examples. One example domain could be $\phi =$ Facebook, in which the context $(\phi, 132)$ represents the profile of a particular Facebook user. Another example domain is “instance 2 of protocol π”. In that domain, every party involved in the protocol is characterized by a profile.

In such a context, pieces of information are represented by *variables*. This representation makes it possible to reason about such personal information without regarding the instantiation. *Data item variables* represent data items (set D), whereas *identifier variables* represent identifiers (set I); consider, e.g., a variable $age \in \mathsf{D}$ denoting the age in a profile. A *context data item* is a data item variable d in a context (η, k), and we denote it $d|_k^\eta \in \mathsf{D}^c$; the set I^c of *context identifiers* is defined similarly. Entities are not represented by variables; instead, an entity $e \in \mathcal{E}$ in a context (η, k) is denoted $e|_k^\eta$; the set of *context entities* is $\mathcal{E}^c$. The reason is that, because entities are not digital information, there cannot be multiple "instances" of an entity. Every context contains exactly one entity who is the data subject, i.e., all information in the context belongs to that entity. $\mathcal{O}^c := \mathcal{E}^c \cup \mathsf{I}^c \cup \mathsf{D}^c$ is the set of *context items of interest*.

Items in the contents layer can be seen as strings of arbitrary length in some alphabet, i.e., the set Σ^*. The exact form of the contents layer is not relevant for our purposes. Rather, it *is* relevant to determine whether two pieces of information have the same contents: this is expressed using the τ function, as described below.

2.3 Maps between Layers and Equivalence

The link between the object layer and the information layer is given by the *substitution* $\sigma : \mathcal{O}^c \to \mathcal{O}$. We write σ as a list of context item-information pairs and application of σ in postfix notation, e.g., $\sigma = \{d|_k^\eta \to age_c, d'|_k^\eta \to haircolor_c\}$ and then $d|_k^\eta\sigma = age_c$. σ satisfies the following four properties: 1. $\sigma(\mathsf{D}^c) \subset \mathcal{D}$; 2. $\sigma(\mathsf{I}^c) \subset \mathcal{I}$; 3. $e|_k^\eta\sigma = e$ for any entity e, context (η, k); 4. $x|_k^\eta\sigma \leftrightarrow y|_k^\eta\sigma$ for any context items $x|_k^\eta$, $y|_k^\eta\sigma$. Intuitively, σ maps: 1. context data items to data items; 2. context identifiers to identifiers; 3. context entities to entities; 4. context items from the same context to related items of interest.

The link between information and its contents is given by function τ. The domain of the function is $\mathcal{I} \cup \mathcal{D}$ (entities have no contents). Function τ is injective on $\mathcal{I}$: this formally expresses the uniqueness of identifiers within the system.

We introduce notation for two context items $x|_k^\eta$, $y|_l^\chi$ representing the same information or contents. If $x|_k^\eta\sigma = y|_l^\chi\sigma$, then we write $x|_k^\eta \equiv y|_l^\chi$ and we call $x|_k^\eta$ and $y|_l^\chi$ *equivalent*. If $\tau(x|_k^\eta\sigma) = \tau(y|_l^\chi\sigma)$, then we write $x|_k^\eta \doteq y|_l^\chi$ and we call them *content equivalent*. Clearly, equivalence implies content equivalence. Two identifiers are equivalent iff they are content equivalence because of the injectivity of τ on identifiers.

Example 1. Consider the three context messages $age|_1^\eta$, $age|_1^\chi$, and $age|_1^\varsigma$ in Fig. 1 where $age \in \mathsf{D}$. Let $\sigma = \{age|_1^\eta \to age_c, age|_1^\chi \to age_c, age|_1^\varsigma \to age_d\}$ with $\tau(age_c) = \tau(age_d) =$ "age=18". Then, $age|_1^\eta$ and $age|_1^\chi$ are equivalent; moreover, all three context messages given are content equivalent. □

3 Knowledge Analysis

In this section, we analyze how personal information can be derived from the messages that a user has observed. Deductive systems are often adopted for this purpose. We present a standard deductive system, and show how it can be adapted to the three-layer model. We also show that this adaptation does not impact its expressiveness.

3.1 Messages Analysis on the Information Layer

We present a formalism of messages and a deductive system similar to the ones usually adopted in protocol analysis [13]. Standard message analysis can be seen, in terms of our three-layer model, as operating on the information layer.

$$\begin{tikzcd}
\text{object layer} & age|_1^{\eta} \arrow[dr, "\sigma"] & & age|_1^{\chi} \arrow[dl, "\sigma"'] & age|_1^{\varsigma} \arrow[d, "\sigma"] \\
\text{information layer} & & age_c \arrow[dr, "\tau"] & & age_d \arrow[dl, "\tau"] \\
\text{contents layer} & & & \text{“age=18”} &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathcal{O}^c = \mathcal{E}^c \cup \mathsf{I}^c \cup \mathsf{D}^c & \\
\mathsf{I}^c \cup \mathsf{D}^c \cup \mathcal{V}^c = \mathsf{P}^c \arrow[r, hook] \arrow[d, "\sigma"] & \mathcal{L}^c \arrow[d, "\sigma"] \\
\mathcal{O} = \mathcal{E} \cup \mathcal{I} \cup \mathcal{D} \quad \mathcal{I} \cup \mathcal{D} \cup \mathcal{G} = \mathcal{P} \arrow[r, hook] \arrow[d, "\tau"] & \mathcal{L}^i \arrow[d, "\tau"] \\
\text{(actual string)} \arrow[r, hook] & \mathcal{L}^{cnt}
\end{tikzcd}$$

Fig. 1. Example of the three-layer model: three different context items with the information and contents they represent (left); the three-layer model of information (right)

Messages. The basic components of messages in communication protocols are *information items*. Apart from the sets $\mathcal{D}$ of data items and $\mathcal{I}$ of identifiers, we also consider a set $\mathcal{G}$ of non-personal information, such as shared keys and nonces. The set of information items is denoted $\mathcal{P} := \mathcal{D} \cup \mathcal{I} \cup \mathcal{G}$. Private and public keys are particular cases of identifiers. Private keys form a set $\mathcal{K}^- \subset \mathcal{I}$, public keys form a set $\mathcal{K}^+ \subset \mathcal{I}$, and, given a private key k^-, the corresponding public key is k^+ and vice versa.

Messages can be constructed from information items using cryptographic primitives. The set of *information messages*, denoted $\mathcal{L}^i$, is given by the following grammar:

$$M, N ::= p \mid E_{k^+}(M) \mid E'_N(M) \mid S_{k^-}(M) \mid \mathcal{H}(M) \mid \{M, N\} \qquad (1)$$

where $p \in \mathcal{P}$, $k^+ \in \mathcal{K}^+$, $k^- \in \mathcal{K}^-$. This models, respectively: asymmetric encryption $E_{k^+}(M)$ of message M with public key k^+, symmetric encryption $E'_N(M)$ of message M with key N, signature $S_{k^-}(M)$ over message M with private key k^-, hash $\mathcal{H}(M)$ of message M, and (associative) concatenation $\{M, N\}$ of messages M and N.

We assume that these cryptographic primitives satisfy a number of properties. First, all primitives are deterministic; that is, given the same inputs, they always give the same output. Randomness in non-deterministic encryption or signing should be modeled explicitly as part of the plaintext. By signing we mean "clear-signing" [14]; that is, the message M can be recovered from $S_{k^-}(M)$ without knowledge of the corresponding public key k^+. (This can be achieved by appending the message to the "raw" signature.)

Finally, we assume *structural equivalence*. Extend τ from $\mathcal{P}$ to $\mathcal{L}^i$ by applying it to all information items in a message, e.g.: $\tau(E'_d(d')) = E'_{\tau(d)}(\tau(d'))$. The image $\tau(\mathcal{L}^i)$ is the language $\mathcal{L}^{cnt}$ generated by grammar (1) with contents instead of information items. Different elements of $\mathcal{L}^{cnt}$ could a priori be the same as strings, e.g. a collision in the hash function could cause $\mathcal{H}(\tau(x))$ and $\mathcal{H}(\tau(y))$ to be the same string even if $\tau(x) \neq \tau(y)$; or $E'_{\tau(x)}(\tau(y))$ could happen to be the same string as $\mathcal{H}(\tau(z))$. Structural equivalence is the assumption that this does not happen, i.e., the grammar $\mathcal{L}^{cnt}$ uniquely represents message contents. As a map from $\mathcal{L}^i$ to $\mathcal{L}^{cnt}$, τ satisfies two properties: 1. τ is injective on $\mathcal{I}$; 2. τ preserves the grammar structure of information messages.

Deductive System. A deductive system on $\mathcal{L}^i$ models which information messages $m \in \mathcal{L}^i$ an actor can *deduce* from the set $\mathcal{C}^i_a \subset \mathcal{L}^i$ of messages he knows (denoted $\mathcal{C}^i_a \vDash m$). Such a deductive system consists of a set of axioms and inference rules that mimic the idealized operation of the cryptographic primitives [13].

Axiom **(⊨0)** $$\frac{}{\mathcal{C}_a^i \vDash m}\,(m \in \mathcal{C}_a^i)\ \textbf{(}\vDash\textbf{0)}$$

Construction **(⊨C*)** $$\frac{\mathcal{C}_a^i \vDash m \quad \mathcal{C}_a^i \vDash n}{\mathcal{C}_a^i \vDash \{m,n\}}\ \textbf{(}\vDash\textbf{CC)}$$

$$\frac{\mathcal{C}_a^i \vDash m}{\mathcal{C}_a^i \vDash \mathcal{H}(m)}\ \textbf{(}\vDash\textbf{CH)} \qquad \frac{\mathcal{C}_a^i \vDash m \quad \mathcal{C}_a^i \vDash n}{\mathcal{C}_a^i \vDash E'_n(m)}\ \textbf{(}\vDash\textbf{CE)} \qquad \frac{\mathcal{C}_a^i \vDash m \quad \mathcal{C}_a^i \vDash k^+}{\mathcal{C}_a^i \vDash E_{k^+}(m)}\ \textbf{(}\vDash\textbf{CA)}$$

$$\frac{\mathcal{C}_a^i \vDash m \quad \mathcal{C}_a^i \vDash k^-}{\mathcal{C}_a^i \vDash S_{k^-}(m)}\ \textbf{(}\vDash\textbf{CS)}$$

Elimination **(⊨E)** $$\frac{\mathcal{C}_a^i \vDash \{m,n\}}{\mathcal{C}_a^i \vDash m}\ \textbf{(}\vDash\textbf{EC)} \qquad \frac{\mathcal{C}_a^i \vDash \{m,n\}}{\mathcal{C}_a^i \vDash n}\ \textbf{(}\vDash\textbf{EC')}$$

$$\frac{\mathcal{C}_a^i \vDash E'_n(m) \quad \mathcal{C}_a^i \vDash n}{\mathcal{C}_a^i \vDash m}\ \textbf{(}\vDash\textbf{EE)} \qquad \frac{\mathcal{C}_a^i \vDash E_{k^+}(m) \quad \mathcal{C}_a^i \vDash k^-}{\mathcal{C}_a^i \vDash m}\ \textbf{(}\vDash\textbf{EA)} \qquad \frac{\mathcal{C}_a^i \vDash S_{k^-}(m)}{\mathcal{C}_a^i \vDash m}\ \textbf{(}\vDash\textbf{ES)}$$

Fig. 2. Deductive system on information ($\mathcal{C}_a^i \subset \mathcal{L}^i$, $m, n \in \mathcal{L}^i$, $k^+ \in \mathcal{K}^+$, $k^- \in \mathcal{K}^-$)

Fig. 2 shows a standard deductive system for information messages. The **(⊨0)** axiom expresses the deduction of any message in the set of known messages. The **(⊨C*)** inference rules express the construction of concatenations, hashes, symmetric encryptions, asymmetric encryptions and signatures of deduced messages. The **(⊨E*)** inference rules express decomposition of concatenations, decryption of symmetric and asymmetric encryptions whose key is known, and recovery of the plaintext from a signed message.

In the case of decryption, note that the deductive system does not express *how* the actor knows the decryption key, only *that* he knows it. Thus, an actor can try out any key to decrypt a message; if it happens to be the correct one, he obtains the plaintext. This means that the system over-estimates the knowledge of the actor in case he cannot actually tell by decrypting whether he used the right key or not, e.g. if the plaintext is something that is unknown, random, and unformatted such as a nonce.

Other properties of idealized cryptographic primitives are expressed by the absence of additional inference rules: e.g., one-wayness of hashes is accounted for by the absence of a rule to deduce m from $\mathcal{H}(m)$. In addition, note that there is no signature verification rule. This is because deductive systems focus on making deductions from known messages rather than checking message validity.

The *deduction* of a message using these rules is usually denoted in tree form. For example, we represent a deduction of age_c from $\mathcal{C}_a^i = \{E'_{key}(age_c), key\}$ as follows:

$$\frac{\dfrac{}{\mathcal{C}_a^i \vDash E'_{key}(age_c)}\,\textbf{(}\vDash\textbf{0)} \qquad \dfrac{}{\mathcal{C}_a^i \vDash key}\,\textbf{(}\vDash\textbf{0)}}{\mathcal{C}_a^i \vDash age_c}\ \textbf{(}\vDash\textbf{EE)}$$

3.2 Message Analysis on the Object Layer

The deductive system above models the actor's knowledge on the information layer. However, for privacy analysis we need to distinguish between information from various contexts and reason about message contents. To achieve this, we adjust the deductive system to work on the object layer.

Messages. We define the set P of *context items* at the object layer analogously to the set $\mathcal{P}$ of information items. That is, $\mathsf{P} := \mathsf{D}^c \cup \mathsf{I}^c \cup \mathsf{V}^c$, with D^c and I^c the sets of context data items and identifiers. Similarly, V^c is the set of *context global items*, which can represent any information message, in particular items in $\mathcal{G}$. Context global items belong to a domain, but not to a profile; an example context global item is $shakey|^{\eta}_{\cdot}$.

The set $\mathcal{L}^c$ of *context messages* is generated by grammar (1), except that here p is any context item, and $k^+ \in \mathsf{K}^{+c} \subset \mathsf{I}^c$ and $k^- \in \mathsf{K}^{-c} \subset \mathsf{I}^c$ are context identifiers representing public and private keys, respectively. Notationally, contexts, domains, and profiles can be applied to messages, indicating application to all context items in the message, e.g., $E_{shakey|_{\cdot}}(age|_1)|^{\eta} := E_{shakey|^{\eta}_{\cdot}}(age|^{\eta}_1)$ and $\{id, age\}|^{\eta}_1 := \{id|^{\eta}_1, age|^{\eta}_1\}$.

The substitution σ extends from context items of interest to context messages in a natural way, e.g.: $\{\mathsf{m}_1, \mathsf{m}_2\}\sigma := \{\mathsf{m}_1\sigma, \mathsf{m}_2\sigma\}$. As a map from $\mathcal{L}^c$ to $\mathcal{L}^i$, σ satisfies properties 1–4 discussed in Section 2 as well as two additional properties: 5. $\sigma(\mathsf{K}^{+c}) \subset \mathcal{K}^+$, $\sigma(\mathsf{K}^{-c}) \subset \mathcal{K}^-$, and $key^+|^{\eta}_k\sigma = k^+$ iff $key^-|^{\eta}_k\sigma = k^-$ where k^- and k^+ are a private/public key pair; 6. σ preserves the grammar structure of context messages.

Sets $\mathcal{L}^c$, $\mathcal{L}^i$, and $\mathcal{L}^{cnt}$ and functions σ, τ form a three-layer model of messages that extends the personal information model (Fig. 1, right). Like context items, context messages m and n are *equivalent* iff $\mathsf{m}\sigma = \mathsf{n}\sigma$, and *content equivalent* iff $\tau(\mathsf{m}\sigma) = \tau(\mathsf{n}\sigma)$.

Deductive System. To perform deduction on $\mathcal{L}^c$, we translate the inference rules on $\mathcal{L}^i$ to $\mathcal{L}^c$, but this is insufficient for two main reasons. First, although the object layer distinguishes between keys used in different contexts, an actor can re-use a key from one context in another. Second, an actor may infer additional information from the fact that different context messages have the same contents. We address the first problem with "key testing" rules, and the second with a "content analysis" rule.

The deductive system on the object layer (Fig. 3) models which context messages m an actor a can deduce from his known messages $\mathcal{C}_a \subset \mathcal{L}^c$ ($\mathcal{C}_a \vdash \mathsf{m}$). The rules **($\vdash$0)** to **($\vdash$EA)** are direct translations from the rules **($\vDash$0)** to **($\vDash$EA)** on the information layer. We now describe the additional object layer rules.

Key testing accounts for an actor knowing the key for decryption or signature verification of a message m, but not in the message's context. Note that in this case, e.g., decryption rule **($\vdash$EE)** can not be used directly. The key testing rules allow an actor, as in the deductive system on information, to try out on m any key he knows. If he uses a key with the correct contents, then he learns that it is the decryption **($\vdash$TA)**, **($\vdash$TE)** or signature verification **($\vdash$TS)** key. (Then, he can decrypt using **($\vdash$EE)** or **($\vdash$EA)**.) Note that in an implementation, to decide whether **($\vdash$T*)** can be applied, we only need to check the existence of a derivable content equivalent key, regardless of its context layer representation. For this, standard deduction techniques at the content layer suffice.

Example 2. Let $\mathcal{C}_a = \{E'_k(goods)|^{\pi}_{\cdot}, l|^{\rho}_{\cdot}\}$ be the set of messages known by an actor a, with $k|^{\pi}_{\cdot} \doteq l|^{\rho}_{\cdot}$. Then $\mathcal{C}_a \vdash goods|^{\pi}_{\cdot}$ can be deduced as follows:

$$\dfrac{\dfrac{}{\mathcal{C}_a \vdash E'_k(goods)|^{\pi}_{\cdot}}\,(\vdash\mathbf{0}) \qquad \dfrac{\dfrac{}{\mathcal{C}_a \vdash E'_k(goods)|^{\pi}_{\cdot}}\,(\vdash\mathbf{0}) \quad \dfrac{}{\mathcal{C}_a \vdash l|^{\rho}_{\cdot}}\,(\vdash\mathbf{0})}{\mathcal{C}_a \vdash k|^{\pi}_{\cdot}}\,(\vdash\mathbf{TE})}{\mathcal{C}_a \vdash goods|^{\pi}_{\cdot}}\,(\vdash\mathbf{DE})$$

Axiom (⊢0) $\dfrac{}{\mathcal{C}_a \vdash \mathsf{m}}\ (\mathsf{m} \in \mathcal{C}_a)$ **(⊢0)** **Construction (⊢C*)** $\dfrac{\mathcal{C}_a \vdash \mathsf{m} \quad \mathcal{C}_a \vdash \mathsf{n}}{\mathcal{C}_a \vdash \{\mathsf{m},\mathsf{n}\}}$ **(⊢CC)**

$\dfrac{\mathcal{C}_a \vdash \mathsf{m}}{\mathcal{C}_a \vdash \mathcal{H}(\mathsf{m})}$ **(⊢CH)** $\dfrac{\mathcal{C}_a \vdash \mathsf{m} \quad \mathcal{C}_a \vdash \mathsf{n}}{\mathcal{C}_a \vdash E'_{\mathsf{n}}(\mathsf{m})}$ **(⊢CE)** $\dfrac{\mathcal{C}_a \vdash \mathsf{m} \quad \mathcal{C}_a \vdash \mathsf{k}^+}{\mathcal{C}_a \vdash E_{\mathsf{k}^+}(\mathsf{m})}$ **(⊢CA)**

$\dfrac{\mathcal{C}_a \vdash \mathsf{m} \quad \mathcal{C}_a \vdash \mathsf{k}^-}{\mathcal{C}_a \vdash S_{\mathsf{k}^-}(\mathsf{m})}$ **(⊢CS)** **Elimination (⊢E*)** $\dfrac{\mathcal{C}_a \vdash \{\mathsf{m},\mathsf{n}\}}{\mathcal{C}_a \vdash \mathsf{m}}$ **(⊢EC)** $\dfrac{\mathcal{C}_a \vdash \{\mathsf{m},\mathsf{n}\}}{\mathcal{C}_a \vdash \mathsf{n}}$ **(⊢EC')**

$\dfrac{\mathcal{C}_a \vdash E'_{\mathsf{n}}(\mathsf{m}) \quad \mathcal{C}_a \vdash \mathsf{n}}{\mathcal{C}_a \vdash \mathsf{m}}$ **(⊢EE)** $\dfrac{\mathcal{C}_a \vdash S_{\mathsf{k}^-}(\mathsf{m})}{\mathcal{C}_a \vdash \mathsf{m}}$ **(⊢ES)** $\dfrac{\mathcal{C}_a \vdash E_{\mathsf{k}^+}(\mathsf{m}) \quad \mathcal{C}_a \vdash \mathsf{k}^-}{\mathcal{C}_a \vdash \mathsf{m}}$ **(⊢EA)**

Key testing (⊢T*) $\dfrac{\mathcal{C}_a \vdash E_{\mathsf{k}^+}(\mathsf{m}) \quad \mathcal{C}_a \vdash \mathsf{k}'^-}{\mathcal{C}_a \vdash \mathsf{k}^-}\ (\mathsf{k}^- \doteq \mathsf{k}'^-)$ **(⊢TA)**

$\dfrac{\mathcal{C}_a \vdash S_{\mathsf{k}^-}(\mathsf{m}) \quad \mathcal{C}_a \vdash \mathsf{k}'^+}{\mathcal{C}_a \vdash \mathsf{k}^+}\ (\mathsf{k}^+ \doteq \mathsf{k}'^+)$ **(⊢TS)** $\dfrac{\mathcal{C}_a \vdash E'_{\mathsf{n}}(\mathsf{m}) \quad \mathcal{C}_a \vdash \mathsf{n}'}{\mathcal{C}_a \vdash \mathsf{n}}\ (\mathsf{n} \doteq \mathsf{n}')$ **(⊢TE)**

Content analysis (⊢C) $\dfrac{\mathcal{C}_a \vdash \mathsf{m}_1 \quad \mathcal{C}_a \vdash \mathsf{m}_2 \quad \mathcal{C}_a \vdash \mathsf{n}_1}{\mathcal{C}_a \vdash \mathsf{n}_2}\ \begin{array}{l}((\mathsf{m}_1 \doteq \mathsf{m}_2) \Rightarrow (\mathsf{m}_3 \doteq \mathsf{m}_4);\\ \mathsf{n}_1 =_{\mathsf{m}_3 \sim \mathsf{m}_4} \mathsf{n}_2)\end{array}$ **(⊢C)**

Fig. 3. Deductive system on context messages ($\mathcal{C}_a$ a set of context messages, m, m_i, n, n_i context messages; $\mathsf{k}^+/\mathsf{k}^-$ and $\mathsf{k}'^+/\mathsf{k}'^-$ public/private key pairs, $\Rightarrow$ as in Def. 2, $\mathsf{n}_1 =_{\mathsf{m}_3 \sim \mathsf{m}_4} \mathsf{n}_2$ means n_1 and n_2 are equal up to replacing m_3 by m_4 and vice versa)

The deduction models an actor testing whether $l|^{\rho}_{\cdot}$ is the decryption key for $E'_k(goods)|^{\pi}_{\cdot}$ (⊢**TE**). By learning it, the actor can decrypt the message (⊢**DE**). □

Content analysis lets an actor derive an unknown message from one context by concluding that it has the same contents as a known message from another. The statement of the rule relies on the syntactic structure of messages, which we first elaborate on.

The syntactic structure of messages describes the way they are constructed using cryptographic primitives. Primitives build up a message m given two (or, in the case of the hash, one) messages n and n': we define one to be the "left part" $\mathsf{n} = \mathsf{m}@l$ and the other to be the "right part" $\mathsf{n}' = \mathsf{m}@r$. Recursively, every submessage of m has a well-defined "position" in m:

Definition 1. *Let* m *be a context message and* $z \in \{l, r\}^*$*. Then,* $\mathsf{m}@z$*, the* submessage *of* m *at* z*, is defined as follows:* $\mathcal{H}(\mathsf{m})@l = \mathsf{m}$; $\{\mathsf{m},\mathsf{n}\}@l = \mathsf{m}$; $\{\mathsf{m},\mathsf{n}\}@r = \mathsf{n}$; $E'_{\mathsf{n}}(\mathsf{m})@l = \mathsf{n}$; $E'_{\mathsf{n}}(\mathsf{m})@r = \mathsf{m}$; $E_{\mathsf{k}^+}(\mathsf{m})@r = \mathsf{k}^+$; $E_{\mathsf{k}^+}(\mathsf{m})@r = \mathsf{m}$; $S_{\mathsf{k}^-}(\mathsf{m})@r = \mathsf{k}^-$; $S_{\mathsf{k}^-}(\mathsf{m})@r = \mathsf{m}$; $\mathsf{m}@z_1 \ldots z_n = ((\mathsf{m}@z_1)@\ldots)@z_n$.

Note that for arbitrary context message m and $z \in \{l, r\}^*$, $\mathsf{m}@z$ may not be defined. For instance, $\mathcal{H}(x)@l$ is defined (and equal to x), but $\mathcal{H}(x)@r$ is not.

If two context messages m_1 and m_2 are content equivalent, then the properties of σ and τ imply content equivalence of their submessages. In particular, if $\mathsf{m}_1@z$ and $\mathsf{m}_2@z$ are both defined, then they are content equivalent. Also, if $\mathsf{m}_1@z = \mathsf{k}^+$ and $\mathsf{m}_2@z = \mathsf{k}'^+$, then not only $\mathsf{k}^+ \doteq \mathsf{k}'^+$ follows, but also $\mathsf{k}^- \doteq \mathsf{k}'^-$, and vice versa. The following notation expresses this intuition:

Definition 2. *Let* $\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}_3, \mathsf{m}_4$ *be context messages. We write* $(\mathsf{m}_1 \doteq \mathsf{m}_2) \Rightarrow (\mathsf{m}_3 \doteq \mathsf{m}_4)$*, if* $\mathsf{m}_1 \doteq \mathsf{m}_2$ *and for some* $z \in \{l, r\}^*$*:*

- $\mathsf{m}_3 = \mathsf{m}_1@z$, $\mathsf{m}_4 = \mathsf{m}_2@z$*; or*
- $\mathsf{m}_1@z$, $\mathsf{m}_2@z$ *represent public keys of which* m_3, m_4 *are the private keys; or*
- $\mathsf{m}_1@z$, $\mathsf{m}_2@z$ *represent private keys of which* m_3, m_4 *are the public keys.*

The "content analysis" inference rule **(⊢C)** then states that if an actor can derive m_1 and m_2 such that $(\mathsf{m}_1 \doteq \mathsf{m}_2) \Rightarrow (\mathsf{m}_3 \doteq \mathsf{m}_4)$, and he can derive a message with m_3 in it, he can also derive the message with m_3 replaced by m_4, and vice versa.

Example 3. Let $\mathcal{C}_a = \{\mathcal{H}(id, age)|_1^\eta, id|_2^\eta, age|_3^\eta\}$ be the set of messages known by actor a with $id \in \mathsf{I}$, $age \in \mathsf{D}$ such that $id|_1^\eta \doteq id|_2^\eta$ and $age|_1^\eta \doteq age|_3^\eta$. $\mathcal{C}_a \vdash \mathcal{H}(id, age)|_1^\eta$ holds, and by **(⊢CC)**, **(⊢CH)** we have $\mathcal{C}_a \vdash \mathcal{H}(id|_2^\eta, age|_3^\eta)$. From this, a knows that $id|_1^\eta \doteq id|_2^\eta$ (as well as $age|_1^\eta \doteq age|_3^\eta$). By **(⊢C)** he can then deduce $id|_1^\eta$:

$$\dfrac{\dfrac{}{\mathcal{C}_a \vdash \mathcal{H}(id,age)|_1^\eta}\,(\vdash\mathbf{0}) \qquad \dfrac{\cdots}{\mathcal{C}_a \vdash \mathcal{H}(id|_2^\eta,age|_3^\eta)}\,(\vdash\mathbf{CH}) \qquad \dfrac{}{\mathcal{C}_a \vdash id|_2^\eta}\,(\vdash\mathbf{0})}{\mathcal{C}_a \vdash id|_1^\eta}\,(\vdash\mathbf{C})$$

In the same way also $\mathcal{C}_a \vdash age|_1^\eta$ follows. □

There are two notable consequences of content analysis. First, if an actor knows a public/private key pair in one context (ζ, k) and just the public key in another context (η, l) then he can deduce the private key in (η, l). Second, an actor can link different profiles of the same entity if he sees that the profiles share an identifier (see §4).

The feasibility of implementing the content analysis rule follows from two observations. First, we can safely assume that content analysis rules are the final steps (from leaf to root) in a deduction tree, and that messages $\mathsf{m}_1, \mathsf{m}_2$ in **(⊢C)** have been deduced without content analysis. Second, in **(⊢C)**, $\mathsf{n}_1 \doteq \mathsf{n}_2$ holds. Thus, to decide whther a given message n_2 can be derived, one can first derive without using **(⊢C)** all messages n_1 content equivalent to n_2, and then verify whether any n_1 can be transformed step-by-step to n_2 using **(⊢C)**.

3.3 Deduction on Object vs Information Layer

Given context messages $\mathcal{C}_a$, one can perform object layer deduction and then apply σ to the result; or one can first apply σ to $\mathcal{C}_a$ and then perform information layer deduction. One proves easily that the first approach gives at least as much information as the second, i.e., object layer deduction is at least as expressive as information layer deduction:

Proposition 1. *Let* $\mathcal{C}_a \subset \mathcal{L}^c$*. Define* $\overline{\mathcal{C}_a}\sigma := \{x\sigma \mid \mathcal{C}_a \vdash x\}$*;* $\overline{\mathcal{C}_a^i} := \{x \mid \sigma(\mathcal{C}_a) \vDash x\}$*. Then,* $\overline{\mathcal{C}_a^i} \subset \overline{\mathcal{C}_a}\sigma$*. Conversely,* $\overline{\mathcal{C}_a^i} \supset \overline{\mathcal{C}_a}\sigma$ *holds for all* $\mathcal{C}_a$ *iff* τ *is injective on* $\mathcal{L}^i$*.*

Note that object layer deduction is strictly more expressive than information layer deduction when τ is not injective, i.e., when different pieces of information have the same contents. This condition reflects a significant difference between IdM and other settings: in IdM, it is likely to come across different pieces of information with the same contents, whereas in other settings the kind of information that is usually considered – nonces, keys, random values, etc. – can for the purposes of analysis be safely assumed to have unique contents.

4 Knowledge of Personal Information

In this section we define the *view* of an actor a, capturing his knowledge about personal information. There are two aspects to this knowledge. First, what information the actor knows, formalized by the set $\mathcal{O}^c_a \subset \mathcal{O}^c$ of *detectable* context items. Second, which context items he knows to represent information about the same entity, formalized by the $\leftrightarrow_a$ equivalence relation on $\mathcal{O}^c$ defining context items *associable* to each other.

An actor's view follows from his sets $\mathcal{C}_a \subset \mathcal{L}^c$, $\mathcal{E}^c_a \subset \mathcal{E}^c$ of known context messages and entities. Associations between context items follow from properties of both σ and τ. First, context items in one context are related, and so is the same entity in different contexts (properties 3, 4 of σ). Second, context identifiers with equal contents are equal (property 1 of τ). Thus, define $\leftrightarrow_a$ as the minimal equivalence relation on $\mathcal{O}^c$ such that:

– For all $e|^\eta_k, e|^\zeta_l \in \mathcal{E}^c$: $e|^\eta_k \leftrightarrow_a e|^\zeta_l$; for all $x|^\eta_k, y|^\eta_k \in \mathcal{O}^c$: $x|^\eta_k \leftrightarrow_a y|^\eta_k$
– If $\mathcal{C}_a \vdash \mathsf{m}_1, \mathcal{C}_a \vdash \mathsf{m}_2$, and $(\mathsf{m}_1 \doteq \mathsf{m}_2) \Rightarrow (\mathsf{i}_1 \doteq \mathsf{i}_2)$ for $\mathsf{i}_1, \mathsf{i}_2 \in \mathsf{I}^c$, then $\mathsf{i}_1 \leftrightarrow_a \mathsf{i}_2$.

Detectability of items follows from our deductive system: $\mathcal{O}^c_a = \mathcal{E}^c_a \cup \mathsf{I}^c_a \cup \mathsf{D}^c_a$, where $\mathsf{D}^c_a = \{\mathsf{d} \in \mathsf{D}^c \mid \mathcal{C}_a \vdash \mathsf{d}\}$ and $\mathsf{I}^c_a = \{\mathsf{i} \in \mathsf{I}^c \mid \mathcal{C}_a \vdash \mathsf{i}\}$. One may expect that $e|^\eta_k \in \mathcal{E}^c_a$ and $e|^\eta_k \leftrightarrow_a i|^\chi_l$ imply $e|^\chi_l \in \mathcal{E}^c_a$, but, as can be seen later, we do need such a rule to define the view as $e|^\eta_k$ and $e|^\eta_k$ will be known by the actor to be equivalent anyway.

Note that actors may associate items which they can not detect. In fact, because of transitivity of $\leftrightarrow_a$, an actor knowing a relation between items he can not detect may help him to establish a relation between items he can detect:

Example 4. Consider a set $\mathcal{C}_a = \{\{E_{shakey|.}(id|_1), d|_1\}|^\eta, \{E_{shakey|.}(id|_1), d'|_1\}|^\chi\}$ of messages known by actor a, where $E_{shakey|.}(id|_1)|^\eta \doteq E_{shakey|.}(id|_1)|^\chi$. Then, $id|^\eta_1 \leftrightarrow_a id|^\chi_1$ by condition 2 for $\leftrightarrow_a$ (even though the actor can detect neither context identifier). By condition 1 for $\leftrightarrow_a$ and transitivity, $d|^\eta_1 \leftrightarrow_a d'|^\chi_1$ follows. □

We simplify the representation of an actor's knowledge by considering his *known equivalences* $\equiv_a$, defined as follows: $x \equiv_a y$ if $x, y \in \mathcal{O}^c_a$, $x \equiv y$ and $x \leftrightarrow_a y$.

Definition 3. *Let a be an actor with set of known context messages $\mathcal{C}_a$. Then, a's* view *is the structure $M^c_a = (\mathcal{E}^c_a/\equiv_a, \mathsf{I}^c_a/\equiv_a, \mathsf{D}^c_a/\equiv_a, \leftrightarrow_a/\equiv_a)$ with $\leftrightarrow_a/\equiv_a$ the canonical equivalence relation on $\mathcal{O}^c_a/\equiv_a$.*

5 Defining and Verifying Identity-Related Properties

In this section, we recap the identity-related properties defined in our previous work [12], adapted to the three-layer model (see Table 1). Identity-related properties with respect to an actor can be seen either as properties of a data item (i.e., on the information layer), or of a context data item representing that data item (i.e., on the object layer). For instance, anonymity of a context data item d with respect to an actor a means that d is not associable by a to a context entity. However, there might be another, equivalent, context data item that *can* be associated by a to a context entity.

We define identity-related properties for a data item d by considering the privacy properties holding for all context data items that represent it; for example, d is anonymous if all its representations are. Note that for complete identifiability of a data item d,

Table 1. Identity-related properties with respect to actor a, defined for a context data item d (middle column) and for a data item d (right column), where $[d] := \{\mathsf{d} \in \mathsf{D}^c \mid \mathsf{d}\sigma = d\}$

Property	Condition on $\mathsf{d} \in \mathsf{D}^c$	Condition on $d \in \mathcal{D}$
detectability (D)	$\mathsf{d} \in \mathsf{D}_a^c$	$\exists \mathsf{d} \in [d]$: d is D
undetectability (UD)	$\mathsf{d} \notin \mathsf{D}_a^c$	$\forall \mathsf{d} \in [d]$: d is UD
identifiability (I)	$\exists \mathsf{e} \in \mathcal{E}_a^c$ s.t. $\mathsf{d} \leftrightarrow_a^c \mathsf{e}$	$\exists \mathsf{d} \in [d]$: d is I
pseudo-identifiability (PI)	$\exists \mathsf{i} \in \mathsf{I}_a^c$ s.t. $\mathsf{d} \leftrightarrow_a^c \mathsf{i}$	$\exists \mathsf{d} \in [d]$: d is PI
complete identifiability (CI)	$\exists \mathsf{e} \in \mathcal{E}_a^c, \mathsf{i} \in \mathsf{I}_a^c$ s.t. $\mathsf{d} \leftrightarrow_a^c \mathsf{e} \wedge \mathsf{d} \leftrightarrow_a^c \mathsf{i}$	$\exists \mathsf{d} \in [d]$: d is CI
anonymity (A)	$\mathsf{d} \notin \mathsf{D}_a^c$, or $\forall \mathsf{e} \in \mathcal{E}_a^c : \mathsf{d} \not\leftrightarrow {}_a^c \mathsf{e}$	$\forall \mathsf{d} \in [d]$: d is A
pseudonymity (PA)	$\forall \mathsf{e} \in \mathcal{E}_a^c$ $\mathsf{d} \not\leftrightarrow {}_a^c \mathsf{e}$ and $\exists \mathsf{i} \in \mathsf{I}_a^c$ s.t. $\mathsf{d} \leftrightarrow_a^c \mathsf{i}$	$\forall \mathsf{d} \in [d]$: d is A $\wedge$ $\exists \mathsf{d} \in [d]$: d is PI
complete anonymity (CA)	$\mathsf{d} \notin \mathsf{D}_a^c$, or $\forall \mathsf{e} \in \mathcal{E}_a^c$ $\mathsf{d} \not\leftrightarrow {}_a^c \mathsf{e}$ and $\forall \mathsf{i} \in \mathsf{I}_a^c$ $d \not\leftrightarrow {}_a^c \mathsf{i}$	$\forall \mathsf{d} \in [d]$: d is A $\wedge d$ is not PI
linkability (L) (to d'/d')	$\mathsf{d} \leftrightarrow_a \mathsf{d}'$	$\exists \mathsf{d}' \in [d'] : \mathsf{d} \leftrightarrow_a \mathsf{d}'$
linkability (UL) (to d'/d')	$\mathsf{d} \not\leftrightarrow_a \mathsf{d}'$	$\nexists \mathsf{d}' \in [d'] : \mathsf{d} \leftrightarrow_a \mathsf{d}'$

we require that the *same* representation of d is both identifiable and pseudo-identifiable; the other properties are obvious. The method developed in the previous sections then allows one to verify identity-related properties in the following three steps:

- **Step 1**: Using the deductive system, determine the detectable context items.
- **Step 2**: Determine associable context items, and thus the actor view.
- **Step 3**: From the actor view, check which properties are satisfied.

6 Case Study: TAS^3 Attribute Aggregation

In this section, we demonstrate our approach by analyzing the TAS^3 attribute aggregation infrastructure [16]. We demonstrate how our approach can be used to verify whether the privacy properties for which it was designed do indeed hold. To be able to check for linkability between different executions, we analyze two executions of the protocol involving the same actors. The results also hold for more than two executions. The analysis leads to some recommendations for improvements to the system.

6.1 TAS^3 Attribute Aggregation

The TAS^3 project (`http://tas3.eu`) is a research project aiming to create an architecture for on-line services based on personal information. Here we focus on the TAS^3 attribute aggregation infrastructure, in which a service provider (SP) collects from different identity providers (IdPs) personal information about a user requesting a service. A main feature of the infrastructure is the linking service (LS), which links the different identifiers of the user at different IdPs, alleviating the need for global user identifiers.

The attribute aggregation infrastructure is described at high level in [2,15,16], and the concrete message formats are described in [17]. These message formats are based on open standards: notably, SAML 2.0 [18] and Liberty ID-WSF 2.0 [19].

The infrastructure aims to guarantee a number of privacy properties [2,16]. First, the SP wants "strong cryptographic evidence that each of the [attributes] does belong to the user who has initiated the session" (*P1*). Second, "none of the user's [IdPs should] know about any of the user's other ones" (*P2*). Third, "the [LS should] not know who the user is, or what identity attributes [he has]" (*P3*). Finally, "the [SP should not be able to] relate visits of the user together" (*P4*).

In our case study, we consider one user with attributes at two different identity providers (IdP1 and IdP2), who wishes to access a service from one SP twice. Thus, the same attribute aggregation process takes place twice. The process begins after IdP1 has authenticated the user. IdP1 informs the SP that the user has been authenticated, provides the SP with the value of the user's attribute at IdP1, and refers the SP to the LS. The SP contacts the LS, who refers him to IdP2. Finally, the SP requests and receives the value of the user's attribute at IdP2.

6.2 Formalization

Our formalization of attribute aggregation is depicted in Fig. 4. Fig. 4(a) shows the messages exchanged in an instance of the attribute aggregation protocol. Fig. 4(b) shows the information layer. The user has profiles at IdP1 and IdP2 consisting of one attribute and one identifier. Also, the LS shares an identifier of the user with each of the two IdPs. Each communicating party (SP, LS, IdP1, IdP2) has a private/public key pair and a public identifier. Finally, the protocol instances use four nonces in total.

Fig. 4(c) displays the actors' knowledge in the object layer before attribute aggregation. The LS, IdP1 and IdP2 know the aforementioned information about the user in a context corresponding to some entry in their respective databases: say $|^{\lambda}_{21}$, $|^{\iota}_{7}$, and $|^{\pi}_{3}$. They also know the public keys and identifiers of the other actors, and their own secret key, in contexts corresponding to their roles in the system: $|^{\pi}_{\mathrm{SP}}$, $|^{\pi}_{\mathrm{LS}}$, $|^{\pi}_{\mathrm{IdP1}}$, and $|^{\pi}_{\mathrm{IdP2}}$. The map σ linking these context items to information is straightforward. Fig. 4(d) shows the messages known by each actor after two instances of attribute aggregation. We assume that each actor learns only the messages that he sent or received.

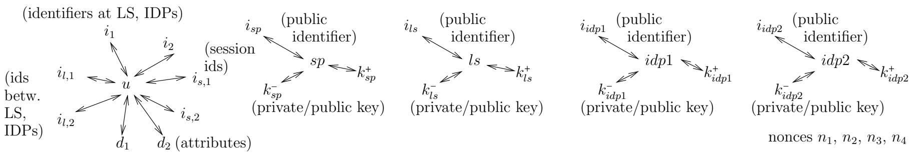
Finally, Fig. 4(e) formalizes the identity-related properties we previously introduced informally. Note that P2 and P3 are at the information layer, whereas P1 and P4, being about linking copies from different contexts, are at the object layer.

For the construction of our model, the high-level protocol descriptions from [2,16] were detailed enough. However, some lower-level aspects can be considered for extended analysis. First, the communication channels used in the protocol are all encrypted, which one could explicitly model. Second, in an implementation, the role of IdP2 would be performed by two logically different parties: a "discovery service" and an "attribute authority", so the communication would be more complex than we sketched. Third, the actual messages may contain information such as timestamps that ensure message portions from one context can not be re-used in another context.

Our formalization of the properties of the architecture differs slightly from their natural language descriptions in two ways. First, P1 mentions "strong cryptographic proof", suggesting that it is interesting to model the assurance an actor has about correctness of information he learns. In our previous work [12], we introduced notions of "provability" and "deniability" that could be used, but the present analysis method does not cover them. Second, P2 can have a stricter interpretation; that is, IdP1 should not even know whether or not the user has a profile at IdP2, and vice versa. However, in [15],

$$\begin{array}{rl}
IDP_1 \rightarrow SP & m_1 = S_{k^-|_{\mathrm{IDP}_1}}(i_{sess}|_{\mathrm{U}}, d_{idp1}|_{\mathrm{U}}, \{i|_{\mathrm{LS}}, E_{k^+|_{\mathrm{LS}}}(i_{idp1,ls}|_{\mathrm{U}}, n|_{.})\}) \\
SP \rightarrow LS & m_2 = \{E_{k^+|_{\mathrm{LS}}}(i_{idp1,ls}|_{\mathrm{U}}, n|_{.})\}), m_1\} \\
LS \rightarrow SP & m_3 = \{i|_{\mathrm{IDP}_2}, E_{k^+|_{\mathrm{IDP}_2}}(i_{idp2,ls}|_{\mathrm{U}}, n'|_{.})\} \\
SP \rightarrow IDP_2 & m_4 = \{E_{k^+|_{\mathrm{IDP}_2}}(i_{idp2,ls}|_{\mathrm{U}}, n'|_{.}), m_1\} \\
IDP_2 \rightarrow SP & m_5 = S_{k^-|_{\mathrm{IDP}_2}}(i_{sess}|_{\mathrm{U}}, d_{idp2}|_{\mathrm{U}})
\end{array}$$

(a) Protocol description



(b) Information layer

$$\mathcal{C}^0_{sp} = \mathcal{C}_{pub} \cup \{k^-|^{\pi}_{\mathrm{SP}}\}, \mathcal{C}^0_{ls} = \mathcal{C}_{pub} \cup \{k^-|^{\pi}_{\mathrm{LS}}, i_p|^{\lambda}_{21}, i_{s,1}|^{\lambda}_{21}, i_{s,2}|^{\lambda}_{21}\},$$
$$\mathcal{C}^0_{idp1} = \mathcal{C}_{pub} \cup \{k^-|^{\pi}_{\mathrm{IDP}_1}, i_p|^{\iota}_{7}, i_s|^{\iota}_{7}, d|^{\iota}_{7}\}, \mathcal{C}^0_{idp2} = \mathcal{C}_{pub} \cup \{k^-|^{\pi}_{\mathrm{IDP}_2}, i_p|^{\iota'}_{2}, i_s|^{\iota'}_{2}, d|^{\iota'}_{2}\},$$
$$\text{with } \mathcal{C}_{pub} = \{k^+|^{\pi}_{\mathrm{SP}}, i|^{\pi}_{\mathrm{SP}}, k^+|^{\pi}_{\mathrm{LS}}, i|^{\pi}_{\mathrm{LS}}, k^+|^{\pi}_{\mathrm{IDP}_1}, i|^{\pi}_{\mathrm{IDP}_1}, k^+|^{\pi}_{\mathrm{IDP}_2}, i|^{\pi}_{\mathrm{IDP}_2}\}.$$

(c) Object layer: initial knowledge

$$\mathcal{C}_{sp} = \mathcal{C}^0_{sp} \cup \{\{m_1, m_2, m_3, m_4, m_5\}|^{\pi,1}, \{m_1, m_2, m_3, m_4, m_5\}|^{\pi,2}\},$$
$$\mathcal{C}_{ls} = \mathcal{C}^0_{ls} \cup \{\{m_2, m_3\}|^{\pi,1}, \{m_2, m_3\}|^{\pi,2}\}, \quad \mathcal{C}_{idp1} = \mathcal{C}^0_{idp1} \cup \{m_1|^{\pi,1}, m_1|^{\pi,2}\},$$
$$\mathcal{C}_{idp2} = \mathcal{C}^0_{idp2} \cup \{\{m_4, m_5\}|^{\pi,1}, \{m_4, m_5\}|^{\pi,2}\}$$

(d) Object layer: knowledge after two instances $(\pi, 1)$, $(\pi, 2)$ of attribute aggregation

- P1: copies of d_1, d_2 in same context detectable and linkable w.r.t. SP
- P2: i_2, d_2 undetectable w.r.t. IdP1; i_1, d_1 undetectable w.r.t. IdP2
- P3: i_1, i_2, d_1, d_2 undetectable, all items of interest related to user anonymous w.r.t. LS
- P4: copies of d_1, d_2 from different contexts unlinkable w.r.t. SP

(e) Goals for actor knowledge

Fig. 4. Formal model of TAS^3 attribute aggregation

it is specified that this interpretation does not hold for IdP2 w.r.t. IdP1. To capture this interpretation, in general, one would need to define an actor's knowledge about the knowledge of another actor, which is not possible in our model. Accordingly, our interpretation is really less strict: if the architecture does not satisfy our version of P2, then it also does not satisfy the strict version; however, the opposite implication does not hold.

6.3 Formal Analysis and Discussion

We follow the three steps outlined in Section 5 to check whether the properties in Fig. 4(e) hold in the formal model in Figs. 4(a)–4(d). Our results have been obtained using a Prolog implementation of the deductive system. First we check the properties about the SP's knowledge: P1 and P4. Step 1 gives $\mathcal{C}_{sp} \vdash d_{idp1}|^{\pi,1}_{\mathrm{U}}$ and $\mathcal{C}_{sp} \vdash d_{idp2}|^{\pi,1}_{\mathrm{U}}$, and step 2 gives $d_{idp1}|^{\pi,1}_{\mathrm{U}} \leftrightarrow_{sp} d_{idp2}|^{\pi,1}_{\mathrm{U}}$. Because $d_{idp1}|^{\pi,1}_{\mathrm{U}}\sigma = d_1$ and $d_{idp2}|^{\pi,1}_{\mathrm{U}}\sigma = d_2$, the copies of d_1, d_2 in $(\pi, 1)$ are detectable and linkable w.r.t. the SP. The same applies to the copies in $(\pi, 2)$. Thus, P1 holds. For P4, we need to check that items from different contexts are unlinkable w.r.t. SP, i.e., $d_{idp1}|_{\mathrm{U}}^{\pi,1} \leftrightarrow_{sp} d_{idp1}|_{\mathrm{U}}^{\pi,2}$ can not be derived in step 2. Indeed the link cannot be made, so TAS^3 attribute aggregation satisfies P4. Note that this conclusion crucially depends on the nonces being different between protocol instances. Note also that these properties depend on linking and distinguishing information instances and so they cannot be verified using standard deductive systems.

On the other hand, P2 and P3 do not hold: both LS and IdP2 can detect the objects $d_{idp1}|_{\mathrm{U}}^{\pi,*}$ (with $* \in \{1, 2\}$) representing the information d_1. This is due to message m_1, representing the authentication assertion signed by IdP1, being included in the messages from the SP to the LS and IdP2. However, undetectability of i_1, i_2, and d_2 and anonymity of these items w.r.t. the LS do hold. As in [15], we see that the stricter interpretation of P2 that we discussed earlier does not hold: indeed, IdP2 receives an authentication assertion about the user which it knows it has been signed by IdP1. Finally, note that all parties involved in the protocol learn the session identifiers $i_{s,1}$ and $i_{s,2}$ in the process. In particular, if IdP1 and IdP2 collude, then from their known messages they can link their user profiles — again a conclusion of studying relations between personal information that standard deductive systems cannot express.

Our analysis leads to two recommendations on how privacy in TAS^3 attribute aggregation may be improved. First, the SP should not forward the attribute d_1 from IdP1 to the LS and IdP2. However, implementing this is difficult. Indeed, according to the TAS^3 attribute aggregation requirements, the LS and IdP2 should receive a signed authentication assertion from IdP1 to be sure that the user did actually authenticate. In the standards used in TAS^3, the attribute value is part of that authentication assertion. Therefore, a mechanism is desired that allows IdP1 to prove that the user has authenticated without disclosing attribute values. Second, the problem of collision between IdP1 and IdP2 should be avoided by not having a shared identifier between IdP1 and IdP2, but this requires IdP2 to trust the LS that the user has indeed been authenticated.

7 Conclusion and Future Work

In this paper, we considered privacy in IdM by presenting a novel method for privacy analysis of communication protocols. We presented a three-layer model of personal information and showed that it allows for an accurate representation of an actor's knowledge. We showed how to reason about this model using deductive methods, and how to check which privacy properties hold after communication. We demonstrated the feasibility of our approach by a) showing that existing deductive systems can be adapted to our approach; b) proving that such an adaptation does not reduce the expressiveness of the deductive system; and c) performing an case study which made it possible to identify a number of privacy issues in the design of an existing IdM architecture.

This work provides several interesting directions for future work. First, we aim to integrate our three-layer model and deductive system into a state transition system approach. This makes it possible to fully automate the protocol verification, and provides opportunities for the development of tooling. The ability to model false information and probabilistic knowledge of links provides an interesting connection to record linkage theory [20]. Namely, it raises the question whether an entity can be identified (almost) uniquely from a profile with data items that by themselves are not identifying. Another extension to the model is to consider provability of links between pieces of information. The signed authentication assertion from TAS^3 attribute aggregate is an example application for this; electronic payment systems are another. Finally, we are analyzing a number of IdM systems and modeling additional cryptographic primitives they use.

References

1. Sommer, D. (ed.): PRIME Architecture V3. Version 1.0, http://www.prime-project.eu/
2. Kellomäki, S. (ed.): D2.1 - TAS^3 Architecture. Version 17, http://tas3.eu/
3. Scavo, T., Cantor, S. (eds.): Shibboleth Architecture: Technical Overview. Working Draft 02, http://shibboleth.internet2.edu/shibboleth-documents.html
4. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: the spi calculus. In: Proc. of CCS 1997, pp. 36–47. ACM (1997)
5. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. 8, 18–36 (1990)
6. Meadows, C.: Formal methods for cryptographic protocol analysis: emerging issues and trends. IEEE Journal on Selected Areas in Comm. 21(1), 44–54 (2003)
7. Paulson, L.C.: The Inductive Approach to Verifying Cryptographic Protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
8. Bella, G., Paulson, L.: Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 361–375. Springer, Heidelberg (1998)
9. Delaune, S., Ryan, M., Smyth, B.: Automatic verification of privacy properties in the applied pi calculus. In: Trust Management II. IFIP AICT, vol. 263, pp. 263–278. Springer, Heidelberg (2008)
10. Aziz, B., Hamilton, G.: A Privacy Analysis for the π-calculus: The Denotational Approach. In: Proc. of SAVE 2002, Copenhagen, Denmark (July 2002)
11. Brusò, M., Chatzikokolakis, K., den Hartog, J.: Formal Verification of Privacy for RFID Systems. In: Proc. of CSFW 2010, pp. 75–88. IEEE (2010)
12. Veeningen, M., de Weger, B., Zannone, N.: Modeling Identity-Related Properties and Their Privacy Strength. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 126–140. Springer, Heidelberg (2011)
13. Clarke, E., Jha, S., Marrero, W.: Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In: Proc. of ICPCM 1998, pp. 86–106. Chapman & Hall, Ltd., Boca Raton (1998)
14. Ramsdell, B., Turner, S.: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2: Message Specification. RFC 5751 (2010)
15. Chadwick, D., Inman, G.: Attribute Aggregation in Federated Identity Management. IEEE Computer 42(5), 33–40 (2009)
16. Chadwick, D. (ed.): Design of Identity Management, Authentication and Authorization Infrastructure. Version 2.1.1, http://tas3.eu/
17. TAS^3 Protocols, API, and Concrete Architecture. Version 10, http://tas3.eu/
18. Cantor, S., Kemp, K., Philpott, R., Maler, E. (eds.): Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, (March 15, 2005), http://saml.xml.org/saml-specifications
19. Hodges, J., Kemp, K., Aarts, R., Whitehead, G., Madsen, P. (eds.): Liberty ID-WSF SOAP Binding Specification. Version 2.0, http://projectliberty.org/
20. Fellegi, I., Sunter, A.: A Theory for Record Linkage. Journal of the American Statistical Association 64(328), 1183–1210 (1969)

Load Time Security Verification[⋆]

Olga Gadyatskaya, Eduardo Lostal, and Fabio Massacci

DISI, University of Trento, Italy
{surname}@disi.unitn.it

Abstract. Modern multi-application smart cards can be an integrated environment where applications from different providers are loaded on the fly and collaborate in order to facilitate lives of the cardholders. This initiative requires an embedded verification mechanism to ensure that all applications on the card respect the application interactions policy.

The Security-by-Contract approach for loading time verification consists of two phases. During the first phase the loaded code is verified to be compliant with the supplied contract. Then, during the second phase the contract is matched with the smart card security policy. The paper focuses on the first phase and describes an algorithm for static analysis of the loaded bytecode on Java Card. The paper also reports about implementation of this algorithm that can be embedded on a real smart card.

1 Introduction

Multi-application smart cards are an appealing business scenario for both smart card vendors and smart card holders. Applications interacting on such cards can share sensitive data and collaborate, while the access to the data is protected by the tamper-resistant integrated circuit environment. In order to enable such cards a security mechanism is needed which can ensure that policies of each application provider are satisfied on the card. Though a lot of proposals for access control and information flow policies enforcement for smart cards exist [2], [9], [10], [12], they fall short when the cards can evolve. The scenario of a dynamic and unexpected post-issuance evolution of a smart card in the field, when applications from potentially unknown providers can be loaded or removed, is novel and not yet treated comprehensively.

For a dynamic scenario, traditionally, run-time monitoring is the preferred solution. But smart cards do not have enough computational capabilities for implementing complex run-time checks. Thus the proposal to adapt the Security-by-Contract approach (initially developed for mobile devices [4]) for smart cards appeared. In the Security-by-Contract (S×C) approach each application supplies on the card its contract, which is a formal description of the application behavior. The contract is verified to be compliant with the application code, and then the system can ensure that the contract matches the security policy of the card.

⋆ Work partially supported by the EU under grant EU-FP7-FET-IP-Secure Change.

The S×C framework deployed on the card consists of two main components integrated with the card manager. These two components are the ClaimChecker and the PolicyChecker. The ClaimChecker performs extraction of the contract and verifies that it is compliant with the application code. Then the PolicyChecker ensures that the security policy of the card is compliant with the contract. This component is also responsible for updating the security policy after each evolution of the card and maintaining it across updates. A proof-of-concept implementation of the PolicyChecker component is described in [3]. The PolicyChecker prototype was developed in a form of an application installable and runnable on a smart card, thus this prototype demonstrated feasibility of the embedded PolicyChecker implementation.

The loading time verification mechanism for secure application interactions requires a careful investigation of multi-application smart card platforms. We have chosen to focus on the Java Card technology as one of the current leaders for open multi-application smart cards implementation. We present in Section 2 a brief overview of this technology and then we outline the S×C solution for Java Card (Section 2.2) emphasizing the changes to the platform. The Java Card internals are discussed more deeply in Section 3. In this section we focus on the loading process and the run-time environment. We then concentrate on the application contracts in Section 4, discussing the contract can be created and the mechanism to deliver it securely on the card.

In this paper we propose an algorithm for the ClaimChecker component of the S×C framework for the Java Card technology (Section 5). The ClaimChecker parses the bytecode loaded on the card, extracts the contract and compares it with the actual code of the application. The ClaimChecker component is an intricate part of the S×C framework, because its implementation requires access to the loaded application code. We report about implementation of the ClaimChecker algorithm in C. For on-card prototypes it is important that they have small memory footprints. We therefore present the memory usage statistics (for EEPROM and RAM) that demonstrates feasibility of the approach (Section 6). The related work is discussed in Section 7 and we conclude with Section 8.

The main contributions of our current work are:

– The specification of the application contracts;
– The algorithm for the ClaimChecker component of the S×C framework;
– The implementation of the algorithm in C demonstrating that the algorithm can be embedded onto an actual smart card chip.

2 The S×C Architecture for the Java Card Platform Evolution

Java Card is a popular middleware for multi-application smart cards that allows post-issuance installation and deletion of applications. Application providers develop *applets* (Java Card applications) in a subset of the Java language. This subset is object-oriented, but misses some traditional Java data types and features. Full description of the Java Card language is provided in [11].

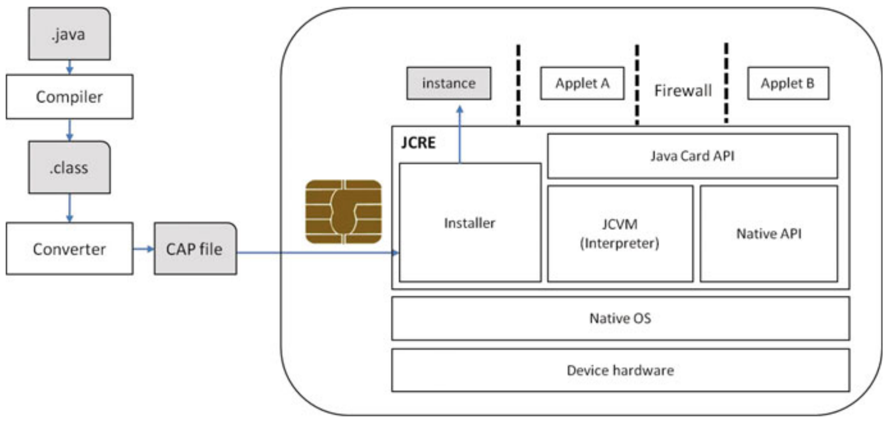


Fig. 1. The Java Card Architecture and the Loading Process

Currently smart cards in the field run on the Java Card version 2.2.2, thus our proposal supports this version. Also a new specification for Java Card 3.0 is published, but its developments are currently frozen due to, among all, security concerns. However, the S×C approach we advocate in the future can be ported also for the third generation of Java Cards.

2.1 The Java Card Platform Architecture and the Loading Process

Figure 1 presents the architecture of a chip with the Java Card platform installed and the application loading process. The architecture comprises several layers which include device hardware, an embedded operating system (native OS), the Java Card run-time environment (JCRE) and the applications installed on top of it [11]. Important parts of the JCRE are the Java Card virtual machine (JCVM) (its Interpreter part) and the Installer, which is an entity responsible for post-issuance loading and installation of applications.

Applets are supplied on the card in packages. The source code of a package is converted by the application providers into class files and then (using a Converter which is actually an off-card part of the JCVM) into a CAP file. The CAP file is transmitted onto a smart card, where it is processed, linked and transformed into a platform-specific executable format (defined by the platform developer). Application providers do not need to consider different on-card executable formats, as they are just required to supply a correct (compliant with the Java Card specifications) CAP file. Then, upon finalization of the linking process, an applet instance is installed.

One of the main technical obstacles for the verifier running on Java Card is unavailability of the application code (in a known format of a CAP file) for reverification purposes after linking. Thus the application policy cannot be stored within the application code itself, as the verifier will not have access to it later.

Applications on Java Card are separated by a firewall and the interactions between applets from different packages are mediated by the JCRE. If two applets belong to different packages, their *contexts* are different, and the Java Card firewall confines applet's actions to its designated context. Thus, normally, an applet can reach only objects belonging to its own context. The only applet's objects accessible through the firewall are methods of specific *shareable interfaces*, also called *services*. A shareable interface is an interface that extends `javacard.framework.Shareable`.

If an application A implements some services, it is called a *server*. An application B that tries to call any of these services is called a *client*. A typical scenario of service usage starts with a client's request to the JCRE for a reference to A's object (that is implementing the necessary shareable interface). The firewall passes this request to application A, which decides if the reference can be granted or not. If the decision is positive, the reference is passed through the firewall and is stored by the client for further usage. The client can now invoke any method declared in the shareable interface which is implemented by the referenced object. During invocation of a service a context switch will occur, thus allowing invocation of a method of the application A from a method of the application B. A call to any other method, not belonging to the shareable interface, will be stopped by the Java Card firewall [11].

As all applet interactions inside one package are not controlled by the firewall and due to the fact that a package is loaded in one pass (thus it is not possible to load a malicious applet in one package with an honest one), we consider that one package contains only one applet and there is an one-to-one correspondence between packages and applications.

2.2 Security-by-Contract for Java Cards

The Security-by-Contract framework for smart cards provides an extension of the Java Card architecture with two main components: the ClaimChecker and the PolicyChecker. The loading time verification process is performed by these components. Another addition to the platform is the Policy applet. The applet appears due to the fact that only applications can allocate space in EEPROM (mutable persistent memory), that is the only type of memory suitable to store the security policy across updates. We have solved the issues of the application code unavailability after linking by storing the security policy (that incorporates each installed application policy) in a separate accessible Policy applet.

Figure 2 depicts the proposed architecture, the additions to the JCRE are in long dashed rectangles. More details about the architecture and its implementation are given in Section 6.

This paper focuses on the ClaimChecker component, that is responsible for contract-code matching. Thus only the application loading scenario is relevant for the ClaimChecker, as during the application removal the code has already been verified to be compliant with the contract. The workflow of the loading scenario follows (only the actions relevant to the S×C process are listed):

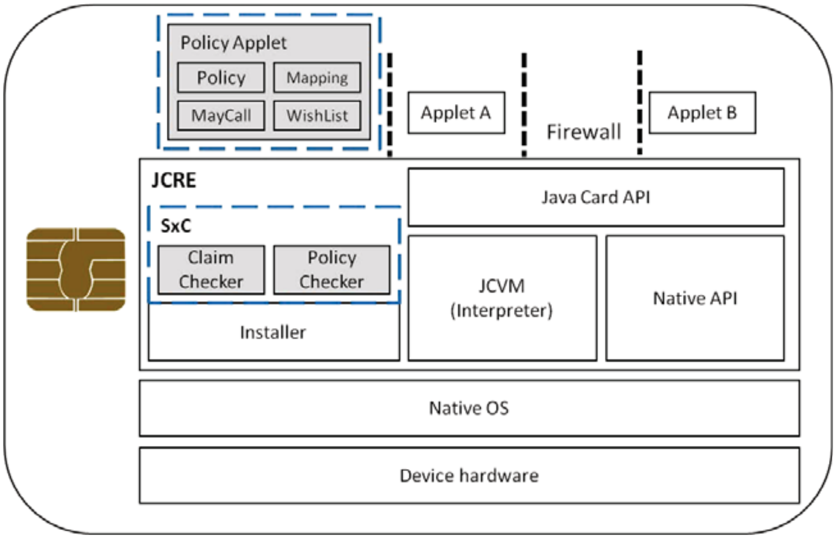


Fig. 2. The Security-by-Contract Extended Architecture

1. New package B is loaded (CAP file is transmitted to the card, the Installer receives it and saves into the modifiable memory);
2. The Installer retrieves the current security policy from the Policy applet and invokes the ClaimChecker;
3. The ClaimChecker gets the contract from the CAP file and runs the verification algorithm;
4. If the ClaimChecker succeeds, it invokes the PolicyChecker and sends it the pointer to the contract;
5. The PolicyChecker gets the security policy and runs the contract-policy compliance algorithm;
6. If the PolicyChecker succeeds, it communicates the update to the security policy;
7. If the ClaimChecker and the PolicyChecker succeeded, B is linked and stored in the persistent memory, and the card security policy is updated to include its contract. Otherwise, B is rejected and removed from the memory.

The S×C framework verifies that the following two properties will be satisfied on the card after any accepted change:

– *Service Invocation Security*: If an application A calls during its execution a service s of an application B, then B has authorized A to access s in B's security policy;
– *Available Functionality*: If an application A declared that it needs a service s of an application B in order to be functional, then the service s is indeed provided by B.

The formal proof of these properties established on the Java Card platform by the S×C framework relied on the fact of existence of a sound ClaimChecker algorithm [6]. In fact, the ClaimChecker component is the corner stone of the S×C framework, and it's specification and implementation were the key tasks while building the framework.

2.3 Threats to Validity of the SxC Approach

The S×C approach and the guarantees it provides are ensured with the certain assumptions made. Obviously, soundness of the framework algorithms relies on the correct implementation of the JCRE and the JCVM, and we assume they are in full compliance with the specifications [11]. For the invoked services we rely on the trustworthiness of the Compiler, that has to be compliant with the Java type safety requirements. We also assume that the bytecode was not tampered with after compilation and conversion.

For the provided services, we rely on the trustworthiness of the servers. Indeed, in the S×C paradigm provision of a service requires a commitment to implement the necessary shared object and to provide a correct object reference in response to a request from any client. The server has to rely on the loading time verification by the S×C framework and it should not use the access control mechanisms embedded into the code. We also have to assume the correctness of the server implementation.

The S×C framework enforces access control for direct services usage. We would like to mention that the current access control enforcement on Java Card is embedded into the application code. Traditionally, the server will receive an AID of the client requesting its service from the JCRE and check that this client is authorized before granting it the reference to the object (that can implement multiple services). Once the object reference is received, the client can access all the services within this object and it can also leak the object reference to other parties. The S×C framework checks the authorizations for each service access, thus the object reference leaks are no longer a security threat.

3 The Java Card Internals

We now present the Java Card platform details that were used to build the S×C framework and to guarantee the security it enforces. In order to realize the application interaction scenario the client has necessarily to import the shareable interface of the server and to obtain the *Export file* of the server, that lists shared interfaces and services and contains their tokens. The server's Export file is necessary for conversion of the client's package into a CAP file. In a CAP file all methods are referred to by their tokens, thus during conversion from class files into a CAP file the client needs to know correct tokens for services it invokes from other applications. As shareable interfaces and Export files do not contain any implementation, it is safe to distribute them.

Tokens are used by the JCRE for linking on the card similarly as Unicode strings are used for linking in standard Java class files. A service s can be identified as a tuple $\langle A, I, t\rangle$, where A is a unique application identifier (AID) of the package that provides the service s, I is a token for a shareable interface where the service is defined and t is a token for the method in the interface I. Further we will sometimes omit an AID and will refer to a service as a tuple $\langle I, t\rangle$.

We discuss now the CAP files and service invocation details used further in the ClaimChecker algorithm. The JCRE imposes some restrictions on method invocations in the application code [11]. Only the opcode `invokeinterface` in the code allows to perform the desired context switch. Thus, in order to collect all potential service invocations we need to analyze the bytecode and infer from the `invokeinterface` instructions possible services to be called.

Opcode "`invokeinterface` *nargs I t*" has 3 (explicit) operands, as defined in the JCVM specification [11, Sec. 7.5.54]. Operand *nargs* defines a number of invoked method arguments (plus 1), operand I provides an index in the Constant Pool component where the structure at this index should correspond to a reference to an interface class and operand t is an interface method token for the method to be invoked. Meanwhile, the stack before execution of the opcode `invokeinterface` *nargs I t* should contain on its top an object reference R, followed on the operand stack by $nargs-1$ words of arguments.

Intuitively, while analyzing the code, we could try to track the object references on the stack, thus inferring all possible objects of the server that could be referenced by the applet during `invokeinterface` opcode execution. But unfortunately, it is only the server's code that defines which objects it will provide and to whom. It is even possible the server is not yet on the card when the client is loaded (and it could never arrive). Thus our analysis can be only as precise as the tokens provided in the client's code.

4 Application Contract

Let $A.s$ be a service s declared in a package A. The contract consists of two parts: a *claim* and a *policy*. AppClaim specifies provided (Provides set) and invoked (Calls set) services. We say that the service $A.s$ is provided if applet A is loaded and service s exists in its code. Service $B.m$ is invoked by A if A may try to invoke $B.m$ during its execution. The AppClaim will be verified for compliance with the bytecode (the CAP file) by the ClaimChecker.

The application policy AppPolicy contains authorizations for services access (sec.rules set) and functionally necessary services (func.rules set). We say a service is necessary if a client will not be functional without this service on board. The AppPolicy lists applet's requirements for the smart card platform and other applications loaded on it.

Thus the application contract has the following structure: $\mathsf{Contract} = \langle \mathsf{AppClaim}, \mathsf{AppPolicy}\rangle$, where $\mathsf{AppClaim} = \langle \mathsf{Provides}, \mathsf{Calls}\rangle$ and $\mathsf{AppPolicy} = \langle \mathsf{sec.rules}, \mathsf{func.rules}\rangle$.

A functionally necessary service for applet A is the one which absence on the platform will crash A or make it useless. For example, a transport application normally requires some payment functionality to be available. If a customer will not be able to purchase the tickets, she would prefer not to install the ticketing application from the very beginning. It is required that for every application A $\mathsf{func.rules}_\mathsf{A} \subseteq \mathsf{Calls}_\mathsf{A}$.

An authorization for a service access includes the package AID of the authorized client (the format of an authorization will be discussed further). The access rules have to be specified separately for each service and each client that the server wants to grant access.

4.1 The Contract Delivered on the Card

Contracts can be delivered on the card within Custom components of the CAP files. CAP files carrying Custom components can be recognized by any Java Card Installer, as the Java Card specification requires.

Custom components require to have a tag and an AID. We have defined the tag to be 0xC3 and the AID 0x010203040506C3 (but these can be easily modified). These details of the Custom component and its length are listed in the Descriptor component of the CAP file.

Table 1. Structure of the Custom component Containing Contract

```
contract {
    u2 provides_count
    provides_info  provides[provides_count]
    u2 calls_count )
    calls_infocalls[calls_count]
    u2 secrules_count
    secrules_info  secrules[secrules_count] }
```

The scheme of the contract is illustrated in Table 1. The order of the contract attributes is expected to be: Provides, Calls, sec.rules. Thus we just add the number of corresponding elements before each attribute. Elements of each attribute have specifically defined structures (we use structures and naming that are similar to the ones defined for CAP files [11], there u2 corresponds to 2 bytes). The contract is just a byte array, but specifying structures corresponding to each entry allows us to perform the contract extraction efficiently. More information on the structures is available in the companion technical report [7].

Functionally necessary services are a subset of called services, thus we just tag necessary services among the called ones. The value of specific funcrules_tag is set to 0x01 if the service should be listed in func.rules. Otherwise the tag value should be 0x00.

4.2 Contract Population

Now we discuss how to populate the contract and embed it into the CAP file. Following are the rules for contract population.

– *Provided Services.* A service is required to be listed in the Provides set if it is a method of an interface extending Shareable. A service is listed in Provides array as a pair $\langle \mathsf{I}, \mathsf{t} \rangle$, where I is the Export file token for shareable interface and t is the Export file token for the method (1 byte each).

- *Called and Functionally Necessary Services.* An application provider should list a service (belonging to another package) in the Calls set, if an invocation of this service is present in the code of the applet. A service from a package with AID XXX is listed in the contract as $\langle XXX, \mathsf{I}, \mathsf{t}, \mathsf{funcrules\_tag} \rangle$, where funcrules_tag tags if this service is also functionally necessary or not. For optimization purposes, the Calls set is then restructured to separate services provided by different servers. The AIDs are space-consuming objects (can take up to 16 bytes) and avoiding their repetitions where possible can bring significant space savings.
- *Authorization Rules.* An authorization rule is listed in the sec.rules set as a pair containing the service details (defined as a provided service) and the authorized client package AID. Thus the structure is the same as for a called service, with a difference that no tag for functionality is needed: $\langle AID, \mathsf{I}, \mathsf{t} \rangle$. Then the same optimization strategy as for called services is applied.

The CAP file is in fact a JAR archive with a known structure. In order to embed the contract created by these rules and in compliance with the structure from Table 1, our CAP modifier takes the CAP file generated with the standard Java Card tools and appends the Contract Custom component within it, modifying the Descriptor component accordingly (as the specification requires).

5 The Claim Checker Algorithm

The ClaimChecker component is responsible for verification of the contract and the bytecode compliance. Thus it has to establish that the services from $\mathsf{Provides_A}$ exist in package A and the services from $\mathsf{Calls_A}$ are indeed the only services that A can try to invoke in its bytecode. The details of the service invocation instructions were already discussed in Section 3. The goal of the ClaimChecker algorithm is to collect for each `invokeinterface` opcode the method index t and the Constant Pool index I. Then we can compare the collected set with the set Calls of the contract. We emphasize that operands of the `invokeinterface` opcode are known at the time of conversion into a CAP file and thus are available directly in the bytecode. All methods of the application are provided in the Method Component of the application's CAP file, an entry for each method contains an array of its bytecodes. Exported shareable interfaces are listed in the Export component of the CAP file and flagged in the Class component. The strategy for the ClaimChecker is to ensure that each service listed in the Provides set is meaningful and no other provided services exist.

5.1 The Algorithm

The ClaimChecker Algorithm 5.1 processes the CAP file components in order of appearance with a standard Installer, the comments on the steps of the algorithm

```
Require: A CAP file.
Ensure: True/False, Contract.
 1: //Header Component: get the current package AID
 2: byte $CurrentPID[16]$ gets current package AID;
 3: // Import Component: get package AIDs of imported packages
 4: add ⟨imported package ID, internal imported package token (index in the current
    array)⟩ to $ImportedPackages$;
 5: // Constant Pool Component: get imported interfaces
 6: for all elements of the Constant Pool array of the type class_ref do
 7:    if the high bit equals to 1 then
 8:       add ⟨imported package token, external class or interface token, internal class
          or interface token (index in the current array)⟩ to $ImportedInterfaces$;
 9: // Method Component: parse bytecode of the methods to identify called services
10: for each method of the methods[ ] array do
11:    if invokeinterface X Y Z opcode is in the method then
12:       add ⟨internal token of the interface, external token of the method⟩ to
          $InvokedServices$;
13: // Export Component: get tokens of shareable interfaces
14: for $i = 0$ to class_count do
15:    add ⟨offset into the Class component, external interface token⟩ to
       $ExportedInterfaces$;
16: // Descriptor Component: get external tokens of provided services
17: for $i = 0$ to classes_count do
18:    if classes[i] has a flag ACC_INTERFACE = 0x40 AND exists $\langle int\_offset, I\rangle \in$
       $ExportedInterfaces$ such that int_offset = classes[i].this_class_ref then
19:       // This interface is shareable and its external token was collected
20:       for all methods of this interface do
21:          add ⟨external interface token, method token⟩ to $ListedServices$;
22: // Custom Component: get Contract
23: for $j = 0$ to provides_count do
24:    add ⟨external interface token, external method token⟩ to $ContractProvides$;
25: for $j = 0$ to calls_count do
26:    add ⟨external interface token, external method token, AID⟩ to $ContractCalls$;
27:    if funcrules_tag = 0x01 then
28:       add ⟨external interface token, external method token, AID⟩ to
          $ContractFuncrules$;
29: for $j = 0$ to secrules_count do
30:    add ⟨external interface token, external method token, AID⟩ to
       $ContractSecrules$;
31: // The Final Check: return true iff the collected sets match with the Contract
32: Check of called services: construct the same structure as in the contract and check
    for mutual inclusion
33: for each $\langle I, t, AID\rangle \in ContractCalls$ do
34:    add $\langle I, t, P\rangle$ to $CALLS$ such that $\langle P, AID\rangle \in ImportedPackages$;
35: for each $\langle P, I, cpt\rangle \in ImportedInterfaces$ and $\langle cpt, t\rangle \in InvokedServices$ do
36:    add $\langle P, I, t\rangle$ to $CALLS1$;
37: if $CALLS1 \neq CALLS$ then
38:    return False;
39: else
40:    // Check for provided services: all services in ContractProvides set have valid
       interface and method tokens
41:    if $ContractProvides \neq ListedServices$ then
42:       return False
43:    else
44:       return {True, $CurrentPID$, $Contract$}
```

Algorithm 5.1. The Claim Checker Algorithm

are inlined. The presented algorithm is a script for an actual implementation of the ClaimChecker. The received CAP file is a byte array, but it is structured accordingly to the CAP file specification [11]. Thus the algorithm refers directly to items (fields) of the structures defined in the CAP file specification, such as CONSTANT_Classref_info structure or Interface_info structure. The algorithm also uses variable-length arrays and arrays of tuples, that do not exist on a smart card. The actual implementation explores just constant-length byte arrays. The function offset(b) is used in the algorithm, that serves as a pointer and returns a structure S which is provided at the given offset b.

Soundness of the algorithm for the service invocation security (in assumption of a correct JCRE implementation) follows from the fact that only invokeinterface opcode allows the JCRE to switch the context, thus any application can only use this opcode to invoke services. Thus the ClaimChecker will accept only the applications that have declared the invoked services set Calls honestly. We discuss the soundness proof in more details in the companion technical report [7].

6 Implementation of the Claim Checker

We have implemented full S×C prototype in C, as it is a standard language for smart card platform components implementation. In this section we will give an overview of the prototype architecture and implementation details, and then we will focus on the ClaimChecker component implementation and present the memory usage statistics.

The main C components of the S×C prototype are:

SxCInstaller. This component is an interface with the Installer. SxCInstaller calls the ClaimChecker that in a positive case (contract and bytecode are compliant) will return the address of the contract in the Contract Custom Component of the CAP file being loaded. The SxCInstaller also comprises (for memory saving reasons) the PolicyChecker component. Any negative result either in the ClaimChecker or PolicyChecker algorithms or errors during parsing of the CAP file are propagated as *false* to the SxCInstaller, that returns a *boolean* to the Installer.

ClaimChecker. This component is called by SxCInstaller. It carries out the check for the compliance between the contract and the CAP file. The check is carried out after parsing the CAP file. By means of the functions of the CAPlibrary library for CAP file parsing on-card (discussed further), this component gets the initial address of the components it needs from which it can eventually parse the rest of the components. If the result is positive, the ClaimChecker will return the address of the contract of the application in the Contract Custom component. Any error during parsing or a negative result from the ClaimChecker leads to return of *null*.

We now discuss the implementation of the proposed algorithm 5.1 in C. In order to reduce the amount of RAM memory the prototype uses, instead of copying parts of the CAP file (for example, the delivered contract) we operated with the pointers to the corresponding parts of the CAP file. We have used a set of functions to access the parts of CAP file components, calling it the CAPlibrary library, assuming that for each component we can retrieve its location in the card memory and its size. These functions belong to a standard functionality of the Installer. As we did not have access to an actual smart card platform implementation, we have implemented these functions in C for testing purposes, but we do not include this implementation in the following memory statistics of the prototype.

6.1 The Policy Checker and the Policy Applet Implementation

Due to the lack of space we do not report the details of the PolicyChecker implementation. However, we present the security policy data structures just to give a flavor of this part of the system.

The security policy stored on the card consists of contracts of the currently loaded applications. A contract in the form supplied on the card is a space-consuming structure. Each AID can occupy up to 16 bytes. Therefore, a set of sec.rules with authorizations given for, for instance, 8 applets can occupy up to 144 bytes. We would like to save the space necessary for storing the security policy while making the operations with the contracts (performed by the PolicyChecker for contract-policy compliance check) faster. To do so we have resolved to store the security policy on the card in a bit vectors format. The current data structure for security policy assumes there can be up to 4 loaded applets, each containing up to 8 provided services. Thus the security policy is a known data structure with a fixed format, the bits are taking 0 or 1 depending if the applet is loaded or the service is called/provided. This structure is called Policy in Figure 2 (see the Policy applet structures). The amount of the loaded applets can potentially be modified dynamically (if the 5th applet arrives).

The chosen security policy data structure requires the table on the card that maintains correspondence between the number the applet gets in the on-card security policy structure and the actual AID of the package, and between the provided service token and the number of this service in the policy data structure. We store this correspondence in the Mapping object. The other two objects that are part of the on-card security policy are MayCall list and WishList list. The MayCall list contains the potential future authorizations, necessary for a case when a loaded application carries a security rule for some application not yet on the card. These authorizations have to be stored on the card in the form they were supplied (with the client's AID), thus they are space-consuming objects. The WishList object is a set of services that are called by applications but are not yet on the card, because the server is not yet loaded, or because the current version of the server does not provide this service. The WishList set maintains the AIDs of the service providers and the services as tuples $\langle I, t\rangle$. Again, the WishList entries are space-consuming, as they contain AIDs of desired packages.

The Policy applet has to communicate the security policy of the card to the PolicyChecker component that will run the contract-policy compliance check.

This communication is currently implemented through the APDU buffer, that is a common object for communication for all entities on the card. We have assumed the size of the APDU buffer to be 255 bytes, as it is one of the standard implementations. Thus the full security policy (the Policy, Mapping, WishList and MayCall objects) has to fit within 255 bytes. That is why we have developed such a small security policy object, which is enough to fit only 4 loaded applets, and we have set restrictions on the number of authorizations in the MayCall object and desired services in the WishList object. We are currently investigating if there are better means for communication (in both directions) of the C components and the applets on the card that will allow us to implement a bigger and dynamically scalable policy model.

6.2 Details of the Claim Checker Implementation Memory Statistics

We now present an overview of the memory consumption by the ClaimChecker prototype. The most important characteristics for an on-card component are RAM and EEPROM consumption. EEPROM space is required to store the prototype and the necessary data between the card sessions. RAM memory, on the other hand, is used to store the temporary data while the verification is performed. We can consider as an example of a modern smart card chip P5CT072 device from Philips Semiconductors [13]. The chip has 72 KB of EEPROM, 160 KB of ROM and 4608 bytes of RAM. Therefore, we can assume that the verifier embedded on the card should occupy at most 20-30 KB of EEPROM.

As we cannot install the prototype on a card and measure its footprint in the linked state, we explored two metrics for the EEPROM usage: the size of the object files in C and the number of lines of code (LOCs). The ClaimChecker prototype requires 6522 bytes (6.36 KB) to store the object files. The .c file of the ClaimChecker contains 155 LOCs, and the .h file contains 7 LOCs.

RAM usage is also very important, as over-consumption of RAM by the prototype can lead to the denial of service. The higher is the RAM consumption, the less is the level of interoperability of the prototype, because some cards cannot provide a significant amount of RAM for the verifier which has to run in the same time with the Installer. We have used a temporary array of 255 bytes to store the necessary computation data. 255 bytes is a small temporary memory buffer which ensures the highest level of interoperability for the prototype.

7 Related Work

A plethora of works exist for verification of application interactions security on Java Card. Ghindici et al [8] proposed an approach for the information flow verification on small embedded systems. Each application gets a certificate with the information flow signature of each method, and on device these signatures are checked using the proof-carrying-code techniques. The expressive information flow security properties captured the interactions of applications on the platform.

This approach is extremely powerful, but has not yet been demonstrated to be implementable on Java Card.

A lot of papers were dedicated to the static scenarios, when all the applications are known a priori and can be verified using off-card facilities [10], [9], [2], [12]. Dynamic scenarios were considered in [1] and [5]. Avvenuti et al [1] developed the tool JBIFV that was similar to a bytecode verifier and could verify absence of illicit information flows between Java applications. The drawback of this tool in a dynamic scenario is that the applications have to be analyzed locally prior being loaded on the card. Thus the card is not empowered with the ability to make decisions itself.

In the work of Fontaine et al [5] the authors consider the same dynamic scenario as we did and propose an on-card loading time verification approach for transitive control flow policies that can control application collusuons. Their algorithm performs verification while parsing the received CAP file. With respect to [5] our work enforces less stronger policies. However, the S×C approach offers greater flexibility than the transitive control flow policies proposed by Fontaine et al. Indeed, as we have mentioned before, the application code after linking is not available for reverification. Thus the approach by Fontaine et al, that makes the policy compliance verification simultaneously while parsing the bytecode, requires to store a significant amount of additional data related to the invoked methods, what can be a prohibitive requirement for an on-card prototype.

8 Conclusions and Future Work

In the paper we have presented the ClaimChecker component of the S×C framework for the Java Card-based smart cards. This component's duty is to ensure compliance of the applet's contract with its code. The contracts are delivered within the Custom component of the CAP file, and they list provided and called services of the applets and the application providers' policies. We have proposed the structure of the contracts expected by the ClaimChecker in the notation similar to the CAP file contents specification [11].

Once the CAP file is received the ClaimChecker invoked by the Installer component on the card, extracts it and analyzes whether the contract is compliant with the bytecode. Our focus is on the invoked services and we have presented the sound algorithm that can capture the comprehensive list of the called services and match it with the claimed list. The implementation of the algorithm is straight-forward provided that one has access to a smart card platform implementation and knows the necessary APIs to access the CAP file contents.

For the future work we plan to validate the S×C framework implementation within the Secure Change project with the help of Gemalto (an industrial partner in the project). We have implemented the algorithm in C and the memory statistics we have provided ensures that a proof-of-concept embedded implementation is possible. Another interesting direction of the future work is richer contracts. We believe that the perfect trade-off between verification time, richness of the contracts and flexibility of the approach for evolution is yet to be found.

References

1. Avvenuti, M., Bernardeschi, C., De Francesco, N.: Java bytecode verification for secure information flow. SIGPLAN Not. 38, 20–27 (2003)
2. Bieber, P., Cazin, J., Wiels, V., Zanon, G., Girard, P., Lanet, J.-L.: Checking secure interactions of smart card applets: Extended version. J. of Comp. Sec. 10(4), 369–398 (2002)
3. Dragoni, N., Lostal, E., Gadyatskaya, O., Massacci, F., Paci, F.: A load time Policy Checker for open multi-application smart cards. In: Proceedings of the 2011 IEEE International Symposium on Policies for Distributed Systems and Networks (2011)
4. Dragoni, N., Massacci, F., Naliuka, K., Siahaan, I.: Security-by-contract: Toward a semantics for digital signatures on mobile code. In: López, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 297–312. Springer, Heidelberg (2007)
5. Fontaine, A., Hym, S., Simplot-Ryl, I.: On-device control flow verification for java programs. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 43–57. Springer, Heidelberg (2011)
6. Fontaine, A., Hym, S., Simplot-Ryl, I., Gadyatskaya, O., Massacci, F., Paci, F., Jurgens, J., Ochoa, M.: D6.3 Compositional technique to verify adaptive security at loading time on device. SecureChange EU project public deliverable (2010), http://www.securechange.eu
7. Gadyatskaya, O., Lostal, E., Massacci, F.: Load time security verification. The Claim Checker. Technical Report DISI-11-471. On the web, at http://eprints.biblio.unitn.it
8. Ghindici, D., Simplot-Ryl, I.: On Practical Information Flow Policies for Java-Enabled Multiapplication Smart Cards. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 32–47. Springer, Heidelberg (2008)
9. Girard, P.: Which security policy for multiplication smart cards? In: USENIX Workshop on Smartcard Technology. USENIX Association (1999)
10. Huisman, M., Gurov, D., Sprenger, C., Chugunov, G.: Checking Absence of Illicit Applet Interactions: A Case Study. In: Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS, vol. 2984, pp. 84–98. Springer, Heidelberg (2004)
11. Sun Microsystems. Virtual Machine and Runtime Environment. Java Card^{TM} platform. Specification 2.2.2, Sun Microsystems (2006)
12. Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V., Toll, D.: Verification of a formal security model for multiapplicative smart cards. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 17–36. Springer, Heidelberg (2000)
13. Philips Semiconductors. P5CT072 Secure Dual Interface PKI Smart Card Controller. On the web, at http://www.usmartcards.com/images/pdfs/pdf-199.pdf

Preserving Location Privacy for Continuous Queries on Known Route

Anuj S. Saxena, Mayank Pundir, Vikram Goyal, and Debajyoti Bera

Indraprastha Institute of Information Technology Delhi
New Delhi, India
{anujs,mayank09025,vikram,dbera}@iiitd.ac.in

Abstract. Protecting privacy in location based services has recently received considerable attention. Various approaches have been proposed, ranging from mix-zones to cloaking. Cloaking based approaches are ill-suited for continuous queries, where correlation between regular location updates may disclose location information. We consider the cloaking strategy with a modification to suit continuous queries: skip location updates at some key positions. The objective is to trade service availability at some locations in exchange of privacy at all times. Considering the case where the entire path of the user is known in advance, we show how to strategically decide these locations in a manner which is efficient, and does not skip too many locations (compared to the optimum). Experimental results show the validity and effectiveness of the proposed algorithm.

1 Introduction

Due to advances in mobile devices and wireless sensor technology, development of applications for mobile devices has gained intensity in the last decade [15]. These applications range from traditional services like voice communication, internet access, entertainment etc. to a new paradigm of services called as *location based services*(LBS): services which depend on the identity and current spatio-temporal information of the requesting user[1]. Services like navigation assistance, traffic/weather reports, locating persons with common interest, friend finder, location based advertisements [7] are getting popular among users and even gaining attention of commercial ventures.

Though location based services has proved very valuable for consumers, their requirement to disclose identity and location information, as part of the service design, opens a whole system of indirect surveillance, and can understandably cause widespread privacy violations [2,18]. To prevent privacy leaks, various solutions have already been proposed, such as location perturbation, location obfuscation using popular k-anonymity [17] and/or l-diversity [12] models, mix-zone [5,16] etc. All these solutions are usually based on client-server or p2p system architectures and they try to protect different sensitive information like user identity, query and/or location. In this study, we aim to prevent disclosure of location information of a user of LBS in a client-server environment.

Majority of the existing solutions were designed for snapshot queries – user requesting a service once in a while, but they may not work well if a user is querying (and hence

[1] In this paper, *location* will be used to denote the combined spacio-temporal information, viz., at which place and at what time.

sending location information) continuously. An adversary can correlate the sequence of location updates (obfuscated or perturbed) sent by a user using sophisticated statistical techniques to determine actual locations visited by the user. These types of indirect violations, which we refer to as *privacy leaks*, has motivated the research community to design solutions for privacy preservation specific to continuous queries [8,19].

This paper studies the privacy preserving problem in a specific scenario where a privacy module has advance knowledge of the complete path to be taken by the users. We provide an affirmative answer to the privacy question in this setting: can the privacy module use this path information to effectively provide a location based service without compromising privacy in any way. The privacy module, for our case, is a logical entity; it can run on the user device itself. It is often the case that a user knows his path of travel well in advance; even otherwise, planning in advance for the sake of privacy is not too demanding. Even if the privacy module is run by a third-party, and if path is not available from a user, the module may be able to infer it from past traces.

There is always a tradeoff between privacy and service in any LBS, and it is not apriori clear how to *effectively* use the complete path information to reduce privacy leak. Our objective here is to guarantee complete privacy but with lower level of service, if necessary. In this paper, we designed an efficient algorithm that, using complete path information, determines locations in the path which should not be reported. Clearly, service cannot be availed at these locations – we call these as *hidden locations/moves* and denote these locations as $*$ in the user path – and the fewer they are, the better.

Consistent with best practices of security research, we assume that our adversary has knowledge of the strategy used by the privacy module, including the policy for obfuscation and hiding. We allow the adversary to guess any location on the path, but on the other hand, require him to be 100% accurate.

Our main contribution is a novel algorithm to decide which locations are safe to report without violating privacy, which we describe in Section 5 after describing the necessary theoretical framework in Section 4. Experimental study of our algorithm with respect to efficiency and optimality are described in Section 6. Some related work and necessary background are discussed in Sections 2 & 3, respectively.

2 Related Work

Privacy for location-based services has attracted intensive research since the first work of spatio-temporal cloaking proposed by Gruteser and Grunwald [11]. In this work, a trusted middleware generalizes the spatial and temporal dimension of a user query by using a notion of k-anonymity: the region reported by the user query should contain atleast k users so that the requesting user is indistinguishable from other k-1 users. Gedik and Liu [10] then proposed an algorithm called Clique-Cloak which implements a personalized k-anonymity model, and determines a minimal cloak area that satisfies the k-anonymity requirement. Thereafter Mokbel [9] proposed the Casper framework which uses a grid based algorithm to determine the cloaking region and which optimizes the cloaking area.

Specifically for location privacy, a lot of work has been done which anonymizes the actual locations of a user using different notions of privacy [6,1,14]. However, all these solutions address the issue of privacy for snapshot queries and are not well-suited for location privacy in continuous queries.

An interesting solution was proposed by Beresford and Stajano [4] where they give an identity-anonymization based solution that uses the concept called *mixzone*. Psuedo-identities are assigned to users in a mixzone in a way such that no adversary can correlate a user who enters the mixzone to a user who leaves the mixzone with high probability. This work then has been extended for road network by Palanisamy and Liu [16] where they point out many factors necessary to achieve the same objective and propose different mixzone shapes and their effectiveness. All these works allow users to share their actual location information only when they are not in mixzone.

There has also been some work on privacy in location-based services in the continuous query setting [8,19]. The authors of [8] differentiate between query privacy (association between identity and query) and location privacy (association between location and query); however, their solution effectively tackles the problem of query privacy. In [19], the authors consider past visited locations (footprints) of the user as well as trajectories of other users to anonymize the trajectory of the current user.

3 Background

In this section we describe the architectural framework used in this paper which is based on the standard client-server model for location-based services [6]. In this model, privacy is preserved by using a *privacy-preserving proxy* service through which all queries from *users* to *service provider(s)* are relayed.

Service Provider/Adversary ($\mathcal{R}$): We denote the honest-but-curious location-based service provider by $\mathcal{R}$. We look upon $\mathcal{R}$ as the adversary because it may try to maliciously infer (private) location information of users based on the information submitted to it. It also knows the obfuscation and hiding strategies employed by the proxy. We make no assumption on the computation and storage capacity of $\mathcal{R}$, but that it does not have access to background/contextual knowledge, including locality information about users' past and future movements. This assumption is essential for this theoretical work because there can be no bound on the entire background knowledge possessed by any person or service [13].

Users: For the continuous query case, a user avails service by sending a query request periodically to the service which is received, modified and retransmitted by the proxy. The request consists of the following parameters: a user id, current timestamp, location information and query text (including other information needed to answer the query). We will henceforth ignore query text and other information since we are only interested in location privacy. We will also assume that every user uses an opaque user id which reveals no information other than correlating multiple requests; it does not bring any advantage in using different user ids since it has been shown that multiple requests in short timespans using different user ids but by the same user can be correlated to deduce that they belong to the same user [4]. This further allows us to drop the user id from subsequent discussions since we will be always concerned with a single user.

Location Obfuscation Strategy: Now we describe the most crucial component for location privacy in our setup. The location obfuscation strategy we use is the square-encoding case described in [3]. It divides the entire region into square (rectangular) regions which we call as *blocks* — these form our basic units of location i.e. any location is represented by the containing block (we will interchangeably use location and block to mean the same thing). The user is allowed to move from a block to any of the eight neighboring blocks. For analytical reasons, we require the user to move at roughly constant speed and report his location once in each block.

Square arrays of blocks are grouped in a regular fashion to create cells. Note that, the layout of the blocks and cells is fixed and is known to everyone, including $\mathcal{R}$. Location obfuscation is achieved by reporting the containing cell as the blurred region (instead of the actual block); for example, reporting C_{00} as the region when the actual location is the top-left block in Fig. 1. This is different from the k-anonymity based approaches where the location and size depend on location of the users, number of neighbours, etc.

We denote the *obfuscation function* by G, i.e., the cell $G(b)$ is reported for the location b. Sequence of blocks taken by a user is called a path and is denoted by p, and the corresponding sequence of regions obtained from privacy mechanism G is defined as the *cell-path*. We denote a cell path by π. The i-th block of a path p is denoted by p_i and the length of a path, denoted by $|p|$, is the number of blocks in the path (similarly for cell-path). We recognise the length of the current path as also the current time and the i-th block as the block during time i (the first block is at time 1). A particular location obfuscation strategy will depend on the size of the cells, and the layout of blocks and cells. For a particular G, the number of blocks along any side of the cell has been defined in [3] as stretch which we denote by σ.

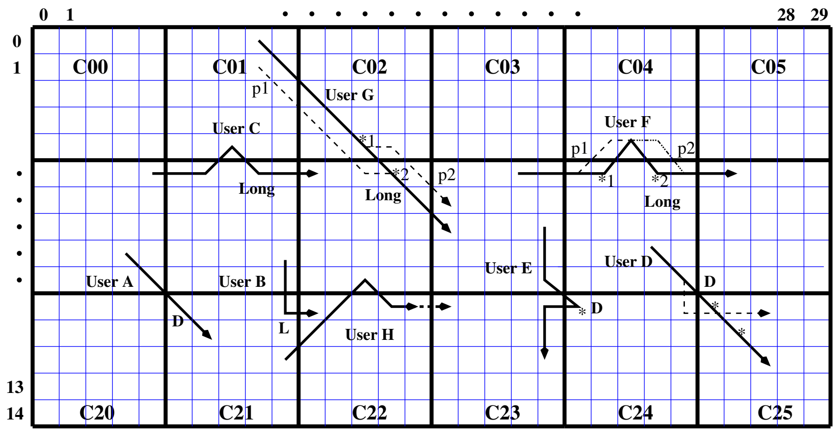


Fig. 1. Location obfuscation with stretch 5

Privacy Preserving Proxy: The main goal of the proxy is to ensure that location privacy of its users is not violated at any point of time. Even though we discuss the proxy separately, but it need not be a separate entity and could run on the user device itself.

With a location obfuscation strategy devised as given above, more than one path may be obfuscated to one cell path, in which case, $\mathcal{R}$ may not be able to deduce which of these is the actual path just given the cell path. Furthermore, if there are two or more paths, say p and p', which have the same obfuscated path and such that for some i in their range, $p_i \neq p'_i$, then $\mathcal{R}$ cannot even accurately deduce which was the i-th block of the path. We require strong guarantees from the adversary, i.e., we say the i-th location is *disclosed* or under *attack* if there is *only one* possible block for the i-th location given the cell-path information – for other cases, the adversary cannot do any better than random guessing[2]. We use the definition of privacy as given in [3] which is based on these observations; we present relevant definitions below, suitably adapted, but please refer to [3] for complete details.

Definition 1 (Privacy Preserving Path). *We call a path p privacy preserving, if at all time $i \in [1, |p|]$ on the path, the i^{th} move will not cause any disclosure (of the then current or of any past locations).*

We state one important lemma which can be proved easily using the results of [3] (proof omitted).

Lemma 1. *For a move at time $t > \sigma + 1$, its effect, with respect to disclosure, is limited to atmost last σ locations (including current).*

Problem Definition: As mentioned earlier, we consider the scenario where the proxy has advance knowledge of the complete path to be travelled by its users. Therefore, we require every user to report the complete path information to the proxy before starting their journey. The proxy pre-computes the locations which are critical for location privacy of the user, and accordingly advises the users to not avail service at these locations which will be called as *hidden locations* (see Section 5.5 where the hidden locations can be computed incrementally if a user wishes to extend his/her path). For the hidden locations, no blurred region is reported, and we denote it by saying $G(p_i) = *$ if p_i should be a hidden location.

The objective of this work is to find a *privacy preserving path* for a known user path by not reporting some locations to the service provider. Therefore, it is essential to compute which locations on a given user path should be marked hidden such that his privacy is preserved as he travels along that path. Of course, it is desirable to have as few hidden location as possible because a user would like to get service at maximum locations. Observe that a brute-force solution to find the *smallest set* of hidden locations will take exponential time (in the length of the path) since it has to check among all possible set of hidden locations to see which ones are privacy preserving. We propose a *Rule Based (RB)* approach (discussed in Section 5) to solve this problem which is a more efficient scheme to determine which locations should be hidden for a given path p such that $G(p)$ is privacy preserving.

4 Theoretical Framework

In this section we develop a theory of attacks by characterising and classifying them based on their properties.

[2] We use the term *attack* to mean the same thing as *disclosure* in [3].

Properties: Following are the required properties of an attack a.

- a.attackType: It represents the type of a, namely D,L or Long.
- a.attackLocation: It represents the location where a took place. If multiple locations are disclosed, then the last location is used.
- a.attackTime: It represents the time instant at which a happened.
- a.attackDistance: It represents the distance between a.attackLocation and a.attackTime.
- a.attackSpan: It represents the shortest sequence of the moves which caused a. If the attackspan is k, then truncating the path to the last k locations will still have the attack in the exact same form.
- a.hideSpan: It represents the span of the hidden locations used to avoid a – this depends on the particular hide policy employed.

Attacks are classified based on their ease of detection — detecting some leaks require more resources than others.
D-Attack: D-attack happens due to the current move and the last move where both the moves are in diagonally opposite cells and which discloses both the locations. As an example, for User-A in Figure 1, right after the third move, the locations at the second and the third moves become evident (block (9,4) at t_2 and (10,5) at t_3).
L-Attack: L-attack happens due to the current move and the last two moves where each of the three is in a different cell (usually, in L shape) and which discloses the last location. As an example, in Figure 1, the last three moves of User-B result into an L attack and the attacker will know that the User-B was at block (10,9) at time t_3.
Long-Attack: Any other type of leak is classified as a Long-attack. It normally requires larger number of moves. As an example, in Figure 1, the movement of user-C results into a long attack – after the t_7-th move, it is possible to determine the location at time t_3 (block (5,6)),t_4 (block (4,7)) and t_5 (block (5,8)).

D-attack and L-attack are easier to detect as they happen due to the last few steps. For this reason we name them as short attacks and use S to denote them. If the attack happened with move t, then the attack span for D-attack and L-attack are $[t-1,t]$ and $[t-2,t]$, respectively, which can be easily verified manually. However, detecting Long-attacks requires more effort and, in fact, its attack span, when attack happens in the t^{th} move, is $[t-\sigma-1,t]$. This is because of three facts from [3]: (1) attackLocation is always at boundary (2) attacks happen when crossing boundary (3) the path to which an attack belongs, must be shortest between the boundary blocks of that path in the cell containing the attack. As a result, any Long attack involves crossing two parallel boundaries, and the attack span includes the part of the path cutting across these two boundaries, which gives the above mentioned attack span as an upper bound.

Next, we consider the path provided by the user and consider the sequence of attacks it may have. We denote the user path by $uPath$ and an attack sequence of $uPath$ denotes one or more attacks in it in order of time. We denote attack sequences by $\mathcal{A}, \mathcal{A}'$ etc. A *complete attack sequence* of $uPath$, denoted by $\mathscr{C}$, represents all the attacks in it. As an example, consider User-D in firgure 2. In his $uPath$ of length 10, second location gets revealed at t_3 (L-attack), third location gets revealed at t_4 (L-attack), seventh, and eight locations gets revealed at t_9 (Long and L attack respectively) and ninth and tenth location gets revealed at t_{10} (D-attack). Therefore his complete attack sequence, $\mathscr{C}$ is

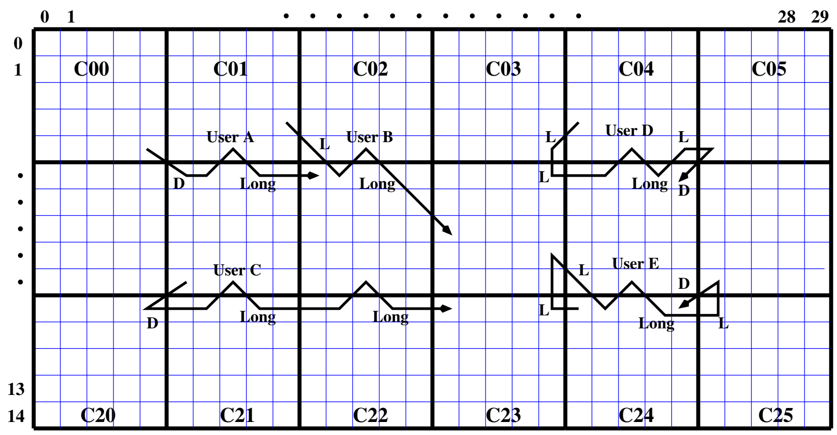


Fig. 2. Attacks in User's path in Disjoint Square Encoding of stretch 5

$L \cdot L \cdot Long \cdot L \cdot D$. For a $uPath$ there is a unique $\mathscr{C}$ but there can be more than one user paths having same $\mathscr{C}$. As an example, User-E in Figure 2 has a different $uPath$ than User-D but the same $\mathscr{C}$. An attack sequence $\mathcal{A}'$ is said to be an *attack subsequence* of $\mathcal{A}$, denoted by $\mathcal{A}' \preceq \mathcal{A}$, if all the attacks in $\mathcal{A}'$ are in $\mathcal{A}$ and have same order of occurrence in both. Two attacks in an attack sequence $\mathcal{A}$ are said to be *consecutive*, if there does not exist any attack subsequence of $\mathcal{A}$ in between them. There can be any number of moves in between any two consecutive attacks which might play a role in defining the rules to hide them. If necessary, we write the number of moves in between any two consecutive attacks with the first attack. For example, consider two consecutive attacks as a_1 and a_2 having k moves in between then we write it as $(a_1 + k) \cdot a_2$. For User-E in Figure 2 attack subsequence corresponding to first four moves can be written as $(L + 1) \cdot L$.

Definition 2 (Hiding policy). *A hiding policy is a well defined procedure to hide some locations in $uPath$ in order to avoid an attack sequence in it.*

4.1 Hiding Policy of Rule Based Approach

In the next section, we present our novel hiding policy which is based on the following observation. Short attacks are easy to hide – just by observation, individual D and L attacks can easily be circumvented by hiding one or two specific locations for each case. Even individual long attacks can be easily avoided on a case-by-case basis. Therefore, if the attacks were only short, and few and far between, then we can consider each short attack in tandem, and handle them individually. On the other hand, if one move is involved in multiple attacks, then it may be possible to reduce the number of hidden locations by cleverly choosing a smaller set of (common) locations; but more than that, there are scenarios in which the adversary, using his knowledge of the hiding policy, may be able to infer some common locations if multiple related attacks are handled individually. Our strategy for these is to *merge/transform* a few consequtive attacks, in a way, so that their hiding properties remain unchanged, and at the end we are left with only a few sparse attacks – which we can comfortably handle. We present the necessary definitions for this approach in the rest of this section.

Definition 3 (Separated attacks). *For a given hiding policy two consecutive attacks, say a and a', in an attack sequence are called separated, denoted by $a||a'$, if hiding one attack does not hide the other attack.*

Our hiding policy treats separated attacks by individually hiding them. If $\mathcal{L}$ and $\mathcal{L}'$ are hidden locations of $uPath$ necessary to hide attacks a and a' respectively, then the hidden locations for hiding $a||a'$ is $\mathcal{L} \cup \mathcal{L}'$.

Definition 4 (Separation normal form). *For a given hiding policy, an attack sequence is said to be in separation normal form, denoted by $\mathcal{A}_{SNF}$, if all the attacks in the sequence are separated.*

Definition 5 (Hide Equivalent). *Two attack subsequences $\mathcal{A}$ and $\mathcal{A}'$ for $uPath$ s.t. $\mathcal{A}' \preceq \mathcal{A}$ are said to be hide equivalent, denoted as $\mathcal{A} \sim_h \mathcal{A}'$, if under the hiding policy hiding $\mathcal{A}'$ also hides $\mathcal{A}$.*

The important property to observe is that hide equivalence is transitive.

5 Rule Based (RB) Approach

One obvious way to obtain a set of hidden locations is to check for all possible locations in $uPath$ in a brute-force manner. As we show later in Section 6, this method takes too long time even for paths of modest lengths. Therefore, we came up with a set of rules, and devised an efficient procedure which allows the proxy to process the attacks in $uPath$ and return a privacy-preserving path. The steps of the procedure are given next, and described in detail in the rest of this section.

1. Generate the complete attack sequence $\mathscr{C}$ from $uPath$.
2. By repeatedly removing non-separated attacks, transform $\mathscr{C}$ to $\mathscr{C}_{SNF}$ which only has separated attacks, s.t. $\mathscr{C} \sim_h \mathscr{C}_{SNF}$. This is achieved by these steps.
   (a) Scan $\mathscr{C}$ (left to right), identifying merged attack sequences and applying merged rule on them, to obtan $\mathscr{C}_1$ such that $\mathscr{C} \sim_h \mathscr{C}_1$.
   (b) Scan $\mathscr{C}_1$ (left to right), identifying contained attack sequences and applying merged rule on them, to obtain $\mathscr{C}_2$ such that $\mathscr{C}_1 \sim_h \mathscr{C}_2$.
   (c) Scan $\mathscr{C}_2$ left to right, identifying overlapped attack sequences and applying merged rule on them, to obtain $\mathscr{C}_3$ such that $\mathscr{C}_2 \sim_h \mathscr{C}_3$.
   (d) We will show that $\mathscr{C}_3$ is in SNF, so set $\mathscr{C}_{SNF} = \mathscr{C}_3$.
3. Apply hide rules for individual attacks to $\mathscr{C}_{SNF}$ to obtain a privacy preserving path.

The first step of obtaining $\mathscr{C}$ from $uPath$ can be implemented by a simple extension of [3, Algorithm 1]. We focus on the other steps in the next few subsections.

5.1 Hide Rules for Individual Attacks

We first discuss the hide rules for individual attacks, namely D-hide for D-Attack, L-hide for L-Attack and Long-hide for Long-Attack, based upon which the other rules will be devised. To discuss rules for short attack, consider the user path $p \cdot b_{t-2} \cdot b_{t-1} \cdot b_t \cdot p'$.

D-hide (at end): If the path ends at b_t, and if there is a D attack at time t, then D-hide is $p \cdot b_{t-2} \cdot * \cdot *$, i.e., hide $t-1^{th}$ and t^{th} move (therefore, $hideSpan = [t-1, t]$).

D-hide (intermediate): If the path does not end at b_{t-1} (b_t exists), and if there is a D attack at time $t-1$, then D-hide is $p \cdot b_{t-2} \cdot * \cdot * \cdot .p'$, i.e., hide$(t-1)^{th}$ and t^{th} move (therefore, $hideSpan = [t-1, t]$).

L-hide: If there is an L attack at time t, then L-hide is $p \cdot b_{t-2} \cdot * \cdot * \cdot p'$, i.e., hide $(t-1)^{th}$ and t^{th} move (therefore $hideSpan = [t-1, t]$).

The trick used in these hide rules is to create ambiguity between D and L attack so that no adversary can confidently find out which of them actually took place – this increases the number of possible blocks for the relevant moves. Take, for example, User-D in Figure 1 which has a D attack. After hiding locations denoted by $*$, an alternative path is shown by the dotted line (which corresponds to an L-attack). Similarly, if the user path is the dotted line (having an L attack), then L-hide leads to an alternative path given by the solid line (which corresponds to a D-attack). It is easy to see that, L and D attacks can not be saved by hiding only a single location (for example, if the policy is to hide only the attack location, then in the case of User-E in Figure 1 an adversary will be able to detect D-attack).

Long-hide: Consider a user path $p \cdot b_{t-\sigma-1} \cdot b_{t-\sigma} \dots b_{t-1} \cdot b_t \cdot p'$ with a Long attack at time t (by the property of Long attacks described earlier, the attack span of any long attack includes atmost $\sigma + 2$ last locations). Then, Long-hide rule is to hide alternate locations starting from $b_{t-\sigma-1}$ till b_t (therefore, set $hideSpan = [t-\sigma-1, t-1]$ if σ is even, and $hideSpan = [t-\sigma-1, t]$ if σ is odd).

The strategy of alternate hiding can be motivated by the example paths of User-F and User-G in Figure 1; hiding locations (marked by $*$) increased the possibility of disclosed moves as evident from alternative paths (shown by dotted line). Hiding any one or both of these locations is not sufficient, since the adversary, being aware of this rule, can still detect some locations. Long attack, therefore, requires alternate hiding.

5.2 Hide Rules for Non-separated Attacks

A sequence of attacks are *related* if hiding one attack will result into hiding of all other attacks in the sequence. For example in Figure 2, the sequence containing first two attacks are related in the attack sequence $\mathscr{C} = L \cdot L \cdot Long \cdot L \cdot D$ of User-D, as hiding first L-attack will result into hiding of the second L attack. Based on the nature of relationship, related attacks in an attack sequence could be of three types — a) merged b) contained, and c) overlapped. We will now discuss these attack sequences and form the rules required to hide them. Table 1 lists all these rules in one place.

Definition 6 (Merged attacks). *A sequence of attacks $a_i \cdot a_{i+1} \cdot \dots a_{i+j}$ is said to be merged attacks if they all occur due to one particular move.*

To denote merged attacks in an attack sequence $\mathscr{C}$, we write $\star$ in between the merged attacks, .i.e, if $\mathscr{C} = a_1 \cdot \ldots a_{i-1} \cdot a_i \star \ldots \star a_t \cdot a_{t+1} \cdot \mathcal{A}'$ then $a_i \star \ldots \star a_t$ is a merged attack sequence in $\mathscr{C}$. For example, in User-D's path in Figure 2, the third and the fourth attack (Long and L respectively) occur due to the move at t_9, and hence are merged attacks. There are no other merged attacks in $\mathscr{C}$ and so, we write $\mathscr{C} = L \cdot L \cdot Long \star L \cdot D$.

Lemma 2. *For merged attacks $a_i \star \ldots a_{j-1} \star a_j$, the following holds.*

1. *If at all the merged sequence contains a short attack, then it must be a_j.*
2. *$a_i.attackDistance < \sigma$.*

Lemma 3. *If $a_i \star \ldots \star a_{j-1} \star a_j$ and $b_k \star \ldots \star b_{l-1} \star b_l$ are two consecutive merged attack sequences in $\mathscr{C}$ at time t_1 and t_2 respectively, where $t_2 > t_1$, then*

1. *$a_j.attackTime < b_k.attackLocation$*
2. *$a_j.attackLocation < b_k.attackLocation$*

Lemma 3 tells us that any two consecutive merged attack sequences are physically apart in $\mathscr{C}$ and therefore we can handle them separately. Also from second part of Lemma 2 and Long-hide rule we conclude that in case of a merged-attack $a_i \star \ldots \star a_{j-1} \star a_j$ the whole sequence of attack can be hidden by considering the single attack a_i. This rule is denoted as the *merged rule* in Table1.

Definition 7 (Contained attacks). *For any two attacks a and a' we say a is contained in a', denoted by $a \subseteq a'$ if $a.attackSpan \subseteq a'.attackSpan$.*

For example, User-E in Figure 2 has $\mathscr{C} = L \cdot L \cdot Long \cdot L \cdot D$. The attack span of second attack (L-attack) is contained within the attack span of third attack (Long-attack) and hence, $L \subseteq Long$.

Lemma 4. *Following results explain some properties which hold for containment relationship.*

1. *One attack may contain the other attack only if they are consecutive.*
2. *For any two long attacks we have $Long_1 \subseteq Long_2$ iff $Long_2 \subseteq Long_1$*
3. *For an attack sequence $Long \cdot a$ we have $Long \supseteq a$ iff $Long \star a$*
4. *For $a_1 \star a_2 \star \ldots \star a_{n-1} \star a_n$, we have*
   (a) $a_1 \supseteq a_2 \supseteq \ldots \supseteq a_{n-1} \supseteq a_n$
   (b) $a_1 \star \ldots \star a_{n-1} \star a_n \sim_h a_1$

Proof. Proof of above lemma is direct from the definition of the containment relation and attack spans of various attacks.

Part(3) of Lemma 4 shows that if we have *contained* relationship between two attacks whereas first attack is a long attack then they will always have a *merged* relationship. Further Part(4) of the Lemma 4 says that attacks having *merged* relationship will always have *contained* relationship. However the converse is not true. For example, consider user User-A in Figure 2. His attack sequence $\mathscr{C}$ is $D \cdot Long$. In this case, we have *contained* relationship between the attacks. But there is no *merged* relationship between the two. These kind of attack sequences which are not covered through the merged rules, can be handled by the *contained rule*: $S \cdot Long \sim_h Long$ if $S \subseteq Long$.

The other possible relation between consecutive attacks is due to either (1) hide span of one attack overlapping the attack span/location of the other attack or (2) attack spans of two attacks overlapping each other. We call such attacks as *overlapped* attacks.

Definition 8 (Overlapped attacks). *For any two consecutive attacks a_1 and a_2,*

1. *If a_2 is a short attack and $a_1.hideSpan \cap a_2.attackSpan \neq \phi$ then we say a_1 overlaps a_2.*
2. *If a_1 is a short attack, a_2 is a long attack and $a_1.attackLocation \in a_2.hideSpan$ then we say a_2 overlaps a_1.*
3. *If a_1, a_2 are both long attacks and $a_1.attackSpan \cap a_2.attackSpan \neq \phi$ then we say a_1 overlaps a_2.*

Two attacks are overlapped if at least one attack overlaps the other attack. If a_1 overlaps a_2, we denote it by $a_1 \hookrightarrow a_2$.

An example of the case when a_1 is short and a_2 is long is User-C in Figure 2. For the first two attacks (D and Long respectively), attack location of the D-attack is contained within the attack span of the Long-attack. Therefore, Long-attack overlaps the D-attack.

Lemma 5. *The following properties hold for the overlapped attacks.*

1. *Two contained attacks are always overlapped.*
2. *Two consecutive attacks are separated iff they are not overlapped.*

If a $Long$ attack overlaps another $Long$ attack, then we call this as $2Long$. In general, $Long_1 \hookrightarrow Long_2 \hookrightarrow Long_3 \ldots \hookrightarrow Long_n$ is $nLong$. i.e we consider a series of overlapped long attacks as a single long attack (named $nLong$) s.t.:

$$nLong.attackSpan = [t - \Sigma - 1, t] \quad nLong.hideSpan = \begin{cases} [t - \Sigma - 1, t - 1] & \Sigma \text{ even} \\ [t - \Sigma - 1, t] & \Sigma \text{ odd} \end{cases}$$

where $\Sigma = Long_n.attackTime - [Long_1.attackTime - \sigma - 1] - 1$.

From second part of Lemma 5, we know that for any two consecutive attacks a_1 and a_2 s.t $a_1 \not\hookrightarrow a_2$ and $a_1 \not\hookleftarrow a_2$ we have $a_1 \| a_2$. This helps in (1) separating the attacks which are not overlapped, and (2) determining the hide equivalent subsequence for the attack sequence having overlapped relationship. The rules are given as *overlapped rules* in Table 1. Unlike the other two rules, these rules have to be *necessarily* applied from left-to-right, since the overlap relation is not symmetric (for example, $D \hookrightarrow L$ but $D \not\hookleftarrow L$ for the attack sequence $(D + 1) \cdot L$).

5.3 Time Complexity

RB-approach will take $O(\sigma n)$ time, which is practically linear since σ is constant for a fixed layout. This can be obtained through the following analysis of the RB procedure.

Table 1. Hide Equivalence Rules

Merged Rules	**Contained Rules**
$a_i \star a_{i+1} \star \ldots \star a_{j-1} \star a_j\cdot \sim_h a_i\cdot$	$S \cdot Long\cdot \sim_h Long\cdot$ if $S \subseteq Long$
Overlapped Rules	
Case O1: For $\mathbf{a_1} \cdot \mathbf{a_2}\cdot$ such that $\mathbf{a_1} \hookrightarrow \mathbf{a_2}$	**Case O3: For $\mathbf{a_1} \cdot \mathbf{a_2} \cdot \mathbf{a_3}\cdot$ such that $\mathbf{a_1} \hookrightarrow \mathbf{a_2} \hookrightarrow \mathbf{a_3}$**
$D \cdot S\cdot \sim_h D\cdot$	$D_1 \cdot D_2 \cdot S\cdot \sim_h D_1\Vert$
$(D+1) \cdot S\cdot \sim_h D\Vert$	$S \cdot L \cdot D\cdot \sim_h S\Vert D\cdot$
$L \cdot D\cdot \sim_h L\Vert$	$S \cdot L_1 \cdot L_2\cdot \sim_h S\Vert$
$L_1 \cdot L_2\cdot \sim_h L_1\cdot$	**Case O4: For $\mathbf{a_1} \cdot \mathbf{a_2} \cdot$ Long$\cdot$ such that $\mathbf{a_1} \hookrightarrow \mathbf{a_2} \hookleftarrow$ Long & $\mathbf{a_2} \not\subseteq$ Long**
$(L+1) \cdot S\cdot \sim_h L\Vert S\cdot$	$D \cdot S \cdot Long\cdot \sim_h D\Vert Long\cdot$
$S_1 \cdot (S_2+1)\cdot \sim_h S_1\Vert$	$L_1 \cdot L_2 \cdot Long\cdot \sim_h L_1\Vert Long\cdot$
$Long \cdot Long\cdot \sim_h 2Long\cdot$	**Case O5: For nLong $\cdot\, \mathbf{a_1} \cdot \mathbf{a_2}\cdot$ such that nLong $\hookrightarrow \mathbf{a_1} \hookrightarrow \mathbf{a_2}$ & nLong $\not\supseteq \mathbf{a_1}$ & $\mathbf{\Sigma}$ is odd**
$(Long_1+1) \cdot Long_2\cdot \sim_h Long_1\Vert Long_2\cdot$	
$Long_1 \cdot (Long_2+1)\cdot \sim_h 2Long\Vert$	
$nLong \cdot S\cdot \sim_h \begin{cases} nLong\Vert S\cdot & \text{if } \Sigma \text{ even} \\ nLong\cdot & \text{if } \Sigma \text{ odd} \end{cases}$	$nLong \cdot D \cdot S\cdot \sim_h nLong\Vert S\cdot$
Case O2: For $\mathbf{a_1} \cdot \mathbf{a_2}\cdot$ such that $\mathbf{a_1} \hookleftarrow \mathbf{a_2}$	$nLong \cdot L_1 \cdot L_2\cdot \sim_h nLong\Vert$
$S \cdot Long\cdot \sim_h Long\cdot$	$nLong \cdot L \cdot D\cdot \sim_h nLong\Vert D\cdot$

1. Time complexity to get $\mathscr{C}$ from $uPath$ is $O(\sigma n)$. After every move, the possibilities of previous $\sigma - 1$ moves need to be updated (Lemma[1], Update Lemma[3]). Whenever the possibility for a move becomes 1, we report it as an attack and include it in the attack list. This gives us $\mathscr{C}$.
2. Reducing $\mathscr{C}$ into $\mathscr{C}_3$ takes $O(k)$ time, where k is $len(\mathscr{C})$. This is because step 2(a), 2(b) and 2(c) take $O(k)$ time as these steps apply merged, contained and overlap rules on an attack sequence, which require only linear scan from left to right.
3. Hiding step takes $O(k)$ time as it applies the hide rules over the attack sequence obtained after step 2.

5.4 Correctness

Lemma 6. *$\mathscr{C}_3$, defined as above, is in SNF.*

Proof. The application of merged,contained and overlapped rules in the order as described in the procedure above results in an attack subsequence having no overlapped attacks. Therefore, from second part of Lemma 5 all attacks in $\mathscr{C}_3$ are separated.

First, it is clear that given the entire path p, the generated cell-path $G(p)$ (with hidden locations) is such that no adversary can deduce the current or any past location with absolute certainty. Furthermore, the RB approach applies it hiding rule in a left-to-right manner, which ensures that given any prefix p' of p, the corresponding prefix of $G(p)$ is also privacy preserving. If the attacks are separated then this is obvious. When attacks are related to each other and hiding one attack saves an attack sequence then hidden locations for that attack will always hide one or more moves required for the attacks in the attack sequence. Therefore there will be no disclosure in any prefix of the path, i.e., at any time $t \leq |p|$, there are at least two or more possible blocks for the i-th location, for

$i \leq t$. Thus, the user is assured that no disclosure of current or past location will happen anytime along p if the hide rules are followed and the resultant obfuscated regions are reported all along the path.

5.5 Change of Plans: Real-Time Modifications to User Path

The stipulation of submitting the planned path at the beginning and following it thereafter begs one important question: how far can the user deviate from the initial plan?

The explanation on correctness in Section 5.4 basically says, that the user may stop anytime during travel without any change wrt. privacy. On the other hand, our approach also allows the user to *extend* his path in a way which allows incremental computation of hidden locations. Recall that the number of moves involved in a Long attack is $\sigma+1$ (and fewer for other attacks), so the user is allowed to modify his path, including extending it, as long as the modification starts σ or more time instants after. RB approach will just have to compute the hidden locations for the new path starting from the current instant. Note that, the requirement of at least σ moves is necessary as can be seen for User-H in Figure 1. If the user wants to extend his path by one more block when he is at the last step, then the new path will actually cause an Long-attack; and, the optimum way to hide this Long-attack will require hiding alternate blocks starting from the beginning of this path, which cannot be done at the last step.

6 Experimental Results

In this section we examine the efficiency of our RB Approach in achieving a privacy preserving path. We compared this approach with the brute-force approach. The brute-force approach tries all possible combinations of locations to hide and returns the combination with the minimum number of hidden locations. The experiment is run for 100 random paths of length 10–50 on cells of stretch 6. (Many of these paths did not have any attack and do not appear in the results below.) All experiments were conducted on a standard dual-core Pentium desktop computer with normal workload.

Our observation was that the RB approach returned more hidden locations for some paths compared to the brute-force approach, but, in significantly less time. Fewer hidden locations is important as the user would like to get service at maximum locations (albeit, without any privacy violation). Therefore our focus of comparison was on these two metric – number of hidden locations and time of execution.

We summarise our findings in the Figures 3 and 4. Observe that, in almost all cases, RB approach hides at most 4 times the number of locations hidden by the brute-force approach. The increase in hidden locations is perfectly complemented by the running time — even for paths that took more than 2-3 hours using the brute-force approach, RB approach computed the hidden locations in less than a minute. For better understanding, we have classified the random paths into long dominant (60% of the attacks are Long attacks), short dominant (60% of the attacks are Short attacks) and mixed (rest). We found that even though for long dominant paths, RB suggested about 4 times more hidden locations (on an average), for short dominant paths, RB suggested only about twice as that by the brute-force.

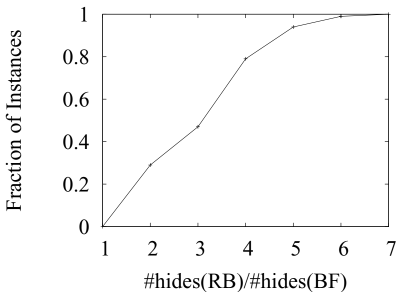


Fig. 3. CDF of the ratio of number of hidden locations returned by RB to the optimum number of hidden locations

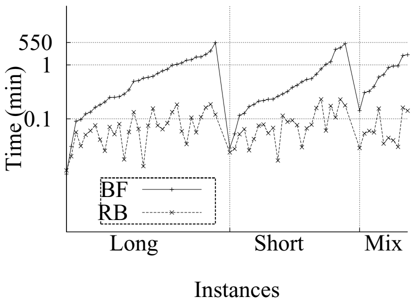


Fig. 4. Comparison of execution time (given in log-scale) between RB approach and brute-force. The instances are classified as Long-dominant, Short-dominant and Mixed.

7 Conclusion

In this paper we have discussed the location privacy issues associated with a continuous query scenario with the privacy module having advance knowledge of user paths. To address the issue of privacy breach in this scenario, we have provided a deterministic approach named RB approach based on the attack classification as defined in [3]. This approach enables us to not only provide relationships between attacks in an attack sequence but also avoid them in a systematic manner by hiding locations according to the hide equivalence rules provided in the paper. We have also shown that our RB approach to get a privacy preserving path takes effectively linear time; moreover, the loss of service quality is upper bounded by a constant when compared to the optimum.

Acknowledgement. The first author was supported by Prof. Pankaj Jalote and the third author was supported by the Department of Science and Technology, Govt. of India for this work.

References

1. Bamba, B., Liu, L., Pesti, P., Wang, T.: Supporting anonymous location queries in mobile environments with privacygrid. In: WWW 2008: Proceeding of the 17th International Conference on World Wide Web, pp. 237–246 (2008)
2. Barkhuus, L., Dey, A.: Location-Based Services for Mobile Telephony: a Study of Users' Privacy Concerns. In: INTERACT 2003: 9th IFIP TC13 International Conference on Human-Computer Interaction, pp. 709–712 (2003)
3. Bera, D., Goyal, V., Saxena, A.S.: Privacy of location obfuscation. Technical Report IIITD-TR-2011-002, IIIT-Delhi (2011)
4. Beresford, A.R., Stajano, F.: Location Privacy in Pervasive Computing. IEEE Pervasive Computing 2, 46–55 (2003)
5. Beresford, A.R., Stajano, F.: Mix zones: User privacy in location-aware services. In: PERCOMW 2004: 2nd IEEE International Conference on Pervasive Computing and Communications, pp. 127–131 (2004)

6. Bettini, C., Mascetti, S., Wang, X.S., Jajodia, S.: Anonymity in Location-Based Services: Towards a General Framework. In: MDM 2007: Proceedings of the International Conference on Mobile Data Management, pp. 69–76 (2007)
7. Burak, A., Sharon, T.: Usage patterns of FriendZone: mobile location-based community services. In: MUM 2004: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, pp. 93–100 (2004)
8. Chow, C.-Y., Mokbel, M.F.: Enabling Private Continuous Queries for Revealed User Locations. In: Papadias, D., Zhang, D., Kollios, G. (eds.) SSTD 2007. LNCS, vol. 4605, pp. 258–275. Springer, Heidelberg (2007)
9. Chow, C.Y., Mokbel, M.F., He, T.: Tinycasper: a privacy-preserving aggregate location monitoring system in wireless sensor networks. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1307–1310 (2008)
10. Gedik, B., Liu, L.: Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms. IEEE TMC: IEEE Transactions on Mobile Computing 7, 1–18 (2008)
11. Gruteser, M., Grunwald, D.: Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In: MobiSys 2003: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, pp. 31–42 (2003)
12. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM TKDD: ACM Transaction on Knowledge Discovery in Databases 1 (2007)
13. Martin, D.J., Kifer, D., Machanavajjhala, A., Gehrke, J., Halpern, J.Y.: Worst-Case Background Knowledge for Privacy-Preserving Data Publishing. In: ICDE 2007: 23rd International Conference on Data Engineering, pp. 126–135 (2007)
14. Mokbel, M.F., Chow, C.Y., Aref, W.G.: The new Casper: query processing for location services without compromising privacy. In: VLDB 2006: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 763–774 (2006)
15. cellular news: Mobile Subscribers to Hit 5.9 Billion in, Driven by China, India, Africa (2013), http://www.cellular-news.com/story/40439.php?s=h
16. Palanisamy, B., Liu, L.: Mobimix: Protecting location privacy with mix-zones over road networks. In: 27th ICDE 2011, pp. 494–505 (2011)
17. Sweeney, L.: Achieving k-Anonymity Privacy Protection using Generalization and Suppression. International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems 10, 571–588 (2002)
18. Voelcker, J.: Stalked by satellite: An alarming rise in GPS-enabled harassment. IEEE Spectrum 7, 15–16 (2006)
19. Xu, T., Cai, Y.: Exploring historical location data for anonymity preservation in location-based services. In: INFOCOM 2008: Proceeding of 27th Conference on Computer Communications, pp. 547–555 (2008)

A Data Mining Framework for Securing 3G Core Network from GTP Fuzzing Attacks

Faraz Ahmed, M. Zubair Rafique, and Muhammad Abulaish

Center of Excellence in Information Assurance (CoEIA)
King Saud University (KSU)
Riyadh, Saudi Arabia
{fahmed.c,zrafique.c,mabulaish}@ksu.edu.sa

Abstract. Since the emergence of 3G cellular IP networks, internet usage via 3G data services has become ubiquitous. Therefore such network is an important target for imposters who can disrupt the internet services by attacking the network core, thereby causing significant revenue losses to mobile operators. GPRS Tunneling Protocol GTP is the primary protocol used between the 3G core network nodes. In this paper, we present the design of a multi-layer framework to detect fuzzing attacks targeted to GTP control (GTP-C) packets. The framework analyzes each type of GTP-C packet separately for feature extraction, by implementing a Markov state space model at the G_n interface of the 3G core network. The Multi-layered architecture utilizes standard data mining algorithms for classification. Our analysis is based on real world network traffic collected at the G_n interface. The analysis results show that for only 5% fuzzing introduced in a packet with average size of 85 bytes, the framework detects fuzzing in GTP-C packets with 99.9% detection accuracy and 0.01% false alarm rate.

Keywords: Intrusion Detection, Fuzzing attacks, GTP Security.

1 Introduction

Connecting millions of people around the globe and providing exciting services to end users, the demand of internet is ever rising [1]. Every effort has been made to improve the user experience and to increase the Internet's circle. Cellular networks provides only voice services but, with the advent of 3G technologies mobile operators are providing data services with low broadband speed. The main reason for the popularity of 3G network is its ability to provide greater bandwidth with wide area coverage. Several data transmission techniques have been proposed for better performance, e.g., WCDMA, TD/CDMA and CDMA2000 are different Code Division Multiple Access techniques used for data transmission. The first two techniques are based on General Packet Radio Service (GPRS) and hence have the same core network architecture [2]. Our framework targets the security of GPRS core network interface, i.e., G_n interface.

As compared to the widespread use of internet via cellular network there is a huge threat to the security of the network. Attacks can come from inside the cellular network [3]. These attacks can cause network degradation and eventually lead to `Denial of Service` (DoS) to end users. However, 3G networks have some of their own security issues as addressed in [4] and [5]. Due to open nature of IP in 3G networks, attackers can exploit vulnerabilities in their core network nodes and protocols. Attacks on the core nodes of the 3G networks can be launched by compromising different nodes of the architecture such as the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN) [6]. As explained in [7], an attacker can establish herself as a legitimate 3G network element by IP spoofing. Such attacks pose serious threat to mobile user privacy by stealing user data such as $IMSI$ number, billing information, contact details, etc. An attacker can exploit protocol vulnerabilities by fuzzing sensitive fields of packet headers [8]. GPRS Tunneling Protocol GTP is the main communication protocol used in the core network. All user requests for internet services are made through GTP.

In this paper, we have analyzed the GTP protocol vulnerabilities and proposed an effective and efficient multi-layered framework for their mitigation. Our analysis is based on real world GTP (v1) traffic collected at the G_n interface. The main contribution of our work is a framework that can detect GTP-C fuzzing attacks in real time. It consists of three main modules: i) Packet Byte Analyzer (PBA), ii) Benign Packet Definition (BPD), and iii) Decision Module. The fuzzing is detected by modeling the differences in byte sequences of normal and fuzzed packets. We use Markov state-space model for extracting features. The less discriminative features are then pruned by using an information theoretic measure known as `Information Gain`. Each incoming packet is fed to BPA which performs the feature extraction and forwards them to the BPD module. The BPD module uses the extracted feature set as input and represents each packet as a feature vector. The decision module implements standard data mining algorithms to classify the incoming packets as normal or malformed. The rest of the paper is organized as follows. Section 2 gives a brief summary of the related works. In Section 4, we report different statistics of our real world benign dataset. Section 5 presents the architectural detail of the proposed framework. Section 6 presents the experimental setup and results. Finally in section 7, we conclude the paper with future directions of work.

2 Related Work

The attacks in the cellular networks are not unprecedented. Some known attacks are directed towards Mobile Stations (MSs) [9] and [10] whereas, some attacks try to disrupt the services in general as mentioned in [11] and [3]. [12] presents a taxonomy of such 3G attacks. The attacks have been classified as *Cross-Infrastructure*, which are directed from the internet to the cellular networks, and *Single Infrastructure* attacks which arise from within a cellular network. In [13], Patrick et al. holds the opposite design philosophies of internet and 3G networks responsible for making 3G networks vulnerable to Denial of Service (DoS) attacks, and also demonstrates two more attacks supporting this theory. The author highlights the fact that bandwidth is not the ultimate cause of such attacks rather, it is the inflexibility of architecture of 3G networks that makes these attacks practical.

One of the foremost attempt to highlight the vulnerabilities of GPRS core network is presented in [4]. In this work, the author has provided an overview of attacks and flaws associated with GPRS architecture. The report also provides recommendations to avoid such type of attacks. A more detailed categorization of attacks against GPRS is followed in [8]. In this paper, the authors have listed Overbilling attacks, misconfigured WAP's exploits and a detailed list of GTP risks. The paper proposes an alternative design for network architecture that can be adopted by network operators. The authors also present `Check point Firewall` product that can provide additional security.

Another important contribution in securing GPRS from attacks on the GPRS core is presented in [6]. Dmitriadis et al. presents a threat model with regard to GPRS core network, depicting nine possible attack groups, and also gives a feasibility study of honeynets in 3G networks. The authors propose `3GHNET`, a honeynet, for the improvement of GPRS core network security. The authors have compared the advantage of 3GHNET implemented GPRS network over an unprotected network and used concepts from the game theory for comparison.

[2] presents a defense mechanism for GTP security threats. The authors propose an event-based description language for the detection of attacks directed towards the GTP protocol. They have classified GTP security concerns as protocol abnormal attacks, infrastructure attacks and resource consumption attacks. They have categorized the GTP protocol into GTP-C, GTP-U and GTP', which are GTP control plane, GTP user plane and GTP prime respectively and analyzed them separately to perform the decision on the basis of events generated. The authors have tested their architecture on OpenGGSN emulator which is an open source implementation of the core network nodes - SGSN and GGSN [14]. Our work is different from [2] as it aims at securing only the GTP-C category of the GTP protocol from fuzzing attacks. GTP-C packets are most important for the communication between the GSNs. The architecture of our scheme enables us to further categorize the GTP-C packets and analyze them separately.

3 GPRS Architecture

GPRS is an extension GSM, in fact it has been overlaid on the already existing GSM infrastructure [15]. To handle packet data, a Packet Control Unit(PCU) is introduced at Base Transceiver Station(BTS). Besides that two GPRS support nodes(GSNs) have been added to the structure. SGSN is connected with many BTSs analogous to BSC, and serves to transfer data requests over the network. Whereas GGSN facilitates to connect the network to external data network. Any user that intends to send/receive data from external network has to register a context with these two nodes(SGSN and GGSN). The different interfaces of GPRS are shown in Figure 5.

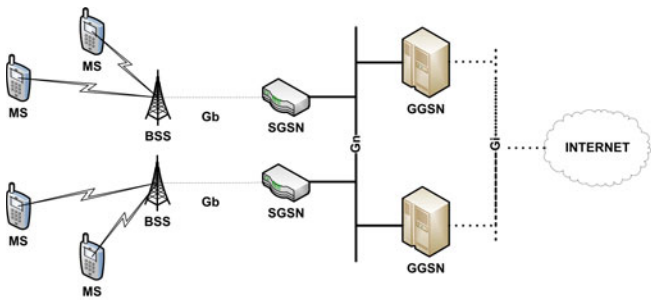
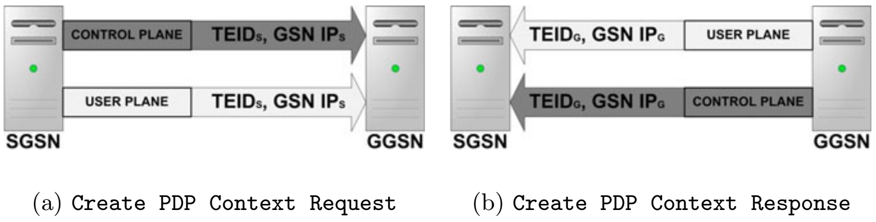


Fig. 1. Architecture of GPRS

The next section is dedicated to the description of this interface, and depicts how communication actually takes place on this interface. For the sake of brevity, we have only considered GTPv1 specifications for the matter at hand.

3.1 G_n Interface

Whenever a user needs to send/receive packet data from external network, it requests the network to activate a PDP context. On receiving such a request, the SGSN sends a `Create PDP context Request` message containing IMSI number of the user,(Access Point NAme) APN and Tunnel Endpoint Identifiers (TEID) for GTP-C and GTP-U plane, to GGSN. Once the GGSN receives this information, it stores it for future correspondence and sends back `Create PDP Context Response` containing information elements(IEs to indicate wether the context was established successfully), End User Address field (which contains the IP address assigned by the GGSN to the user) and TEID for both GTP-C and GTP-U plane.



(a) `Create PDP Context Request` (b) `Create PDP Context Response`

Fig. 2. Context Establishment between SGSN and GGSN

Figure 2 demonstrates how a context is established between the two nodes, and how do SGSN and GGSN recognize tunnels at their ends, both in User and Control plane. When SGSN sends a `Create PDP context Request` to the GGSN as shown in Figure 2(a), it advertises a $TEID_S$ and an IP_S address for User plane and a $TEID_S$ and an IP_S(subscript S is used for SGSN) for

Table 1. Benign dataset summary

Type	No.	Avg. Size(Bytes)	Description
Create PDP Request	1183681	197	Request for initiation of user session
Create PDP Response	3866	135	Response to the initiation request
Update PDP Request	555	85	Request to update the QoS, TFT etc parameters
Update PDP Response	684	95	Response to the update parameter request
Delete PDP Request	4317	60	Request for termination of user session
Delete PDP Response	3237	56	Response to the termination request

Control plane to the GGSN, to be used in future by the GGSN when addressing the specified tunnel at SGSN. SGSN uses the same parameters(the $TEID_S/IP_S$ that it advertised) to discern between different tunnels operating at SGSN. Similarly, when GGSN responds with a `Create PDP context Response` message as shown in Figure 2(b), it advertises a $TEID_G$ and IP_G for User plane as well as for the Control plane to the SGSN, which are to be used in future by the SGSN when addressing a specific tunnel at GGSN. The GGSN uses these parameters to discern between different tunnels operating at GGSN. Also, the port numbers are fixed for both Control and User plane data. Similar to the `Create PDP Context Request/Response` messages, `Delete PDP Request/Response` messages also exist, which are used to delete an active tunnel. Since the payload of user is tunneled through the G_n interface, it becomes a natural choice for analysis when it comes to anomaly/intrusion detection in the core network. A compromised SGSN or GGSN can host attacks to other critical systems, such as the Mobile Switching Center (MSC), home location register (HLR), visitor location register (VLR) and other SGSN/GGSN nodes of the network. Such attacks directly affect crucial information such as subscriber identity database residing in the HLR, charging/ billing gateways (CG/BG), handoff operations which involves VLR etc.

4 Dataset

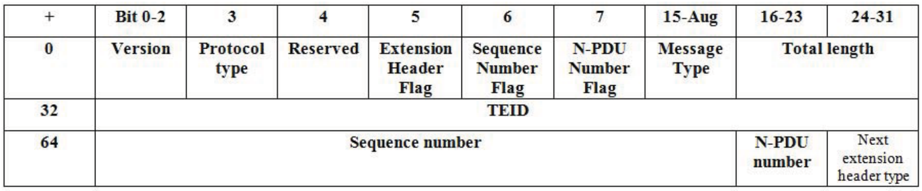
In this section we describe the benign and malformed GTP dataset that we have used in this study. We also give a brief description of our fuzzing algorithm used to generate malformed GTP packets.

4.1 Benign Traffic

Our benign dataset consists of real world GTP-v1 traffic collected at the G_n interface. The traffic was logged at GPRS core network `node`, during the peak usage hours of the day. All type of GTP packets were captured however, our analysis is based on only GTP-C packets, which are responsible for the creation and deletion of user sessions between the GSNs. Table 1 provides different statistics of the data set. The total number of PDP contexts shows the number of GTP tunnels created, updated or deleted between the SGSN and the GGSN. It is obvious that there are unequal number of requests and responses, which is due to window censoring phenomenon [16]. This means that user sessions initiated during the data logging period are not torn down before the end of the logging process.

4.2 Fuzzed Dataset

We performed fuzzing of each type of GTP-C packet separately. The format of the GTP packets is shown in Figure 3. For fuzzing, we have employed standard bit-fuzzing technique used for other IP-based protocols, i.e., for 1% fuzzing a bit is randomly selected from a packet and is inverted. Similarly for n% fuzzing, we select n% bits randomly from a packet and invert them. In this way, we have generated 24 different fuzzed datasets for each GTP-C packet category corresponding to 2%, 5%, 10% and 20% fuzzing of each n-gram where, n varies from 1 to 6.



+	Bit 0-2	3	4	5	6	7	15-Aug	16-23	24-31
0	Version	Protocol type	Reserved	Extension Header Flag	Sequence Number Flag	N-PDU Number Flag	Message Type	Total length	
32	TEID								
64	Sequence number							N-PDU number	Next extension header type

Fig. 3. GTP packet format

Our fuzzed dataset consists of packets with fuzzed fields such as `message type` field. Fuzzing this type of field changes the message type, for example, from `Create PDP Context Request` message(message type=0x10) to some other message type, which may result in a message type that is not recognizable by the GGSN or in a message type that GGSN is not expected to receive. In addition, there are some information elements following the mandatory header in the message that are more apposite for fuzzing. This is because each type of packet uses the extension header information elements differently. More specifically, the information elements(IEs) are divided into TV (Type, Value) or TLV (Type,Length,Value) format. Figure 4 shows details of the formatting of such IEs. Our fuzzed dataset include packets with fuzzed TV-formatted IE's because when we fuzz such a field, the length of the fuzzed field may increase from that of the expected length known to the GGSN, making the IEs following it to be unreadable. The fuzzed packet dataset also contains fuzzed values of TLV-formatted IEs fields, end user address, access point name (APN), protocol configuration options (PCO) and GPRS serving node (GSN) address IEs. Table 2 describes the possible impact of fuzzing different fields of GTP packet.

Table 2. Fuzzed fields and possible results of fuzzing

Fuzzed Field	Explanation	Result
Message Type	Allows 255 different message types values	Invalid message type
IE	Contain packet specific information	DoS/Dependent on Device Vulnerability
IE length	Contains the length of IE	Buffer overflow/System Crashes
End User Address	Address of the Mobile Station	DoS/Dependent on Device Vulnerability

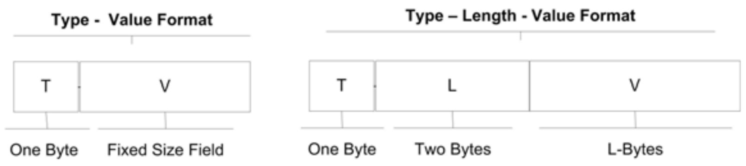


Fig. 4. Information element formats

5 GTP Malformed Packet Detection Framework

In this section, we present the architectural detail of the proposed intrusion detection framework, which consists of a bi-directional detection module at the G_n interface. Figure 5 shows the architecture of the proposed framework for detection of malformed GTP packets. GTP protocol is used by most of the 3G transmission techniques including WCDMA and TD/CDMA, which employ the GPRS core network architecture. So for simplicity we consider the GPRS network for explanation of the proposed framework. SGSN is connected with many Base Transceiver Stations (BTSs), and serves to transfer data requests over the network. Whereas GGSN facilitates to connect the network to external data network. The architecture secures the control plane of the GTP protocol by employing a parallel design. The parallel architecture has two main advantages. Firstly, it reduces the processing overhead by the simultaneous analysis of different GTP control packets and secondly, it allows a deeper level of inspection by analyzing each packet type according to its use of extension headers as explained in section 4.2. The detection framework perform byte-level analysis of the incoming GTP-C packets and classify them as normal or malformed. The proposed framework consists of three main modules - Packet Byte Analyzer, Benign Packet Definitions, and Decision module. A detailed description of these modules appears in the following sub-sections.

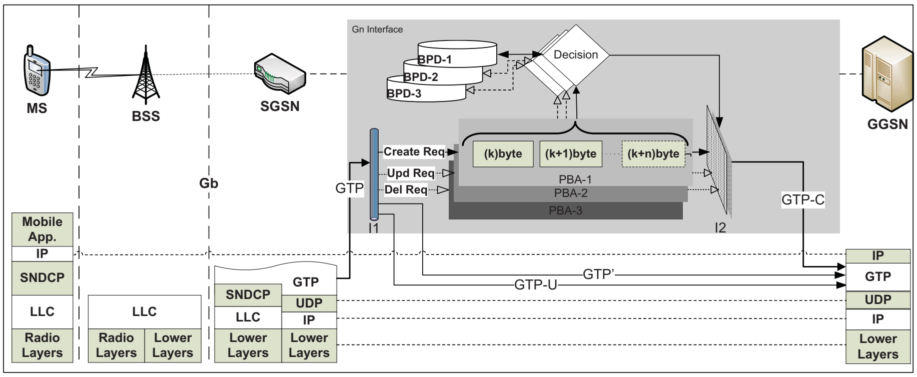


Fig. 5. Architecture of proposed framework for GTP fuzzing attacks

5.1 Packet Byte Analyzer

The PBA module acts separately for each type of GTP control packet. Its inputs are the validated GTP packets. The validation process is done through an input interface, which checks input packets for explicit errors like invalid message type. For each packet, it performs a byte-level analysis. The module uses a windowing methodology for the collection of important discriminating features. It implements a sliding window of n bytes. Given a byte sequence P_s, of a packet P, sliding a window of size $n = 1$, we get $P_s =< \Phi_1, \Phi_2, ..., \Phi_i, .. >$, where Φ_i represents the i^{th} byte of P. Similarly for $n = 2$ the representation becomes $P_s =< \Phi_1|\Phi_2, \Phi_2|\Phi_3..., \Phi_{i-1}|\Phi_i, .. >$, where the symbol $|$ represents a string concatenation operator. This relation explains the tradeoff that exists between the amount of information and the size of the training data. Therefore, a thorough analysis for the selection of the window size is necessary for better performance. Accordingly, we model the byte sequences so that analysis can be performed with varying window size. For this, we use discrete time Markov chain. We consider the position of the window ($size = n$) as a state which changes in accordance with the window slides. Therefore, for a representation P_s if $S = s_0, s_1, ..., s_k$ is the set of possible states, then the position to state mapping function can be described as: $(f : p_i \rightarrow s_j \in S)$ where, $p_i \in P =< p_0, p_1, ..., p_m >$. So for two consecutive window positions the mapping functions are $(f : p_i \rightarrow s_x)$ and $(f : p_{i+1} \rightarrow s_y)$. The transition between two states is represented as s_{xy} and that the transition probability as τ_{xy}. This gives a state transition probability matrix calculated as $F : S \times P \rightarrow \tau(S)$ where, F is a transition function. The PBA computes $\tau(S)$ for each packet and outputs the probability matrix which is used by the decision module.

5.2 Benign Packet Definitions

This module is used to model incoming data into an n-dimensional feature space where, n represents the number of features identified by PBA module. n varies for different types of control packets depending on the number and size of the packet type. During training phase the PBA calculates transition probabilities of the training dataset. Each transition probability is considered as a potential feature which can help in discriminating normal packets from malformed packets. So, during training phase six different feature vector sets are created one for each packet type.

5.3 Decision Module

The decision module implements three classifiers: Decision tree (J48), Naïve Bayes (NB) and inductive rule learner (Jrip). The module takes $\tau(S)$ as an input from the PBA and on the basis of training dataset residing in the respective BPD and generates the output for output filter. A brief description of the three classifiers used is presented in the following paragraphs.

Decision Tree (J48). Decisions trees are usually used to map observations about an item to conclusions about the items target value using some predictive model [17]. They are very easy to understand and are efficient in terms of time especially on large datasets. They can be applied on both numerical and categorical data, and statistical validation of the results is also possible. We use $C4.5$ decision tree (J48) that is implemented in WEKA. We do not utilize binary splits on nominal attributes for building trees. The confidence factor for pruning is set to 0.25, where lower values lead to more pruning. The minimum number of instances per leaf is set to 2. The number of folds of training data is set to 3, where one fold is used for pruning and the rest are used for growing the tree.

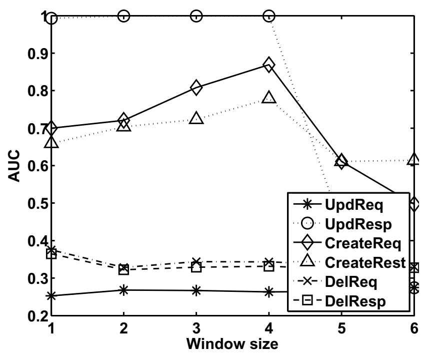
Naïve Bayes (NB). Naïve Bayes is a simple probabilistic classifier assuming naïve independence among the features, i.e., the presence or absence of a feature does not affect any other feature [18]. The algorithm works efficiently when trained in a supervised learning environment. Due to its inherent simple structure it often gives very good performance in complex real world scenarios. The maximum likelihood technique is used for parameter estimation of Naïve Bayes models. We have neither used kernel estimator functions nor numeric attributes for supervised discrimination that converts numeric attributes to nominal ones.

Inductive Rule Learner (Jrip). We chose rule based learners due to their inherent simplicity that results in a better understanding of their learner model. Jrip, performs quite efficiently on large noisy datasets with hundreds of thousands of examples.The algorithm works by initially making a detection model composed of rules which are improved iteratively using different heuristic techniques. The constructed rule set is used to classify the test cases.

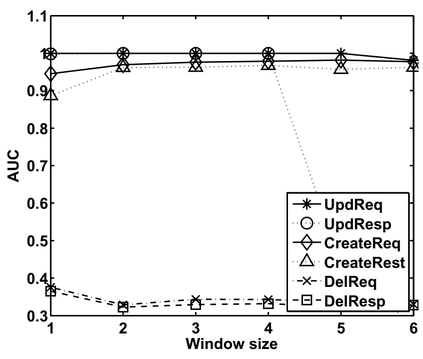
6 Experiments and Results

In this section we evaluate the performance of the proposed GTP-C fuzzing detection framework. We measure the performance on the basis of detection rate. We have carried out the standard Receiver Operating Characteristics (ROC) analysis to evaluate the detection accuracy of our system. We report area under the ROC curve (AUC) of three data mining algorithms: decision tree (J48), Naïve Bayes (NB) and inductive rule learner (RIPPER).

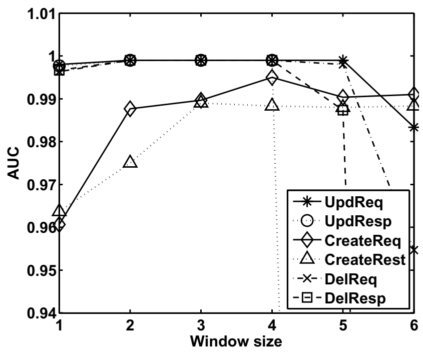
Our experiments are based on two sets of analysis. In the first set we determine the optimum value of n for best average detection accuracy. We perform ROC analysis for window sizes of 1 to 6. For generalization we averaged the AUCs of the three classifiers and using their AUC averages we calculated detection accuracy for all categories of packets. In Figure 6, the overall average detection accuracy for different levels of fuzzing is shown. The figure shows that in most cases window size of 4 gives the best performance in terms of AUC. Increasing the size of n increases the number of features and hence the dimensionality of the data set, thereby exhibiting the `curse of dimensionality`. Whereas, features extracted at smaller values of n, due to simplicity, do not have sufficient discriminative abilities.



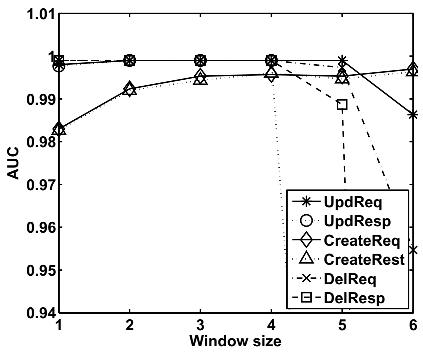
(a) `Fuzzing Rate 2%`



(b) `Fuzzing Rate 5%`



(c) `Fuzzing Rate 10%`



(d) `Fuzzing Rate 20%`

Fig. 6. Average AUC at $2, 5, 10$ and 20% fuzzing rate showing peaks at $n = 4$

In the second set of experiments, to select the most *discriminative* features, we have used standard feature selection method. We employ information-theoretic measure for feature ranking. Information gain is one such measure to calculate the discriminative ability of a feature.

$$IG(Y;X) = H(Y) - H(Y|X)$$

Where ($IG \in [0,1]$) and $H(X)$ and $H(Y)$ are the entropies of a given attribute X and a class attribute Y. We perform feature quantification to support the notion of introducing feature selection. Figure 7 shows the normal probability plot of the information gain of the extracted features. It can be observed that for smaller values of n the IG values of almost all of the features are very low. However for larger values of n some features exhibit significantly large IG values. But as we increase the value of n the curse of dimensionality increases. Therefore our analysis show that $n = 4$ is most suitable in terms of detection.

Table 3. Performance evaluation results for different packet types

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	.392	.570	1.00	0.00	1.00	0.00	1.00	0.00
Jrip	.474	.514	1.00	0.00	1.00	0.00	1.00	0.00
J48	.470	.516	1.00	0.00	1.00	0.00	1.00	0.00

(a) Update PDP context request

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	1.00	0.00	.999	.001	.999	.001	.999	.001
Jrip	.999	.001	.999	.001	1.00	0.00	.999	.001
J48	1.00	0.00	.999	.001	.999	.001	.999	.001

(b) Update PDP context response

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	.490	.506	.490	.510	.999	.001	1.00	0.00
Jrip	.498	.501	.498	.502	1.00	.001	1.00	0.00
J48	.500	.500	.500	.500	.999	.001	.999	.001

(c) Delete PDP context request

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	.481	.506	.490	.510	1.00	0.00	.999	.001
Jrip	.498	.502	.498	.502	.999	.001	1.00	0.00
J48	.498	.502	.498	.502	.999	.001	.999	.001

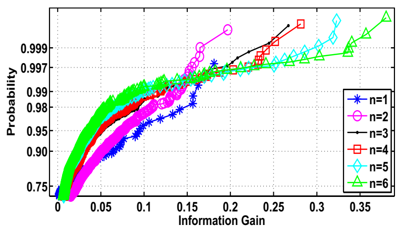
(d) Delete PDP context response

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	1.00	0.00	1.00	0.00	.999	.001	1.00	0.00
Jrip	.999	.001	.999	.001	1.00	0.00	1.00	0.00
J48	.999	.001	.999	.001	.999	.001	1.00	0.00

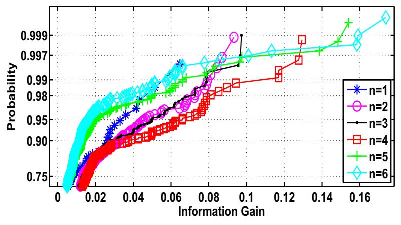
(e) Create PDP context request

FR→	2%		5%		10%		20%	
Classifier↓	DA	FA	DA	FA	DA	FA	DA	FA
NB	.764	.226	.957	.130	.998	.170	.999	.001
Jrip	.760	.229	.956	.051	.996	.110	.995	.007
J48	.640	.362	.906	.113	.990	.015	.995	.004

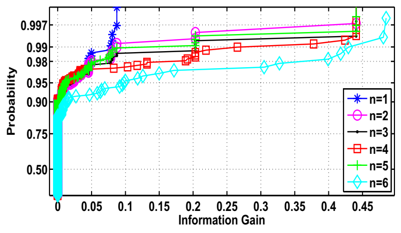
(f) Create PDP context response



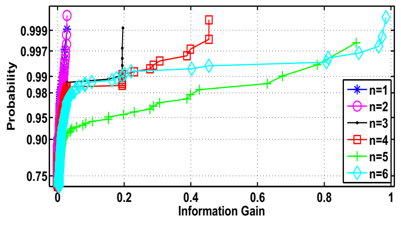
(a) Create PDP Request



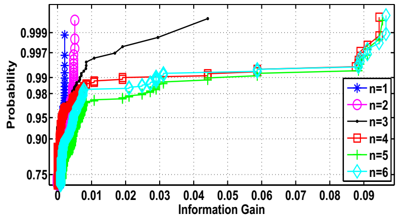
(b) Create PDP Response



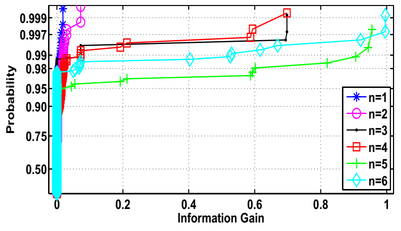
(c) Update PDP Request



(d) Update PDP Response



(e) Delete PDP Request



(f) Delete PDP Response

Fig. 7. Normal probability plot of different types of GTP-C packets

After determining the suitable value of n, i.e., 4 we improve the results by selecting features of high IG values, which results in reduced number of features. The analysis include all types of control packets for the value of window size 4. Table 3 gives detection accuracies (DA) and false alarm rate (FA) for different levels of fuzzing rate (FR). In this figure, we can see that 2% fuzzing is most difficult to detect for some type of packets. The difficulties arrive when the packet size is small. For example in Delete PDP Request/Response packets the average sizes are 60 and 56 bytes respectively. So even for 5% fuzzing the number of bits fuzzed will be 3, which makes it difficult to detect. Packets with fuzzing rate as low as 2% have a very low threat level and can be considered as minor bit errors. However, when the packet size increases as in the case of Create PDP Request/Response the number of bits fuzzed are relatively larger and have a higher threat level. It can be seen from the results that the detection accuracy for packets with higher threat level is as high as 99.9% whereas, the false alarm rate is as low as 0.1%.

7 Conclusion and Future Work

In this paper, we have presented an efficient data mining framework for detection of fuzzing attacks directed towards $3G$ core networks using the control packets of the GTP protocol. The results show that the Markov chain model for feature selection combined with standard classification algorithms is a good technique for detection of fuzzing attacks. The analysis done for $n = 4$ shows that it is most suitable for efficient detection of fuzzing attacks with fuzzing rate of 5% or more whereas, performance results are also satisfactory in most of the cases where fuzzing rate is less than 5%. Currently, we are working on exploring some other data mining techniques to identify features resulting in improved detection accuracy for lower fuzzing rates (1% and 2%). The future work also includes a thorough analysis of the processing overheads of the proposed framework to make it deployable in a real environment.

References

1. Odlyzko, A.: Internet traffic growth: Sources and implications. In: Proc. SPIE, Citeseer, vol. 5247, pp. 1–15 (2003)
2. Peng, X., Yingyou, W., Dazhe, Z., Hong, Z.: GTP Security in 3G Core Network. In: 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, pp. 15–19. IEEE (2010)
3. Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., McDaniel, P., La Porta, T.: On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 223–234. ACM (2009)
4. Whitehouse, O.: GPRS wireless security: not ready for prime time. In: GSM Association Security Group Meeting, Berlin (2002)
5. 3GPP: Security Threats and Requirements. TS 21.133 (V 4.1.00)
6. Dimitriadis, C.: Improving mobile core network security with honeynets. IEEE Security & Privacy, 40–47 (2007)
7. Xenakis, C., Merakos, L.: Vulnerabilities and Possible Attacks Against the GPRS Backbone Network. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 262–272. Springer, Heidelberg (2006)
8. Whitehouse, O., Murphy, G.: Attacks and counter measures in 2.5 G and 3G cellular IP networks. Atstake Inc. (March 2004)
9. Mulliner, C., Vigna, G.: Vulnerability analysis of MMS user agents. In: 22nd Annual Computer Security Applications Conference, ACSAC 2006, pp. 77–88 (2006)
10. Racic, R., Ma, D., Chen, H.: Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. IEEE SecureComm (2006)

11. Enck, W., Traynor, P., McDaniel, P., La Porta, T.: Exploiting open functionality in SMS-capable cellular networks. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, p. 404. ACM (2005)
12. Kotapati, K., Liu, P., Sun, Y., LaPorta, T.F.: A Taxonomy of Cyber Attacks on 3G Networks. In: Kantor, P., Muresan, G., Roberts, F., Zeng, D.D., Wang, F.-Y., Chen, H., Merkle, R.C. (eds.) ISI 2005. LNCS, vol. 3495, pp. 631–633. Springer, Heidelberg (2005)
13. Traynor, P., McDaniel, P., La Porta, T.F., et al.: On attack causality in internet-connected cellular networks. In: USENIX Security Symposium, SECURITY (2007)
14. http://www.openggsn.org/
15. Sanders, G.: GPRS networks. John Wiley & Sons Inc. (2003)
16. Madsen, T., Schwefel, P., Hansen, M., Bogh, J., Prasad, R.: On Traffic Modelling in GPRS Networks, pp. 1785–1789 (2005)
17. Quinlan, J.: C4. 5: programs for machine learning. Morgan Kaufmann (1993)
18. Maron, M., Kuhns, J.: On relevance, probabilistic indexing and information retrieval. Journal of the ACM (JACM) 7(3), 216–244 (1960)

An Efficient Decentralized Rekeying Scheme to Secure Hierarchical Geographic Multicast Routing in Wireless Sensor Networks

Prithu Banerjee, Mahasweta Mitra, Ferdous A. Barbhuiya, Sandip Chakraborty, and Sukumar Nandi

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati, Assam, India
{prithu,mahasweta,ferdous,c.sandip,sukumar}@iitg.ernet.in
http://www.iitg.ernet.in

Abstract. Wireless Sensor Networks consist of several autonomous sensors deployed in a distributed fashion for different purposes like that of wildlife tracing, detection of intruders, environment monitoring etc. Efficient multicast is necessary to scale down the exchange of messages for group communication in sensor networks. However incorporating security in wireless sensor group communication is a huge challenge because of the limited resource availability. In this paper, a decentralized rekeying mechanism based on Logical Key Hierarchy is proposed to secure the efficient Hierarchical Geographic Multicast Routing without affecting its actual performance. The proposed protocol, Secure Hierarchical Geographic Multicast Routing (SHGMR) meets all the security requirements necessary for any kind of secure group communication. This protocol is also efficient in terms of scalability as it uses only $O(log_2 n_{max})$ message transmissions, where n_{max} is the size of the largest subgroup among all the HGMR cells.

1 Introduction

Wireless Sensor Networks (WSN) [1] consist of large number of widely distributed self configuring sensor nodes. Sensor nodes are small, low-powered, low-cost multifunctional devices with data processing and communication capabilities. WSNs have huge applications in traffic management, enemy vigilance, environment monitoring and health care [2]. In most of these domains, the sensor nodes participating in the communication form a group amongst themselves. These groups are dynamic as the nodes can join or leave causing a rapid change in the structure of the group.

The sensor nodes belonging to a group exchange important messages. Thus efficient routing mechanisms are required for proper group communication. Several routing mechanisms exist in literature for group communication in wireless sensor networks. *Hierarchical Geographic Multicast Routing(HGMR)* [3] is an efficient group communication mechanism based on geographical location of the nodes. HGMR integrates the features of *Geographic Multicast Routing(GMR)*

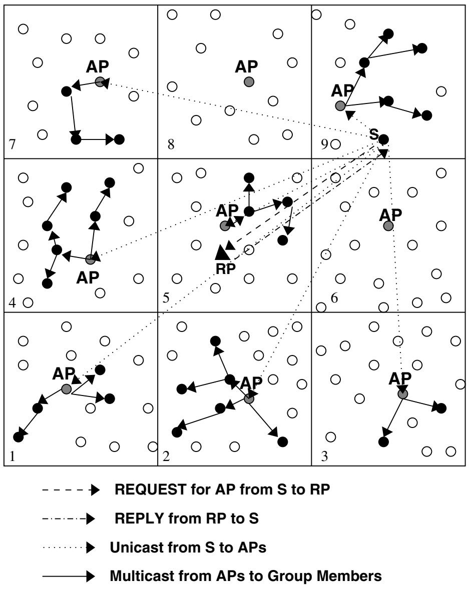

AP
S
RP
1
2
3
4
5
6
7
8
9
REQUEST for AP from S to RP
REPLY from RP to S
Unicast from S to APs
Multicast from APs to Group Members

Fig. 1. Hierarchical Geographic Multicast Routing

[4] and *Hierarchical Rendezvous Point Multicast (HRPM)* [5] to optimize group communication in WSNs. In *Geographic Multicast Routing (GMR)*, the forwarding node selects a subset of its neighbours as *relay* nodes such that choosing them reduces remaining distance towards the destinations. GMR reduces the total number of transmissions by using the basic broadcast advantage of WSNs. However IDs of all the destinations and the chosen relay nodes have to be included in the packet header which leads to a large encoding overhead. *Hierarchical Rendezvous Point Multicast (HRPM)* on the other hand reduces this overhead by hierarchically dividing the arena into smaller cells. Each cell has an *Access Point(AP)* and the entire arena has a *Rendezvous Point* (RP) which can be considered as the root of this hierarchy. A new node joining the arena is assigned the cell of the AP closest to this node. But for communication from RP to AP and AP to nodes, HRPM uses unicast which increases the number of message transmissions. HGMR intelligently combines the low encoding overhead of HRPM with the forwarding efficiency of GMR. Whenever the source wants to send data, it first sends the list of destinations to the RP. The RP returns the set of APs' under which the group members fall. The message is unicast from source to the APs and then each AP multicasts it to the members of its cell using GMR as shown in Figure 1. Thus HGMR provides a trade-off between locality and load balancing. So it is a very efficient routing protocol for group communication in WSNs. Securing such group communication in WSN is an important requirement for e.g. in military applications. In Wireless Sensor Networks any group communication have the following security requirements:

1. *Group Confidentiality:*
   Only the group members must be able to decipher the messages exchanged. This is one of the main challenges of multicast communication. All the nodes that belong to one group can share a single secret key among themselves called the *group key* to secure communication among themselves.This key provides group secrecy as well as source authentication.
2. *Forward and Backward Secrecy:*
   When an already existing member leaves the group, *forward secrecy* has to be ensured so that the node can not decipher any new messages with the group key already known to it. Same is the case when a new node wishes to join the secure group. It becomes important to ensure *backward secrecy* so that the new member can not decipher any previous messages encrypted with the old group key.
3. *Collusion Resistance:*
   The evicted members should not be able to exchange the information known by them to discover the group key i.e. *collusion* has to be prevented.

For all above reasons *rekeying* is an important requirement for dynamic groups. One way of rekeying can be to have a shared secret key between every source-destination pair. The new group key can be sent by symmetric encryption. However for n number of nodes, $O(n)$ transmissions are required and hence, this scheme is not scalable and energy efficient. Furthermore, public key based approaches can not be used as it increases the message complexities enormously which becomes difficult for the resource constrained sensor nodes to process [7].

Securing group communication in WSNs is difficult for many reasons as mentioned in [6]. In order to make the use of sensor networks affordable, the sensor devices are constrained in energy availability and communication capabilities. They are susceptible to physical attacks as they are mostly used in areas which can be accessed by intruders. Several works exist in literature that aim to secure group communication in sensor networks. In SPINS [7], data confidentiality is provided using SNEP protocol. In the same paper, μ-TESLA has been proposed to authenticate broadcast mechanism in resource constraint devices. However, the proposed mechanism is not scalable because of high communication complexity to bootstrap a receiver being in the order of the size of the network, which is an essential part of the scheme. Eschenauer *et al* [8] propose a random key pre-distribution mechanism to secure multicast communication. In [9], the authors propose two centralized group rekeying scheme to secure group communication in wireless sensor networks. However, these schemes do not address the underlying multicast routing mechanism for group communication. WSNs are prone to attacks like node capturing, physical tampering and denial of service [6]. Thus not taking care of security issues in underlying routing protocols often causes it to become prone to attacks as mentioned earlier, especially energy deprivation attacks. The paper on directed diffusion based secure multicast scheme for WSNs [10] presents a protocol called the Logical Key Hierarchy For Wireless Sensor Networks(LKHW) which uses Directed Diffusion mechanism for routing and an extension of LKH for securing the routing mechanism. LKHW provides both backward and forward secrecy and the energy cost also scales logarithmically with the group size. However, to the best of our knowledge, there are no works till now that aims to secure HGMR protocol for group communication in wireless sensor networks.

Logical Key Hierarchy(LKH) is one of the most popular group key management protocols that requires only $O(log_2 n)$ transmissions for rekeying [11], [12]. A tree of keys is created for the group members with the group key occupying the root of this hierarchy. The internal nodes of the tree hold the encryption keys. Here it is assumed that the group controller (GC) assigns keys to group members. Each group member maintains a copy of the individual key it shares with the group controller and the set of encryption keys encountered on the path from the root to the leaf node. However, the basic LKH, being a centralized key management protocol, creates a bottleneck at the group controller. The problem is more intense in the case of WSN as computing power of each sensor node is constrained. Dondeti *et al* [13] propose a dual encryption protocol exhibiting a decentralized architecture that uses hierarchical subgrouping. But this scheme uses signature based public key approach that can not be practically applied in resource constrained sensor nodes.

In this paper, a decentralized rekeying scheme is proposed to secure the efficient HGMR based routing. To reduce the bottleneck at GC, the hierarchical structure of the routing protocol is exploited to decentralize the key management scheme. A modified version of LKH i.e. LKH+, as proposed by Waldvogel *et al* [14], is applied inside each cell instead of applying it on the whole group. The security analysis shows that it does not violate the security criterion stated earlier. The proposed scheme also uses multicast communication for rekeying both at the time of node join and node leave, and hence utilizes the broadcast nature of WSN. The theoretical analysis shows that the proposed scheme uses constant number of message transmissions in normal communication, whereas rekeying requires $O(log_2 n_{max})$ message transmissions, where n_{max} is the size of the largest subgroup among all the cells in HGMR hierarchy. In general, this bound does not affect the efficiency of HGMR and is lesser than that of basic LKH based rekeying.

2 Secure Hierarchical Geographic Multicast Routing (SHGMR)

The proposed scheme aims to secure the communication between the group members of a WSN that use the HGMR protocol. The members are assumed to be distributed among different cells of HGMR hierarchy. The group members in each cell form a subgroup among themselves. Let SG_i denote i^{th} such subgroup. The AP of a particular cell multicast the group communication messages coming from the source to the subgroup members under it. The two main advantages of HGMR protocol that are exploited are as follows:

1. Each cell can be considered as a separate cluster and changes in one cell does not affect sensor nodes in the other cells. This can be exploited to perform rekeying in a decentralized way.

2. At each level of the HGMR hierarchy, an AP holds the information required for routing packets to its child nodes only. This property motivates to go for a rekeying similar to LKH based scheme.

In HGMR, unicast is used to communicate between source and AP whereas multicast is used to communicate between the AP and the group members inside a HGMR cell. In the paper, unicast and multicast are depicted using '$\rightarrow$' and '$\Rightarrow$' symbols respectively. Following assumptions are used throughout this paper while describing the proposed security mechanism.

1. There are total t number of APs in the arena that spans the whole group of n members. An AP can be a group member or a third party assisting the forwarding process. In the latter case, the AP must not be able to decrypt the messages that it forwards.
2. A two level hierarchy of HGMR is assumed for simplicity while describing the scheme. However the protocol is applicable on any number of levels.
3. A group controller (GC) is a randomly picked group member who masters the key refreshing process whenever a node joins or leaves the group.
4. A key distribution protocol through a third party key distribution center (KDC) [15] is used. In the proposed protocol, the RP is assumed to be the KDC as the source nodes for group communication contact the RP to obtain the required list of APs. Along with the list of APs, the RP distributes the secret session keys to the source i.e. one separate key for each AP.

2.1 Description of Key Hierarchy

Figure 2 shows the key hierarchy defined over a group. All the members of a group share a common group key $GKEK$ known only to the members of the group. Along with the $GKEK$, the subgroup members under each AP form a logical key hierarchy among themselves. LKH_m denotes the logical key hierarchy structure for the m^{th} subgroup SG_m. $|SG_m|$ denotes the number of members in SG_m. The leaves of LKH_m consist of $|SG_m|$ number of keys $KSGM_{mj}$, which are the shared secrets between AP_m and each group member $j \in SG_m$. The remaining tree is formed according to the basic LKH+ protocol. The arity of the tree is two. So the LKH_m has $log_2(|SG_m|)$ levels. In Figure 3, an example of key hierarchy is shown for a HGMR cell with Eight members. The details of keys possessed by a group member will be described in the initialization phase. To secure the communication between the source and the APs of different cells in HGMR, a session key KAP_m is generated at the KDC, which is distributed to AP_m and the source. The key distribution mechanism used here is similar to [15].

There are few other keys used in this scheme for the decentralization of basic LKH. This decentralization reduces the bottleneck at GC and makes the approach more optimized for resource constrained nodes of WSN. The notation used to represent those keys and some other notations used in this paper are described in the Table 1.

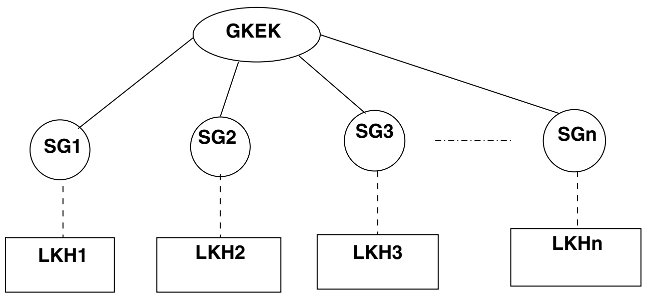


Fig. 2. The Key Structure of the Group

Table 1. Notations used in Rekeying Protocol

Key Notation	Meaning of the Notation
KGM_i	Secret key between i^{th} group member u_i and the Group Controller
$KSGM_{ij}$	Secret key between group member $u_i \in SG_j$ and AP_j
$SGKEK_i$	Subgroup Key Encryption Key of SG_i, known to all members of SG_i. This key is the root of LKH_i
LK	The set of the keys to form LKH
KRP_i	Shared secret between RP and node i
$GKEK'$,$SGEK'$,LK'	denote the new keys after rekeying has been done

2.2 Initialization Phase

A set of keys are distributed to the group members by the GC when a group is initialized. Figure 3 shows the key hierarchy for a subgroup SG_m, where $\{u_i | i \in (1,8)\} \in SG_m$ are the members of a group. For example, node u_8 contains the group key $GKEK$, a secret key KGM_8 shared with GC, secret key $KSGM_{8m}$ shared with AP_m and the set of keys from the key hierarchy-$\{LK_{78}, LK_{5678}, LK_{12345678}\}$. The key at the root of the hierarchy is common to all the members of the subgroup. Here $LK_{12345678}$ is at the root of the key hierarchy, hence $SGKEK_m = LK_{12345678}$.

2.3 Secure Message Transmissions

There can be two kinds of message transmissions:

1. The normal message transmission in an already formed group using group key $GKEK$.
2. The set of messages transmitted for rekeying when a node joins or leaves the group.

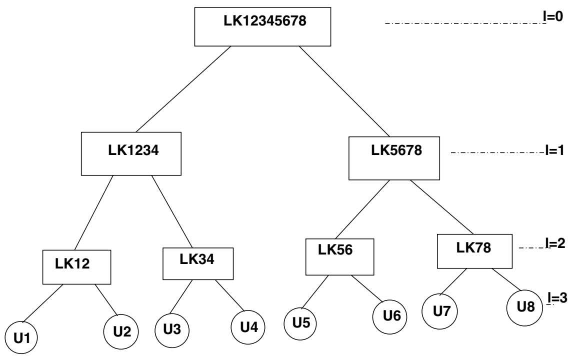


Fig. 3. Logical Key Hierarchy of an SG with Eight Members

Normal Message Transmission: When the source S wants to communicate with other group members, according to HGMR it first identifies the proper APs under which the group members belong to. To get the list of APs, it first queries the RP and the RP replies with the list of APs. In SHGMR, RP also sends the corresponding KAPs, using the key distribution protocol similar to [15]. Additionally all communication between source and RP is encrypted by KRP_S so that none can alter the message content.

In the original HGMR protocol, the source unicasts the message to the APs of the list and from each AP the message is multicast to the corresponding subgroup members. In SHGMR, for every AP of the list, the source first encrypts the message with $GKEK$ and appends the list of destinations to which the AP has to forward the encrypted message. Then the whole packet (i.e. the encrypted message and destination list) is encrypted by KAP and unicast to the APs of the list. Upon receiving the packet an AP decrypts it with key KAP to get the list of destination to which it needs to forward the message. It can be noted that the message is encrypted with group key $GKEK$. Hence if an AP is not a group member, it can not decrypt it.

Assume source S wants to send message M to set of group members $\{D_1$, D_2, ..., $D_n\}$. The list of APs returned by RP is $\{AP_1$, AP_2, ..., $AP_t\}$. Now to each AP_i of the returned list, S unicasts the encrypted message and a subset of group members that belongs to the cell of that AP_i i.e. $\{D_{i1}$, D_{i2}, ..., $D_{id}\}$ (assume that the subset size is some d where $d \leq n$) encrypted with the secret key KAP_i established between S and AP_i. After receiving the message, AP_i decrypts it to obtain the destination nodes and multicast the $E_{GKEK}(M)$ encrypted with $SGEK$ to those nodes. So the message sequence is as follows:

1. $S \rightarrow RP$: $E_{KRP_S}\{D_1, D_2, ..., D_n\}$
2. $RP \rightarrow S$: $E_{KRP_S}\{AP_1, AP_2, ..., AP_t\}$
3. $S \rightarrow AP_i$: $E_{KAP_i}(E_{GKEK}(M)||\{D_{i1}, D_{i2}, ..., D_{id}\}), \forall i \in (1, t)$
4. $AP_i \Rightarrow \{D_1, D_2, ..., D_d\}$: $E_{SGKEK_i}(E_{GKEK}(M))$

Rekeying Mechanism: In order to provide forward and backward secrecy whenever the member set of the group changes the group keys must be changed.

It has been shown in this paper that every change in the group does not always require changing the existing group encryption key $GKEK$. Hence the overhead is reduced. At the same time the rekeying procedure does not solely depend on GC. It delegates the load of rekeying at the subgroup level to the AP of the subgroup. This decentralization avoids the bottleneck at GC.

Whenever a node u_i joins or leaves the subgroup SG_m, all the keys on the path from the root to the new leaf of LKH_m has to be rekeyed. In Figure 3, suppose a change occurs at u_8. The compromised key set is $\{LK_{78}, LK_{5678}, LK_{12345678}\}$. The new key set for the compromised keys must be sent to the corresponding subgroup members. However all of them need not be transmitted by AP_i in LKH+, as proposed by Waldvogel *et al* [14]. Each node u_i only needs to know the first key affected on the path from its key at the leaf to the key at the root of the LKH tree, as at each level of key hierarchy, only a single key has to be refreshed. It can generate the other keys using a one way hash function applied on the above key. This reduces the number of rekeying message transmissions. Assume node u_8 joins the subgroup, as shown in Figure 3, and the new key set for the key tree be $\{LK'_{78}, LK'_{5678}, LK'_{12345678}\}$. The relationship among the keys is as follows:

$$LK'_{12345678} = H(LK'_{5678}) = H(H(LK'_{78})), \text{ where H is the hash function.}$$

The advantage of using LKH+ is that only one key is required to send at any node. From it the node can compute other refreshed key. This reduces number of messages required for rekeying.

(i) Node Join

The new node sends a JOIN message to the RP indicating its own location. RP replies with the location of the AP under which it must join. The node finally sends an UPDATE message to that AP. These messages constitute the join process as per the original HGMR protocol.

All the above messages are transmitted also in SHGMR. However some additional messages are encountered for the rekeying process. Let node u_i joins in the subgroup SG_m. u_i has a secret key KGM_i with GC. The AP_m, under which it joins, assigns a secret key $KSGM_{im}$ to it. This $KSGM_{im}$ is the new leaf in LKH_m. All the keys on the path from the root to the new leaf of LKH_m have to be rekeyed. Thus $SGKEK_m$, which is the root of the hierarchy, is automatically refreshed. Two cases can arise in this situation.

1. If AP_m joins as a member of SG_m, the group key GKEK has to be rekeyed along with all the keys on the path from the root to the new leaf of LKH_m. The new $GKEK'$ has to be informed to all the group members.
2. If any other node joins as a member of SG_m, then there is no need to change GKEK. Here rekeying is necessary only for LKH_m. Thus it passes on the burden of rekeying from group level down to subgroup level.

Let node u_8 joins SG_m, as shown in Figure 3. Then the keys of LKH_m are changed first. The sequence of rekeying messages transmitted are as follows.

1. $AP_m \rightarrow u_8$: $E_{KSGM_{m8}}(LK'_{78})$, u_8 decrypts it and computes LK'_{5678} and $LK'_{12345678}$ using the hash function.
2. $AP_m \rightarrow u_7$: $E_{KSGM_{m7}}(LK'_{78})$, u_7 decrypts it and computes LK'_{5678} and $LK'_{12345678}$ similarly.
3. $AP_m \Rightarrow \{u_5, u_6\}$: $E_{LK_{56}}(LK'_{5678})$, u_5 and u_6 decrypts it and computes $LK'_{12345678}$ in similar way.
4. $AP_m \Rightarrow \{u_1, u_2, u_3, u_4\}$: $E_{LK_{1234}}(LK'_{12345678})$ The keys for the LKH are refreshed and new $SGKEK'_m = K'_{12345678}$.
5. **Case I.** If $u_8 \neq AP_m$ then GC forwards the unaltered group key $GKEK$ using following two messages. It must be noted that, in this case GC contacts with the KDC to get the KAP_m.
   (a) $GC \rightarrow AP_m$: $E_{KAP_m}(E_{KGM_8}(GKEK)||u_8)$
   (b) $AP_m \rightarrow u_8$: $E_{SGKEK'_m}(E_{KGM_8}(GKEK))$

   Case II. If $u_8 = AP_m$ then the GKEK should be changed to a new group key $GKEK'$. It is distributed to the other group members using following messages.
   (a) $GC \rightarrow AP_m$: $E_{KAP_m}(GKEK'||\{u_1, u_2, ..., u_8\})$
   (b) $AP_m \Rightarrow \{u_1, u_2, ..., u_d\}$: $E_{SGKEK'_m}(GKEK')$
   (c) $GC \rightarrow AP_i$: $E_{KAP_i}(E_{GKEK}(GKEK')||\{D_{i1}, D_{i2}, ..., D_{id}\})$, $\forall i \in (1, t) \wedge i \neq m$
   (d) $AP_i \Rightarrow \{D_{i1}, D_{i2}, ..., D_{id}\}$: $SGKEK_i(GKEK')$, $\forall i \in (1, t) \wedge i \neq m$

(ii) Node Leave

Whenever a node wants to leave the group, as per HGMR, it sends an UPDATE message to its AP. If it is the last node under the AP, AP sends a NOTIFY message to the RP. In SHGMR, two cases can arise. The member leaving the group can be a normal group member or an AP. When an AP leaves a group, it continues to serve like an AP as before but as it is not a group member, it should not be able to decrypt any group messages further. The set of keys of the subgroup where the membership changes is required to be refreshed is shown. Assume node u_8 leaves the group from SG_m. Then the keys of LKH_m are changed using the following messages.

1. $AP_m \rightarrow u_7$: $E_{KSGM_{m7}}(LK'_{78})$, u_7 decrypts it and computes LK'_{5678} and $LK'_{12345678}$ similarly.
2. $AP_m \Rightarrow \{u_5, u_6\}$: $E_{LK_{56}}(LK'_{5678})$, u_5 and u_6 decrypts it and computes $LK'_{12345678}$ in similar way.
3. $AP_m \Rightarrow \{u_1, u_2, u_3, u_4\}$: $E_{LK_{1234}}(LK'_{12345678})$ The keys for the LKH are refreshed and new $SGKEK'_g = K'_{12345678}$.
   If $u_8 \neq AP_m$ then no further messages are required. Else a new group key $GKEK'$ is chosen and it is unicast to all the remaining members of SG_m by GC itself but not via AP_m. For all other subgroups, $GKEK'$ is multicast using HGMR protocol via their APs, similar to the case of node join.

3 Security Analysis

In this section, the proposed SHGMR is critically analyzed to show that the four requirements of secure group communication- Group Confidentiality, Forward Secrecy, Backward Secrecy and Collusion Resistance are met.

1. Group Confidentiality: The group confidentiality is satisfied by assuring that only the group members can possess group key $GKEK$ using which the group messages can be decrypted. Any node, even an AP which is not part of the group can not decrypt the message being exchanged among the members. LKH+ is applied inside a subgroup. It assures that all communication made in a subgroup encrypted by subgroup key remains confidential. Although it is applied only inside the subgroups, the presence of $GKEK$ guarantees that no other nodes apart from the group members can decrypt the group messages.

To send message M to the group members, source first contacts RP. All communication between source S and RP is encrypted by KRP_S known only to S and RP. So no one else can interpret or modify the message content i.e the destination list or the list of APs. Then S encrypts the message M with group key $GKEK$. The encrypted message is appended with a destination list. The appended message is further encrypted by KAP and sent from source to the APs. Thus only destination APs can decrypt it to get their destination list, but they can not decrypt M if they do not know $GKEK$. Again the messages sent from AP to the subgroup members under it, are always encrypted by the subgroup key. So any node, inside the cell, that is not a group member can not decrypt it.

2. Forward Secrecy: Whenever a node leaves a group, two cases can occur. If the node is not an AP, $GKEK$ need not to be rekeyed. However forward secrecy is maintained as $SGKEK_m$ is refreshed. Any communication from AP_m to the subgroup members is encrypted by $SGKEK'_m$. Thus the member who left can not decrypt messages as $SGKEK'_m$ is not available to it. In case the leaving member is the AP of the subgroup i.e. AP_m, it has access to $GKEK$, $SGKEK_m$ and $SGKEK'_m$. Thus in this case $GKEK$ has to be refreshed. $GKEK'$ is unicast to each of the subgroup members in the affected cell using the shared secret key KGM. This unicast is done by GC directly to the members of SG_m so that AP_m can not get the $GKEK'$. The new group key (i.e. $GKEK'$) is used to encrypt all future messages. It should be noted that the newly chosen $GKEK'$ is completely independent of previous $GKEK$, so that knowing previous $GKEK$, a departing node can not guess new $GKEK'$. Thus forward secrecy is maintained.

3. Backward Secrecy: Whenever a node joins a group, two cases can occur. If the node is not an AP, backward secrecy is maintained as $SGKEK_m$ is refreshed. The new node is not aware of the old $SGKEK_m$ and can not decrypt previous messages. In case the new member is an AP, GKEK is rekeyed as the AP of the subgroup has access to $GKEK$, $SGKEK_m$ and $SGKEK'_m$. The new group key (i.e. $GKEK'$) is used to encrypt future traffic after the AP joins. Thus it can not decrypt any previous messages encrypted with old $GKEK$.

4. Collusion Resistance: Whenever an AP joins or leaves a group, the $GKEK$ is changed and $GKEK'$ is informed to the group members using the above described scheme. Similarly, if the node is not an AP, $SGKEK$ is changed. Thus even if the evicted members discuss among themselves and try to obtain information about the old group key or the old subgroup key it shall be of no use as those keys are already changed. The encryption keys that were compromised are also changed. Thus the proposed scheme is collusion resistant.

4 Performance Analysis

The complexity of the proposed algorithm is analyzed in terms of number of message transmissions required. At each stage, a comparison between SHGMR and the existing HGMR protocol is made. This vividly shows that SHGMR has been successful in securing group communication without affecting its original performance.

In traditional LKH approach, rekeying of group key for a group with n members requires $O(log_2 n)$ message transmissions. However the decentralized architecture of the proposed scheme has been successful in reducing this bound.
The analysis takes the same assumption that the group spans across t APs and has n members. For any WSN, $t \ll n$. As per HGMR protocol, for a particular arena the RP and APs are fixed.

Normal Message Transmission: The number of messages transmitted are as follows:

SHGMR

1. One unicast message from source to RP providing the list of destinations it wants to communicate with.
2. One unicast message from RP to source returning the list.
3. According to the key distribution protocol [15], three unicast messages are required to be transmitted between the source and each AP_i of the list before actual messages can be transmitted.
4. t unicasts of the actual message between the source and each AP in the list.
5. One multicasts initiated from each of the t APs to their corresponding subgroup members.

So total number of messages transmitted is $5t + 2$ that is of $O(1)$ as total number of APs in the arena is constant and $t \ll n$.

HGMR

In this case, (1), (2), (4) and (5), as described previously in case of SHGMR hold. So the total number of messages transmitted is $2t + 2$ in case of the original HGMR protocol. Thus only at the cost of a few additional messages, SHGMR provides secure group communication in a WSN working on the HGMR protocol.

Rekeying Messages: These messages are required when a new member joins the group or any existing member leaves. AP_m is the AP of the cell affected by a node join or leave and SG_m is the subgroup under AP_m.

(i) Node Join:

When a new node joins the group, the number of transmitted messages are:

SHGMR

1. One unicast JOIN message from the joining node to the RP.
2. One unicast message from the RP to the node informing the location of the AP_m under which it should join.
3. One unicast UPDATE message from the joining node to AP_m.
4. $log_2|SG_m|$ number of multicasts from AP_m to the members of SG_m.
5. **Case I.** When the new node is not AP_m,
   (a) One unicast message from GC to RP requesting for KAP_m
   (b) One unicast message from RP to GC in reply to the previous request
   (c) Three unicast messages between GC and AP_m for authenticating each other before starting communication.
   (d) One unicast message between GC and AP_m transmitting the encrypted $GKEK$.
   (e) One unicast message between AP_m and the new member informing $GKEK$.

   Whenever GC wants to communicate with AP_m, five unicast messages i.e. (a), (b) and (c) are required for key distribution as per [15].

   Case II. When the new node is AP_m,
   (a) Five unicast messages for key distribution between GC and AP_m as explained earlier.
   (b) One unicast message between GC and AP_m to inform $GKEK'$.
   (c) One multicast message from AP_m to the subgroup members of SG_m to inform $GKEK'$.
   (d) $t-1$ number of unicast messages from GC to the other unaffected APs containing subgroup members for informing $GKEK'$. Each such unicast will require five extra unicast messages between GC and each unaffected AP for key distribution as explained earlier i.e. a total of $5(t-1)$ unicast are performed for key distribution.
   (e) One multicast from each of the $t-1$ APs to send $GKEK'$ to the subgroup members.

The total number of messages transmitted is $log_2|SG_m| + 10$ when the node is not AP_m , and $7t + log_2|SG_m| + 3$, when the node is AP_m. As t is constant for a particular arena, the total number of messages transmitted during a node join is $O(log_2 n_{max})$ where n_{max} is the size of the largest subgroup among all the cells.

HGMR

In this case, (1), (2) and (3), as described for SHGMR (node join) hold. Thus three messages are transmitted in the original HGMR protocol whenever a new node joins the group.

(ii) Node Leave:

When a node leaves a group, the number of transmitted messages are:

SHGMR

1. One unicast UPDATE message from the node to the AP_m.
2. One unicast NOTIFY message from AP_m to the RP in case the node is the last one in the subgroup SG_m.
3. $log_2|SG_m|$ number of multicasts from the AP to the members of the subgroup.
4. **Case I.** When the node is not an AP, no other message transmission is required. Thus the total number of message transmissions in this case is $log_2|SG_m| + 2$ i.e. $O(log_2 n_{max})$.
   Case II. When the node is an AP,
   (a) $|SG_m|$ unicasts of $GKEK'$ to each subgroup member under AP_m from GC.
   (b) $t - 1$ number of unicast messages from GC to the other unaffected APs containing subgroup members for informing $GKEK'$. Each such unicast will require five extra unicast messages between GC and each unaffected AP for key distribution as explained earlier i.e. a total of $5(t-1)$ unicast are performed for key distribution.
   (c) $t - 1$ number of multicasts from the above APs to send $GKEK'$ to the subgroup members.
   Thus $|SG_m| + 7t + log_2|SG_m| - 5$ message transmissions are required in this case i.e. $O(n_{max})$.

HGMR

Here, (1) and (2), as described in case of SHGMR (node leave) hold. Thus two messages are transmitted in the original HGMR protocol whenever a new node leaves the group.

Though the number of message transmissions encountered during a node join or leave are slightly higher than the constant bounds of HGMR, each transmission is made following the protocols of the scheme and thus its functionality is not hampered. Securing group communication in WSNs is a primary requirement and SHGMR has been successful in providing this as shown in the security analysis of the scheme. Thus these extra messages are worthwhile as this makes the original HGMR deployable in real life scenarios.

Thus for each of the above cases, an $O(log_2 n_{max})$ bound is achieved by the SHGMR protocol except when an AP leaves a group where an $O(n_{max})$ bound is obtained. However an AP leaving or joining a group is not a common event. So this bound does not affect reduction in the message count than the $O(log_2 n)$ bound of the traditional LKH approach. This is achieved as a result of decentralizing the rekeying process. Also, SHGMR protocol, as seen earlier, does not require rekeying the GKEK every time a node leaves or joins which significantly reduces the overhead at the GC. Thus it is scalable and more energy efficient for WSN.

5 Conclusion

In this paper, a scheme is proposed to secure HGMR based group communication in sensor network. The SHGMR protocol uses location based efficient routing protocol HGMR and a decentralized version of LKH+ key management protocol. It achieves low encoding overhead than signature based schemes. Forwarding efficiency is enhanced with reduced number of message transmissions incurred during rekeying. Compared to traditional rekeying mechanisms that run using linear number of message transmissions, the proposed model uses much less number of transmissions. Rekeying at subgroup takes place locally, independent of others, thus reducing the overhead at GC. Though it does not deal with the issue of data aggregation, it can be extended to incorporate this.

References

1. Akyildiz, I.F., Su, E.W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: Survey. Journal of Computer Networks 38(4), 393–422 (2002)
2. Chong, C.Y., Kumar, S.P.: Sensor Networks: Evolution, Opportunities, and Challenges. Proceedings of the IEEE 91(8), 1247–1256 (2003)
3. Koutsonikolas, D., Das, S.M., Hu, Y.C., Stojmenovic, I.: Hierarchical Geographic Multicast Routing for Wireless Sensor Networks. Journal of Wireless Networks 16(2), 449–466 (2010)
4. Sanchez, J.A., Ruiz, P.M., Stojmnenovic, I.: GMR: Geographic Multicast Routing for Wireless Sensor Networks. In: The 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications, pp. 20–29 (October 2006)
5. Das, S.M., Pucha, H., Hu, Y.C.: Distributed Hashing for Scalable Multicast in Wireless Ad Hoc Networks. IEEE Transaction on Parallel and Distributed System 19(3), 347–362 (2008)
6. Perrig, A., Stankovic, J., Wagner, D.: Security in Wireless Sensor Networks. Communications of the ACM 47(6), 53–57 (2004)
7. Perrig, A., Szewczyk, R., Tygar, J., Wen, V., Culler, D.: SPINS: Security Protocols for Sensor Networks. Journal of Wireless Networks 8(5), 521–534 (2002)
8. Eschenauer, L., Gligor, V.D.: A Key-Management Scheme for Distributed Sensor Networks. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (2002)
9. Wang, Y., Ramamurthy, B.: Group Rekeying Schemes for Secure Group Communication in Wireless Sensor Networks. In: The Proceeding of IEEE ICC, pp. 3419–3424 (2007)
10. Pietro, R.D., Mancini, L.V., Law, Y.W., Etalle, S.: LKHW: A Directed Diffusion-Based Secure Multicast Scheme for Wireless Sensor Networks. In: Proceedings of International Conference on Parallel Processing Workshops, pp. 397–406 (October 2003)
11. Wong, C.K., Gouda, M., Lam, S.S.: Secure Group Communications Using Key Graphs. Department of Computer Sciences, The Univ. of Texas at Austin, Tech. Rep. TR-97-23 (July 1997)
12. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure Group Communication using Key Graphs. IEEE/ACM Transaction on Networking 8(1), 16–30 (2000)

13. Dondeti, L.R., Mukherjee, S., Samal, A.: A Dual Encryption Protocol for Scalable Secure Multicasting. In: Proceedings of IEEE International Symposium on Computers and Communications (1999)
14. Waldvogel, M., Caronni, G., Sun, D., Weiler, N., Plattner, B.: The VersaKey Framework: Versatile Group Key Management. IEEE Journal on Selected Areas in Communications 17(9), 1614–1631 (1999)
15. Popek, G., Kline, C.: Encryption and Secure Computer Networks. ACM Computing Surveys, 331–356 (December 1979)

SecWEM: A Security Solution for Web Based E-mail

Ravi Shankar Yadav, Praveen Likhar, and M. Keshava Rao

Centre for Artificial Intelligence and Robotics (CAIR)
Defence Research and Development Organisation (DRDO) Bangalore-93, India
{ravi.yadav,praveen.likhar,keshava}@cair.drdo.in

Abstract. Web based e-mail (Webmail) service is a popular mode of e-mail communication and is being widely used for personal and business purposes. Security of webmails carrying sensitive commercial or corporate information is an important requirement today. A comprehensive solution is expected to cover confidentiality and integrity requirements during transit as well as authentication of the origin of webmails. Although some e-mail security solutions such as PGP, S/MIME, SMS and solution from Freenigma are currently available, most of them are tailored for handling e-mail sent or received by mail clients such as the Outlook Express or Eudora and they cannot handle webmails sent or received by browsers. The Freenigma solution handles a few specific webmail services but does not provide a generic solution. The main challenge in developing a security solution for webmails lies in building a parser to extract e-mail header details and mail body from a HTTP message, that can work with all webmail services. To address this challenge, we propose SecWEM, a desktop level end-to-end security solution. The problems involved in development and how they have been solved are presented in this paper.

Keywords: Information security, e-mail encryption, Webmail, and Webmail security.

1 Introduction

One of the important services that the Internet supports today is electronic-mail or e-mail. The users of this service include government officials, professionals, executives in private and public sector, teachers, students and more. Next only to the medium of telephone, the e-mail has become an indispensable part of everyday operations of its users. The sensitivity of the messages being carried by e-mail today has a much wider range than it was a decade ago, which raises concerns about protecting e-mail messages from unauthorized access and inspection[1][2]. Thus, along with the need to communicate sensitive messages comes the need for secure delivery of such messages. Firstly, the messages must be protected from eavesdropping, ensuring confidentiality of the message. Secondly, tampering of messages during transit must be prevented, thus ensuring integrity of the messages. Finally, the origin of the mail must be authenticated so that spoofing of sender's identity is prevented. Collectively these three requirements constitute security of e-mail.

Web-based e-mail, called webmail, is a widely used form of e-mail communication. In the context of providing security to webmail, it is necessary to appreciate the difference between standard e-mails and webmails. The former type of e-mail services use the Simple Mail Transfer Protocol (SMTP) [3] for sending the mails and the Post Office Protocol (POP) [4] or the Internet Mail Access Protocol (IMAP) [5] for receiving or reading the mails. The SMTP is used for transfer the mails from one server to another over the Internet. On the other hand, the webmail services use the Hyper Text Transfer Protocol (HTTP) [6] for both sending and receiving webmails. As a consequence of the differences in the protocols used, the user interfaces are different as well. One uses what may be called as mail clients also known as mail user agents (MUAs) such as the Outlook Express or Eudora to send or receive standard e-mail. A browser such as the Internet Explorer or Firefox is used as an MUA also termed as *webmail client*, for availing webmail services.

To provide security to the e-mails there are some solutions like Entrust SMS [7], Freenigma [8], PGP [9], S/MIME [10], openPGP [11] and TLS [12]. Among these e-mail security solutions only Entrust SMS and Freenigma can provide security to webmails, but both these solutions have their own limitations.

Given the present options for solutions for e-mail security, it is desirable to have a solution for webmail security that can handle different types of webmail services at the desktop to desktop level. Additionally, the solution should be a generic one independent of specific browsers. We have developed a solution keeping these factors in mind. Making this solution compatible with all types of webmail services and dealing with updates of the e-mail formats of webmail services were two major problems. The latter problem required the development of a module to train the solution to adapt to the changing HTTP message formats of webmail services.

The paper is organized in seven sections. Following this introductory section we give a detailed account of the security services required for protecting sensitive webmails. In the third section we discuss the existing solutions. The fourth section explains a generic approach for security of webmails, the challenges involved and a security solution based on this approach. The fifth section covers implementation and testing. In the sixth section we present the performance and overhead statistics. The seventh section concludes the paper.

2 Security of Webmails

All the security threats to the traditional e-mails are equally applicable to the webmails. In this section we list common security threats to e-mails and cryptographic services required to protect against them. The most important security services required are Authentication, Access control, Confidentiality, Integrity, Non-repudiation, and Availability [13].

2.1 Security Threats to E-mails

Usually e-mail is transferred in plain text passing through one or more intermediaries. Also, e-mail servers may take backup of e-mails that pass through them. There is a risk that an attacker can access the e-mail contents, either by eavesdropping on the network or accessing a repository. Some of the contents within these e-mails may be sensitive in nature. Disclosure of such sensitive data can result in loss or damage, such as identity theft or loss of business. Sometimes e-mail contents can be modified during transit or storage [14]. As e-mail passes through various routers and e-mail servers, a person who has access to these routers or mail servers can modify the contents of the e-mail. An attacker that resides on the same Local Area Network (LAN) can use a tool for spoofing Address Resolution Protocol (ARP) such as "ettercap" to intercept or modify all the e-mail packets going to and from a mail server or gateway [14].

It is relatively easy to modify e-mails such that they appear to come from a legitimate source; this attack is known as e-mail spoofing. Usually an attacker modifies the sender's address in the 'From' field of SMTP protocol and other parts such as 'Return-Path', 'Reply-To' fields of the e-mail header to appear as though the e-mail is sent by some known sender. E-mail spoofing is also used for phishing to hide the origin of an e-mail. However, existing e-mail standards do not put any emphasis on security and that's why e-mail is inherently vulnerable to both passive and active attacks [14]. The most common threats to e-mail are listed in Table 1.

Table 1. Common Email Threats

S.No.	*ThreatType*	*Threats*
1	Active	Message Modification, Spoofing
2	Passive	Eavesdropping, Invasion of privacy, Identity theft, Unprotected Backups

2.2 Protection of E-mail against Security Threats

Encryption is the most preferred solution for mitigating the risk of loss of confidentiality. If e-mail is sent in encrypted form, then it cannot be read even by the person who has the access or a person who hacks the en-route e-mail servers or routers, unless the encryption key is compromised or the algorithm used is broken. Message manipulation by an attacker can take the form of data modification within the message or even substitution of credentials, to change the apparent source of the e-mail. This threat can be addressed by data authentication, which consists of two components namely data integrity and data origin authentication. Integrity check mechanisms such as hashing can be used to ensure that messages have not been tampered with in transit. Digital signature enable the recipient to verify that message originate from the expected sender. It makes sure that the person who is claiming to be the sender of the message really is the one from whom it originates.

3 Related Work

A number of solutions for providing confidentiality, integrity, and origin authentication to the e-mails are available today. Some of them are commercial while some are freeware. The foremost and perhaps the oldest is Pretty Good Privacy (PGP) [9]. Though it started as a freeware, now commercial versions are available. The main point to note is that the different versions of the PGP are solutions for security of standard e-mails but they are not designed to handle webmails. There is another popular solution for e-mail security from the Entrust Corporation called the Secure Messaging Solution (SMS) [7]. This solution too is primarily aimed at securing standard e-mails. Realising the importance of providing security to webmails, the SMS suite offers a solution for webmail services. However, this solution entails an extra server system within the customer premises. A number of other less popular solutions are available [10] [11] but none of them handle webmail security. Within the scope of a single webmail service, it is possible to provide security to webmail services using the transport layer security option, SSL/TLS [12]. However, in this approach the e-mail will remain in plain in the e-mail servers, something that the users would not be comfortable about. The Freenigma [8] group has announced that they have developed a solution for Google webmail (Gmail) and yahoomail but this solution does not support other webmail services.

Over the years several new secure e-mail protocols have also been proposed by researchers. Some examples of these e-mail security protocols are SMEmail [14], WSEmail [15] and Trusted e-mail Protocol [16]. The process of adopting these protocols may not be viable most of the times, as these protocols require replacing the existing e-mail protocols. These protocols may be suitable for some closed user group but the members of such group cannot communicate with others because of the interoperability issues.

4 Our Approach to Webmail Security

In this section we briefly describe our approach to webmail security, SecWEM. It is an end-to-end security solution for webmails. To make SecWEM independent of webmail clients, we design it to work as a proxy. The SecWEM performs cryptographic operations on outgoing webmails and forwards them to their respective webmail servers. While receiving at the destination, after performing the reverse cryptographic operation, SecWEM forwards the webmails to the local webmail client.

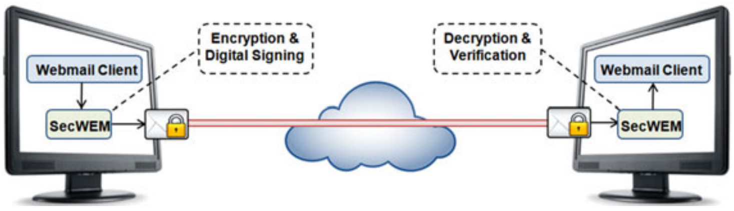


Fig. 1. Webmail communication through SecWEM

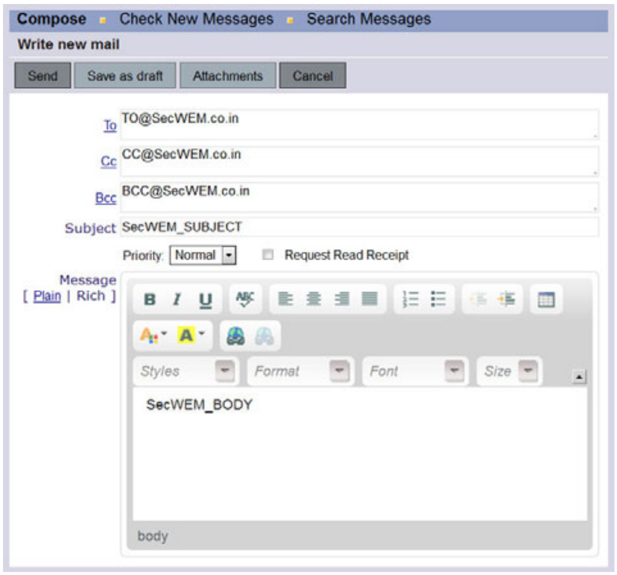
4.1 Service-Independent Handling of Webmails

Extracting e-mail fields of the webmail such as sender's e-mail address (From field), recipients' e-mail address (To, Cc and Bcc fields) and mail body is one of the major tasks in developing any security solution for webmails. If a solution is desired to work with all webmail services, then to extract all the information mentioned above, a universal parsing module is required and building this is a challenge. This subsection explains the challenge involved in building a universal parsing module and our approach to overcome the challenge.

All webmail services use HTTP protocol to transfer webmails between web browser and webmail server. For this purpose webmail services place all the e-mail fields such as From, To, Cc, Bcc, Subject *etc.* and mail body in a particular format into the data portion of the HTTP message. Building a parser to extract these e-mail fields and mail body requires knowledge of the format used by the webmail service. Even though every webmail service uses HTTP protocol to transfer webmail, there is still no standard format for placing the e-mail fields and the mail body into the HTTP data portion. Different webmail services use different formats and sometimes they also change these formats from time to time. This is the reason why existing webmail security solutions do not work with all types of webmail services.

However some existing webmail security solutions work with limited webmail services. The principal approach used by all these solutions is to manually find out the format used by a particular webmail service and hardcode it in the solution. But there is a main drawback with this approach; these solutions are susceptible to fail whenever the webmail service changes or modifies the format.

To address this challenge, we have used an automated reverse engineering approach to design a recogniser module and a universal parser module. The universal parser module extracts the information using the format identified by the recogniser module. To find out the format, the recogniser prompts the user to send a dummy webmail to a pre-specified dummy recipient with a pre-specified dummy mail body, as shown in Fig. 2. After the user sends the dummy webmail, the recogniser module intercepts the webmail and finds out the format by identifying the start and end delimiters of each e-mail fields and the mail body. The recogniser module extracts the delimiters for each field by identifying the prefix and suffix string of specified length for the corresponding dummy value of the field in the HTTP Message. A delimiter template is formed for a particular webmail service, which contains delimiters of all e-mail fields and



To TO@SecWEM.co.in
Cc CC@SecWEM.co.in
Bcc BCC@SecWEM.co.in
Subject SecWEM_SUBJECT
Priority: Normal
Request Read Receipt
SecWEM_BODY

Fig. 2. Dummy Webmail

that of the mail body. This template is used by the universal parsing module for extracting the information. When a webmail service changes or modifies its format, the universal parsing module fails to extract the required fields from the HTTP message and then prompts the user to run the recogniser to find out the new format. In this way SecWEM works for any webmail service. A user just needs to run the recogniser once, when he is using SecWEM for the first time or when the webmail service changes its format.

4.2 SecWEM Architecture and Working

There are six main modules in SecWEM as depicted in Fig. 3. The main functions of each module are described below.

The **HTTP Parser and E-mail Type Detector (HPED) Module**, parses the HTTP message and extracts the HTTP message headers and HTTP message data. From the extracted headers it finds out the HTTP message type and Uniform Resource Locator (URL). By using these parameters it determines whether the type of the HTTP message is webmail transaction or not. While receiving the webmail, this module also determines whether it is encrypted or plain.

The **Universal Outgoing Webmail Parser (UOWP) Module**, parses the HTTP message data to extract e-mail address of sender, e-mail address(es) of receiver(s) and mail body. These are essential parameters required during cryptographic operations.

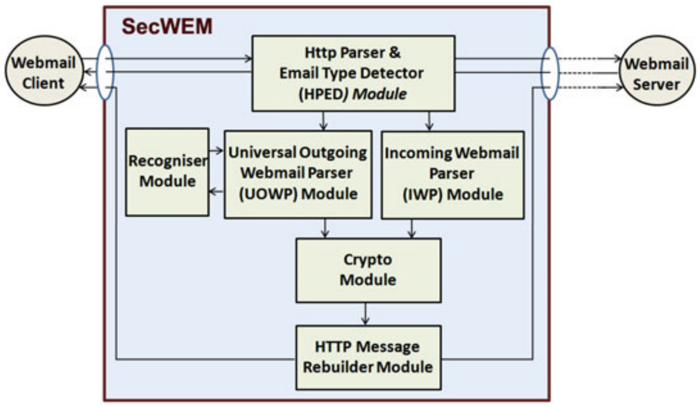

SecWEM
Webmail Client
Http Parser & Email Type Detector (HPED) Module
Webmail Server
Recogniser Module
Universal Outgoing Webmail Parser (UOWP) Module
Incoming Webmail Parser (IWP) Module
Crypto Module
HTTP Message Rebuilder Module

Fig. 3. SecWEM Block Digram

The **Recogniser Module**, is associated with the UOWP module. It identifies the format used by the webmail service for placing the e-mail fields into the HTTP message.

The **Incoming Webmail Parser (IWP) Module**, parses the incoming encrypted webmail to extract the sender's e-mail address, encrypted mail body, encrypted session key and digital signature.

The **Crypto Module**, facilitates all the cryptographic operations of SecWEM. These operations are encryption, decryption, digital signing and verification of the signature. The encryption operation involves generation of a session key (sk) followed by encryption of the mail body (mb) using this session key. Then this session key is encrypted using the public key of each recipient ($R_{n_{PUB}}$). Encrypted message bundle is formed by the encrypted mail body ($E[sk, mb]$) appended with the e-mail addresses of the recipients ($R_{n_{Email}}$) and their corresponding encrypted session keys ($E[R_{n_{PUB}}, sk]$. The encrypted message bundle ($\{msg\}_E$) is:

$$E[sk, mb] \| R_{1_{Email}}, E[R_{1_{PUB}}, sk] \| \ldots \| R_{n_{Email}}, E[R_{n_{PUB}}, sk]. \tag{1}$$

For digital signing, the encrypted message bundle is signed using the private key of the sender (S_{Pri}) and this digital signature is appended to the encrypted message bundle. The final encrypted message bundle is:

$$\{msg\}_E \| E[S_{Pri}, H(\{msg\}_E)]. \tag{2}$$

The decryption operation involves extraction of the encrypted session key corresponding to the recipient from the encrypted message bundle. Then the encrypted session key is decrypted using the recipient's private key($R_{n_{Pri}}$).

$$D[R_{n_{Pri}}, (E[R_{n_{PUB}}, sk])] \rightarrow sk. \tag{3}$$

By using this session key the mail body is decrypted.

$$D[sk, (E[sk, mb])] \rightarrow mb \tag{4}$$

For the digital signature verification, the digital signature is extracted from the message bundle and verified using the sender's public key.

$$D[SPub, (E[SPri, H(\{msg\}_E)])] \rightarrow H(\{msg\}_E) \quad (5)$$

The **HTTP Message Rebuilder (HMR) Module**, rebuilds the webmail in the format used by particular webmail service after the successful completion of cryptographic operations. In case of the outgoing webmail, the plain mail body is replaced by the final encrypted message bundle. For the incoming webmail after successful sign verification and decryption, the encrypted mail body is replaced by the decrypted mail body. After replacing the mail body, HTTP message headers are modified accordingly.

4.3 Steps Involved in Securing the Webmail Using SecWEM

The process of sending webmail through SecWEM is depicted in Fig. 4, and it involves the following steps:

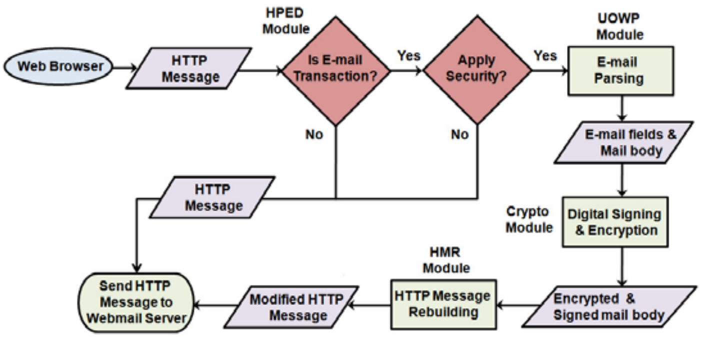


Fig. 4. Sending Webmail through SecWEM

When SecWEM gets the HTTP message from the webmail client, the HPED module determines the HTTP message type for webmail. If the HTTP message type is other than the webmail transaction or the user wants to send the webmail without applying security then SecWEM forwards these HTTP messages to its designated web server. If the user wants to send the webmail after applying security then the HPED module passes the HTTP message to the UOWP module. The UOWP module extracts the e-mail fields and the mail body from the HTTP message and passes it to Crypto module. The Crypto module performs the encryption and digital signing and passes the final encrypted message bundle to the HMR Module. The HMR module rebuilds the webmail back in the format used by the webmail service and forwards it to the webmail server.

The process of receiving the webmail through SecWEM is depicted in Fig. 5, and it involves the following steps:

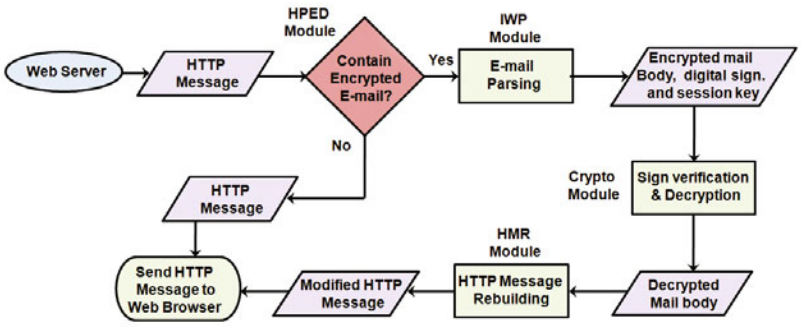


Fig. 5. Receiving Webmail through SecWEM

When SecWEM receives the HTTP message from the web server, the HPED module determines whether the HTTP message contains encrypted webmail or not. If the HTTP message does not contain encrypted webmail, SecWEM simply forwards it to the webmail client. If the HTTP message contains encrypted webmail then it passes the HTTP message to the IWP module for extracting the encrypted mail body, the encrypted session key and digital signature. Then the IWP module passes this extracted information to the Crypto module for sign verification and decryption. After successful verification of the digital signature, the Crypto module decrypts the mail body and passes the decrypted mail body to the HMR module. The HMR module rebuilds the webmail back in the format used by the webmail service and forwards it to the webmail client.

5 Implementation and Testing

The SecWEM solution is implemented using C++ programming language and network programming using sockets networking API. User interface of the SecWEM is implemented using Microsoft Visual C++. We have tested the SecWEM in a live setup. This setup includes multiple servers located at different geographical locations, the test scenario is depicted in Fig. 6. We have also successfully tested SecWEM with various webmails like Squirrel mail, Open webmail, Icewrap *etc.* in the lab environment and also with some commercial webmail service providers.

For automated public key distribution among the SecWEM users we developed a Key Management Server (KMS). The communication between SecWEM users and KMS takes place through e-mails. The SecWEM user first registers himself with the KMS and posts its public key to the KMS. The SecWEM user interface facilitates the user registration process which in turn involves sending an e-mail in a particular format to the KMS. The registration process completes with the confirmation of an e-mail from the KMS. After successful registration, the KMS sends the public keys of all other registered users to the newly registered user. The KMS also distributes the public key of the newly registered user to all the registered users. Descriptive diagram of the KMS is shown in Fig. 7.

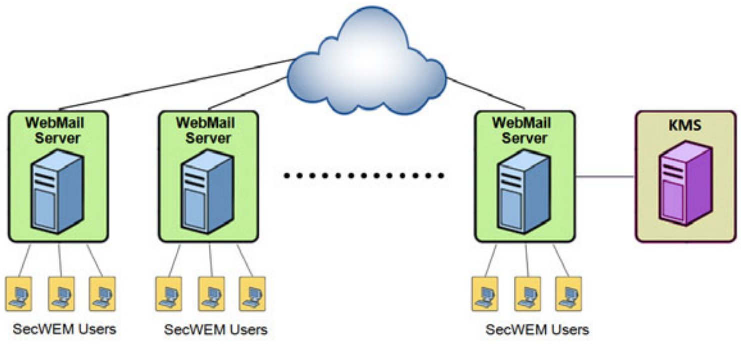


Fig. 6. SecWEM test scenario

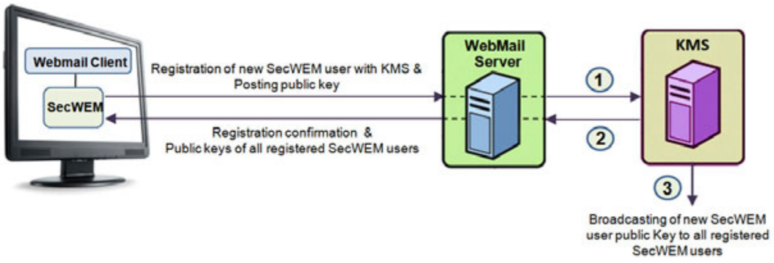

Webmail Client
SecWEM
Registration of new SecWEM user with KMS & Posting public key
Registration confirmation & Public keys of all registered SecWEM users
WebMail Server
KMS
1
2
3
Broadcasting of new SecWEM user public Key to all registered SecWEM users

Fig. 7. Key Management System

6 Performance and Overhead Statistics

For the purpose of estimating performance and overhead statistics we consider two scenarios in the test setup as described in the previous section; first the overhead while sending the message and second while receiving the message. For both the scenarios we measure the overhead for two cases, one for the plain and another for encrypted messages. Each of these measurements has been performed for various message sizes and repeated for twenty iterations to find the average time taken. To obtain these measurements we integrated a timer module inside SecWEM. The timer module is triggered by SecWEM when it receives a message from a webmail client or the server. And it stops the timer at the time of forwarding the message to its destination after completing all the processing. SecWEM requires user input for sending the message in encrypted or plain form. But the time taken by the user to provide this input will add to the overhead measurement. To avoid this we hard-code the option appropriately during the measurement for both the cases. These measurements are performed on a machine having Intel Xeon 2.8GHz CPU with 2GB RAM and 64 bit Windows 7 operating system. The results of these experiments are presented in Fig. 8 and Fig. 9. Fig. 8 shows the average time taken by SecWEM to relay the messages of various sizes for both the cases. Similarly Fig. 9 shows the average time taken by SecWEM while receiving the messages. For a

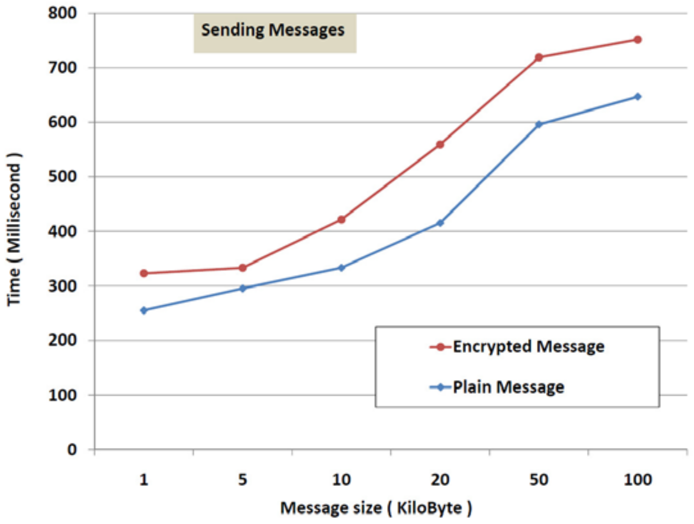


Fig. 8. Time taken for sending the message as a function of message size

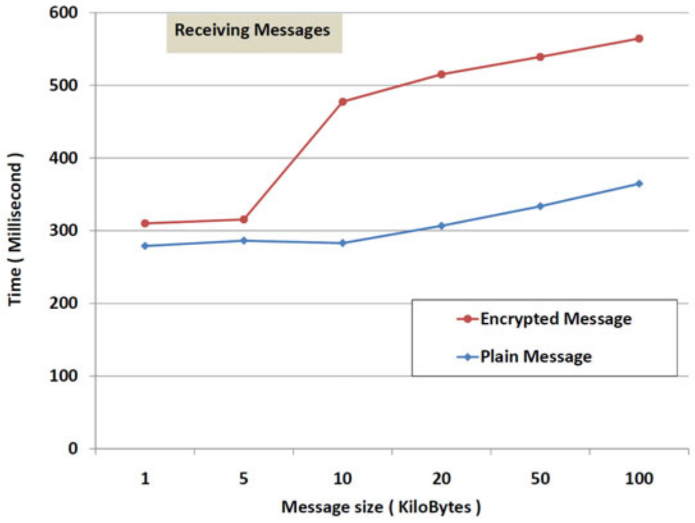


Fig. 9. Time taken for receiving the message as a function of message size

message size of 100kBs and less the maximum overhead is 752ms while sending the messages and 562ms while receiving the messages.

7 Discussion and Conclusion

In this paper we have described SecWEM, a webmail security solution to provide the security attributes of confidentiality, data integrity and data origin authentication for the webmails. For any webmail security solution the main task is to extract information regarding sender, recipient(s) and mail body of the webmail. While this is fairly straightforward in standard e-mail, because it follows well formed standards, Simple Mail Transfer Protocol (SMTP), Post Office Protocol-Version 3 (POP3) and Internet Message Access Protocol (IMAP), the webmail case involves considerable challenges in implementation as there is no standard such as SMTP, POP3 or IMAP for webmails. The major challenge in developing the security solution for webmails is that every webmail service uses its own format to transfer mails and these webmail services sometime change or modify their formats. To overcome this we have designed the recogniser module which invokes a recogniser session to find out the format. Another issue is that a webmail sends each attachment separately and the server returns a link corresponding to each attachment which is included in the webmail. Here the challenge is that, attachments does not contain recipients' addresses and without these recipients' addresses cryptographic operations cannot be performed. To overcome this problem SecWEM pops up one window, asking the user to provide the the recipients' addresses. The user can provide these addresses manually or by selecting addresses from SecWEM address-book. After getting recipients addresses, required cryptographic operations are applied to the attachments before sending them to the server.

SecWEM works as a proxy which makes it independent of the web browsers. It is a desktop solution which runs on user's desktop to provide end-to-end security for webmails. We tested SecWEM in a live setup with more than 50 domains and it works fine with them. We have successfully tested SecWEM with various webmails like Squirrel mail, Open webmail, Icewrap *etc.* in the lab environment and also with some commercial webmail service providers.We also obtained estimates of performance and overhead statistics experimentally in a lab setup. The overheads have been marginal and hence acceptable. Therefore we can conclude that SecWEM is a webmail security solution which is independent of the web browser, provides end-to-end security, and works with all webmail services.

Acknowledgments. We would like to thank Director CAIR for supporting us to work in this area. We would also like to thank Mr. Philip Abraham, Dr. Subrata Rakshit and Dr. G. Athithan for their help and constructive suggestions.

References

1. Bishop, M., Cheung, S., Wee, C.: The Threat from the Net. IEEE SPECTRUM 34, 56–63 (1997)
2. Mathew, A.R., Al Hajj, A., Al Ruqeishi, K.: Cyber crimes: Threats and Protection. In: International Conference on Networking and Information Technology, pp. 16–18 (2010)
3. Foster, I., Kesselman, C.: Simple Mail Transfer Protocol (SMTP).: RFC 5321 (2008)
4. Myers, J., Rose, M.: Post Office Protocol Version -3 (POP3).: RFC 1939 (1996)

5. Crispin, M.: Internet Message Access Protocol (IMAP),version 4rev1.: RFC 3501 (2003)
6. Fielding, R., Gettys, J., Mogul, J., Frystylc H., Masinter, L., Leach P., Berners-Lee T.: Hypertext Transfer Protocol -HTTP/1.1. : RFC 2616 (1999)
7. Entrust Secure Messaging Service, http://www.entrust.com
8. Freenigma, http://www.freenigma.com
9. Pretty Good Privacy (PGP), http://www.pgp.com
10. Ransdell, B., Turner, S.: Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.2.: RFC 5751 (2010)
11. Open PGP, http://www.openpgp.org
12. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2.: RFC 5246 (2008)
13. Stallings, W.: Network Security Essentials: Applications and Standards. Prentice Hall (2000)
14. Mohsen, T.: SMEmail - A New Protocol for the Secure e-mail in Mobile Environments. In: Australian Telecommunications Networks and Applications Conference, Adelaide, Australia, pp. 39–44 (2008)
15. Lux, K.D., May, M.J., Bhattad, N.L., Gunter, C.A.: WSEmail: Secure Internet Messaging Based on Web Services. In: IEEE International Conference on Web Services, Orlando Florida USA, pp. 75–82 (2005)
16. Jang, J., Nepal, S., Zic, J.: Trusted e-mail Protocol: Dealing with Privacy Concerns from Malicious e-mail Intermediaries. In: IEEE International Conference on Computer and Information Technology, Sydney NSW, pp. 402–407 (2008)

A Multilayer Overlay Network Architecture for Enhancing IP Services Availability against DoS

Dimitris Geneiatakis, Georgios Portokalidis, and Angelos D. Keromytis

Department of Computer Science,Columbia University, New York, NY, USA
{dgen,porto,angelos}@cs.columbia.edu

Abstract. Protection against Denial of Service (DoS) attacks is a challenging and ongoing problem. Current overlay-based solutions can transparently filter unauthorized traffic based on user authentication. Such solutions require either pre-established trust or explicit user interaction to operate, which can be circumvented by determined attackers and is not always feasible (*e.g.,* when user interaction is impossible or undesirable). We propose a Multi-layer Overlay Network (MON) architecture that does not depend on user authentication, but instead utilizes two mechanisms to provide DoS resistant to any IP-based service, and operates on top of the existing network infrastructure. First, MON implements a threshold-based intrusion detection mechanism in a distributed fashion to mitigate DoS close to the attack source. Second, it randomly distributes user packets amongst different paths to probabilistically increase service availability during an attack. We evaluate MON using the Apache web server as a protected service. Results demonstrate MON nodes introduce very small overhead, while users' service access time increases by a factor of 1.1 to 1.7, depending on the configuration. Under an attack scenario MON can decrease the attack traffic forwarded to the service by up to 85%. We believe our work makes the use of overlays for DoS protection more practical relative to prior work.

1 Introduction

Denial of service attacks (DoS), and their distributed counterparts DDoS, frequently cause disruptions, and sometimes even complete outages, of services offered over the Internet. E-mail, financial, publishing, and even e-government services have repeatedly been the targets of such attacks, which have only intensified with the proliferation of botnets that employ compromised PCs ("zombies") to perform large scale DDoS attacks. Recently, we have also witnessed incidents, where groups of users deliberately performed DDoS attacks using freely available tools, like the low orbit ion cannon [1].

Centralized approaches such as [23,12,27] have not been able to thwart DDoS attacks, mainly because they can also be congested, while distributed network level solutions [15,33,16] require operators to deploy and manage new, and potentially complex, architectures. On the other hand, distributed architectures based on overlay networks [18,28,29,6,20,21,5] *can* operate on existing network infrastructure, and have shown to be able to withstand attacks involving *millions* of attackers. Overlay based solutions can be categorized in three types:

1. Strict access: These approaches [18,6,20,5,29] assume that the users of the protected service are pre-authorized, and only them are allowed to access the service. They cater to scenarios like attacks against *emergency services* used during disasters.
2. Relaxed access: Such networks [37,36] still use some form of authentication, but they either allow anyone to "register" with the service, or mix authorized and unauthorized users. In the first case, authentication is used to uniquely identify clients without relying on their IP address (*i.e.,* spoofing is no longer relevant), while in the latter authorized user traffic is given precedence.
3. Open access: Open systems [28,29] can be accessed by everyone. They limit or distinguish attackers by using user sessions and require some type of user interaction, like a CAPTCHA [32], that separates real users from programs.

These techniques mitigate the consequences of DoS attacks using authentication or user interaction to distinguish authorized from unauthorized traffic, limiting or completely warding off the latter. However, limited access systems are not applicable to open Internet services like the world wide web (WWW), and, more importantly, they are still vulnerable to DoS attacks from authenticated users. Also, user interaction mechanisms, like CAPTCHAs, is impractical, while they are not impervious to attacks either [8].

We propose *a new multilayer overlay network* (MON) architecture that builds on the advantages of ticketing and multi-path traffic distribution proposed in previous relaxed access work [29] to *enable open access* to the protected service *without* requiring any user interaction. Our solution operates on existing network infrastructure and consists of multiple layers of nodes that are interposed between the client and the protected service. MON *collectively* applies a throttling-based DoS prevention mechanism (DPM) that alleviates the effects of DoS attacks. The mechanism is applied in a distributed fashion by lightweight DoS detection and mitigation engines (DDME) running on each overlay node. A client accesses the service by contacting any of the nodes of the first layer, which randomly routes his packets through the overlay, increasing MON's robustness in the case of node failures and DoS attacks.

We implemented a prototype using an uncomplicated practical threshold-based filtering mechanism. Briefly, the DDME on each node monitors the IP packets being send to the service, calculating the packet sending rate of each client based on his IP address. When a client exceeds a predefined threshold, some or all of his packets are dropped depending on the employed security policy. Note that we do not invent a new defense against DoS attacks, but instead propose the distributed application of prevention mechanisms based on overlay network architecture. Results show that the overhead introduced by MON nodes is small (in the range of 30 and 50 microseconds), while using MON to retrieve a web site served by the Apache web server increases the client's access time by a factor of 1.1× to 1.7×, depending on the configuration. Moreover, we demonstrate that the proposed solution is able to throttle and block malicious network flows effectively when a service is under a DoS attack.

The rest of the paper is organized as follows. In Sect. 2, we describe the types of DoS attacks that we tackle with this work. In Sect. 3, we describe the architecture, while we discuss implementation details in Sect. 4. In Sect. 5, we evaluate MON in terms of overhead and effectiveness. Sect. 6 presents an overview of the related work, and compares it with ours. Finally, we conclude this paper and give some pointers for future work in Sect. 7.

2 Threat Model

The goal of DoS attacks is to render a targeted services inaccessible to legitimate users, by draining server resources like memory and CPU, or consuming network bandwidth [22]. The simplest DoS attacks consume network bandwidth and server resources simply by saturating the target with a high number of requests, generated either from a single or multiple sources. However, as server processing power and network bandwidth increases alternative, more sophisticated, methods requiring fewer connections have been devised. For instance, a flood of SYN TCP packets can be used to prevent the establishment of new connections with a server [9]. Other approaches exploit the way the service handles users' requests to achieve the same goal [7]. Based on their method of operation, we classify DoS attacks into the following categories:

(a) *Attacks consuming network bandwidth.* These attacks aim to congest the target's network by generating an overwhelmingly large number of data packets.
(b) *Attacks consuming server resources.* Malicious users send a large number of otherwise legitimate requests to the server to consume its resources. They usually require less traffic than (a) to be sent by a malicious user, as network bandwidth increases at a higher rate than computational power and storage.
(c) *Attacks consuming limited operating system (OS) resources.* These attacks exploit the way the target's software and OS operates to consume limited resources such as file and socket descriptors, and TCP connections.
(d) *Attacks exploiting server bugs.* Software frequently has bugs like null pointers [14] and deadlocks [17]. If such bugs can be triggered using user input, an attacker sending the appropriate data to the server can cause it to crash, or simply stop responding to new requests. This type of attacks are beyond the scope of this paper, since are tackled by solutions such as [34,10].

3 A Secure Multilayer Overlay Network Architecture

The goal of the MON architecture is to improve the availability of critical services under a DoS attack, by "hiding" the protected service behind a semistructured overlay network. Users of the service communicate with it through the overlay, which by design and by means of a distributed DoS prevention mechanism (DPM) mitigates DoS attacks. We adopt an overlay network architecture to incorporate certain properties in our design. Specifically:

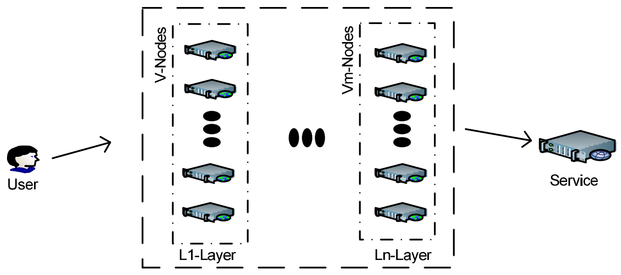


Fig. 1. The multilayer overlay network (MON) architecture. Multiple layers of nodes are interposed between the clients and the provided service. The number of layers is configurable at overlay setup, and can be tuned to favor performance over resistance to intrusion by reducing the degree of internal redirection.

(a) Easy deployment on existing network infrastructure.
(b) Transparency to end-users.
(c) Decentralized design inherently strong against DoS attacks, and suitable for applying intrusion prevention mechanisms in a distributed way.

An overview of the MON architecture is illustrated in Fig. 1. The first layer is the entry point between users and the service. It ensures that no spoofed traffic is forwarded to the protected service based on tickets (discussed in Sect 3.2), and blocks attacks using the DPM. The rest of the layers remain invisible to end-users, while they also use the DPM to protect from malicious or compromised nodes of the previous layer. However, if an attacker discovers and compromises a node in the last layer, the service will be exposed. The number of layers deployed, depends on the network delay that the protected service can tolerate, and the desired protection level that must be established.

MON provides open access to users, without requiring user interaction or pre-established trust between clients and the provided service. Instead, it throttles user traffic that exhibits abnormal behavior on a per- flow and ticket basis. Also, MON operates at the IP level, supporting multiple transport layer protocols like TCP, UDP and SCP.

3.1 The MON Structure

MON consists of a number of overlay-nodes N_n, which are organized in different layers L_m. Clients can access the protected service through the first layer (entry point) nodes after acquiring an access ticket. Tickets protect against DoS attacks themselves, as they allow us to control the rate new clients can connect to the network, but they also allow us to validate the sender of each packet. The nodes of the first layer know the nodes of the next one, where they forward packets to, and so forth (*i.e.*, nodes of L_i know the nodes L_{i+1}). As a result, the actual location (IP address) of the service is only known by the nodes in the last layer.

Also nodes instead of routing packets through a specific path, they randomly route it in one of the available nodes of the next layer. The last hop of the

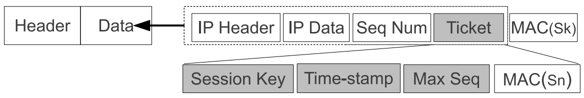


Fig. 2. MON packet structure. User's IP traffic is encapsulated into a new transport layer packet to enable routing by MON nodes, including an encrypted access ticket and message authentication code to ensure MONs' message integrity, and authenticity. Only entry nodes can decrypt and validate the ticket.

employed architecture delivers the packets to the protected service. If one of the nodes becomes "unavailable" due to network congestion or software failure, traffic is forwarded through a different node in the same layer.

Consequently, an attacker would have to concurrently flood all nodes of at least one layer of the overlay to successfully disrupt the service. Assuming that the layer with the smallest number of nodes is L_v and contains V nodes, the attacker would have to successfully flood all V nodes. However, since the route followed by packets is chosen randomly, and every node implements the DPM, which throttles flows classified as malicious, this task is far from trivial. Even if an attacker compromises a first layer node (*e.g.,* by exploiting a vulnerability in its software) and uses it in its DoS attack, the next layer nodes can identify and mitigate the attack, while the service remains hidden. *Hence, MON is resistant to DoS attacks by design.* All the nodes in MON architecture operate autonomously. MON encapsulates at the client side (see also Sect. 3.2) user packets into a new packet (similarly to IP-in-IP encapsulation), as illustrated in Fig. 2.

3.2 Users and the Ticket Mechanism

Users access a MON-protected service through the first layer's nodes. This is done transparently, by installing an *end-user module* at the client, that captures data packets destined for the service and re-routes them to one of the entry points of MON. To overcome single points of failure, users' traffic is randomly distributed among the nodes of L_1, similarly to the random routing described in Sect. 3.1. MON-enabled users are allowed to connect to the protected service, only if they have acquired a session key and the corresponding ticket from an entry point. Particularly, a client receives a session key and a valid ticket by issuing an access request to a randomly chosen entry node. The request is performed over a secure channel (*e.g.,* using SSL) to ensure the confidentiality of the session key sent to the client.

Every session key S_k is computed based on the user's IP and a pseudo-random id generated by the contacted node, encrypted using a master key K_n shared among all MON nodes using the following formula.

$$S_k = Enc(K_n, User\ IP || Random\ Id)$$

The *ticket* sent to the user includes the user's session key, a time-stamp, the maximum number of packets allowed to be sent with the specific ticket, and a message authentication code (MAC) generated using the master key. The latter enables any MON-node to validate the integrity and authenticity of every ticket. Finally, the entire ticket is also encrypted using the master key and sent to the user as a response.

$$ticket = Enc(K_n, session\ key||timestamp||MAC(K_n))$$

Clients include the ticket and a MAC based on their session key in all subsequent packets (see Fig. 2). MON nodes are able to decrypt the ticket and validate MON packet's integrity and authenticity, using the shared master secret key. This way an attacker cannot spoof a client's packets (*i.e.,* send packets pretending to be another client), and nodes can validate tickets without having to store additional information for the session key and the ticket.

3.3 A Collaborative DoS Detection and Mitigation Mechanism

MON is robust by design, but as the resources of the attackers grow, additional protective measures are necessary. Although, various centralized intrusion detection and prevention solutions against DDoS attacks have been proposed in literature [25], they can also be congested and rendered useless by the attack against the protected service. MON adopts a collaborative DoS prevention mechanism (DPM) that is applied individually by every node, and is thus more resistant to attacks.

The core of the mechanism is implemented by the DoS detection and mitigation engines (DDME) running on the nodes. Its goal is to throttle or drop incoming packets, whenever it detects that a client is performing a DoS attack. It works by classifying incoming packets into IP network flows based on their source, destination IP addresses, and the ticket (as it uniquely identifies a client). This way, we can detect abnormal behavior and match it to a specific user. We use IP layer flows, primarily because we desire that MON protects services independently of the transport and application protocol being used. Secondly, this will allow us to easily expand MON to support the IPv6 protocol.

Each DDME keeps track of all the incoming flows using a flow information monitor (FIM). The FIM records the number of packets observed per flow, and the time that each flow was initiated. Then, the DDME periodically examines (with a period T) whether the packet rate of any flow has exceeded the legitimate configurable threshold. If this is the case, packets belonging to this specific flow are "punished" that is, delayed or dropped. For the scope of this work, we adopt an exponential punishment mechanism, as in [2]. This technique is widely used in networking to penalize users causing conflicts during transmission. The delay introduced by the DDME for "punished" flows is computed by the following formula $delay = delay \times 2$. Devising mechanisms to calculate an appropriate threshold is beyond the scope of this paper. In the future, we plan to investigate threshold calculation mechanisms like the ones proposed in [15].

4 Implementation

4.1 Ticket Acquisition

A user inquires a ticket from the service by sending a ticket acquisition request to a randomly chosen MON node. The request contains the user's RSA public key and the corresponding digital signature. As soon as a node receives a ticket request, it validates the included digital signature and generates the response message, which consists of the session key and the ticket as described in Sect. 3.2. Note that the session key and the ticket are encrypted using the user's RSA public key and the AES algorithm. To compute the MAC, both for the ticket and the response message, we use an HMAC function based on SHA-1. All the cryptographic functions used are part of the OpenSSL library [31].

4.2 MON-enabled Users

On the client side, we developed a service, namely MONeU, operating transparently to existing applications in order to deliver user traffic to MON nodes. User packets destined to the protected service, instead of following the normal network path, are sent to a virtual interface implemented using the tun pseudo-device driver [19]. As soon as a packet is received in this interface, the MONeU service encapsulates it in a new UDP packet, which includes the ticket, the packet's sequence number, and an SHA-1 HMAC computed on the whole MON packet to protect its integrity. This new packet is forwarded to a randomly chosen first layer MON node. The technique shields end-users from single point failures, as their traffic does not follow a specific path. This stands even for packets belonging to the same session. As a result, a malicious user needs to compromise all available MON entry nodes to cause a DoS to a legitimate user.

In the current implementation the available first layer nodes are included in a pre-defined list. However, an alternative solution for MONeU to receive the list is through DNS. In that case, the DNS instead of returning the actual IP address of the protected service, will send back the IP addresses of all available first layer nodes.

Whenever a response packet arrives, the MONeU service passes it to the virtual interface, which is responsible for delivering it to the corresponding application. The decision to use UDP instead of TCP for packet encapsulation is based on the fact that the encapsulation of TCP within TCP is not efficient [30].

4.3 MON Nodes

MON nodes are the core of the proposed architecture and are responsible for detecting and mitigating the effects of malicious flows (*i.e.,* flows responsible for a DoS attack), and routing user traffic to the actual destination. First, entry nodes validate MON packet authenticity and integrity. To accomplish this task the nodes decrypt the ticket using AES and the secret key S_n shared among the MON nodes. They also ascertain its validity using the same key to validate the

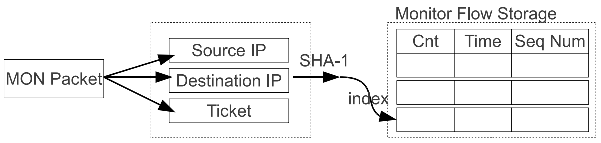


Fig. 3. MON flow information monitor (FIM) architecture. FIM uses the hash value of the source and destination IP as an index to the monitor flow storage, and records the number of packets per IP flow, the time-stamp of the first packet in the flow, as well as MON packet sequence number.

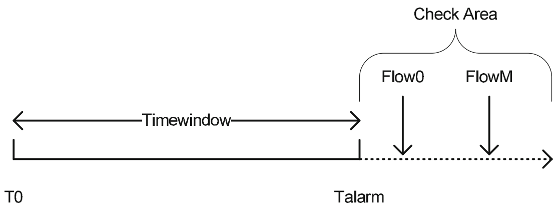


Fig. 4. DDME's time-window structure. The DDME checks all the flows exceeding the allowed time in FIM.

SHA-1-based HMAC. If the ticket is valid, we extract the session key S_k, which is used to confirm the authenticity and integrity of the whole packet, similarly to the process used to validate the ticket.

After message and ticket validity is confirmed, the node extracts the source and destination IP, and combines them with the ticket in order to uniquely identify each flow. The FIM module records for every flow the number of received packets, the arrival time of the first packet in the flow, and the sequence number of the last received packet to protect MON against replay attacks. All numbers are 32-bits long, so we store 96 bits per flow, which corresponds to 12 MBs of data for one million users. For storage and searching efficiency, instead of recording the source and destination IP for every flow, we use the remainder of the hash value of the flow information with the maximum number of allowed flows (see the formula below) for identifying a record in the FIM storage.

$$index = hash(SrcIP||DestIP||Ticket)\ mod\ MAXCON$$

The main disadvantage of this approach is that if we receive more connections than FIM's capacity ($MAXCON$), it will cause a "conflicting" FIM record, affecting the DDME's accuracy. Otherwise, "conflicts" in the FIM storage solely depend on the hash function's collision probability [35]. For every received packet, the FIM computes the flow's index in order to update the packet counter, and saves the timestamp of the first packet in the flow (t_{flow}). Figure 3 illustrates FIM's general architecture. Note that the hash and modulo functions we use are OpenSSL's SHA-1 [31] and GMP's modulo [13].

The DDME is triggered by a SIGALARM signal every T_{window} seconds in order to check whether any flow in the FIM has exceeded a pre-defined threshold.

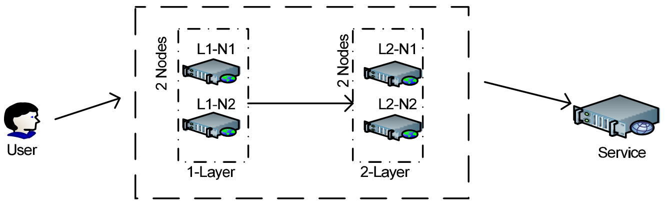


Fig. 5. The two-layer test-bed we employed for the evaluation of the MON architecture

If this is the case, the packet is either punished by introducing a delay to its forwarding. Note that the DDME inspects all the flows found "outside" of the T_{window}, where $t_{alarm} - t_{flow} > T_{window}$ (see also Fig. 4). As soon as the DDME checks a flow, it resets its corresponding record in the FIM.

Afterwards, the DDME encapsulates the incoming packet into a new UDP packet, and forwards it to a randomly selected node at the next layer. When the packet reaches a node in the last layer of MON, it is decapsulated to obtain the original packet, and is forwarded to the protected service via a RAW socket.

For implementing the networking functionality of MON nodes, we relied on Linux's system calls. Based on current implementation MON nodes can be deployed on routers, as well as desktop PCs, that are willing to participate in the MON architecture, assuming that they are running a *nix type OS.

5 Evaluation

To evaluate our architecture in terms of performance and effectiveness, we deployed a two-layered prototype as depicted in Fig. 5. Table 1 lists the technical characteristics of the systems where we installed MON nodes. As a protected service, we utilized the Apache web server. We employed five scenarios, where a client is requesting files of different sizes from the Apache web server (sizes of 150KB, 500KB, 1MB, 5MB, and 13 MB). In all five scenarios, the client accesses the service *directly*, using MON *without DPM* functionality, and lastly *with DPM* functionality. Furthermore, to validate MON's correctness and efficiency, all the scenarios were executed in two different configurations:

1. All MON nodes running in our lab's local network (LAN configuration).
2. MON's 1^{st} layer nodes running in a different network connected to the Internet with an ADSL connection, while the 2^{nd} layer nodes were in the lab's network (ADSL configuration).

5.1 Performance Evaluation

To quantify the overhead MON introduces to the protected service, we measured the end-to-end service access time as well as the overhead introduced by the MON nodes itself. On both configurations, when users access the service using MON, they do not experience any apparent delay in service access time (SAT).

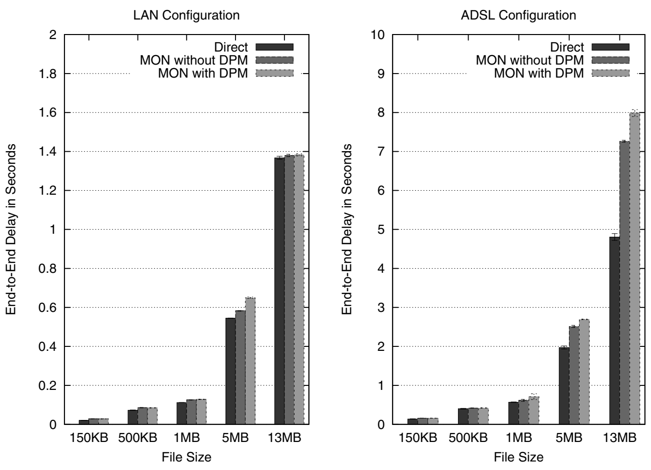


Fig. 6. End-to-end service access time (SAT) comparison among direct access, and MON without and with DPM. User SAT is increased by a factor of 1.1 to 1.7, depending on the configuration.

The average SAT is very close in all the scenarios, as depicted in Fig. 6. In the results, we include the case of using MON without the DPM functionality, in order to show the lower bound of overhead imposed by the overlay network itself. In the LAN configuration, MON increases the end-to-end SAT by a factor of 1.04 to 1.4, while in the ADSL configuration by a factor of 1.1 to 1.7. Though this increase might be considered significant, we believe that is an acceptable "cost" for enhancing IP service availability. Note that the total transfer time is affected by the network link's quality in which MON nodes are installed. For instance, the % difference in average SAT between the LAN configuration and direct access ranges from 4% to 40%. The same stands for the ADSL configuration, however, in scenario 5 there is an increase of 66%.

Regarding the overhead introduced by entry and core MON nodes, the average ranges between 30 and 50 microseconds (see Fig. 7). As we use the same underlying infrastructure for both configurations, there are no differences in the delay introduced by each MON node. This fact validates our initial hypothesis regarding the end-to-end SAT fluctuations in the ADSL configuration.

Based on our experimental results, we deduce that MON does not influence the SAT, while the overhead imposed on the provided service by each MON

Table 1. Characteristics of the systems hosting the MON nodes in the test-bed

Name	CPU Speed	Memory	Other characteristics
L1-N1	2.4 GHz	4Gb	Ubuntu 10.04, Intel Core i5 laptop
L1-N2	2.4 GHz	2Gb	Ubuntu 10.10, virtual machine
L2-N1	2.8 GHz	3Gb	Ubuntu 10.04, Xeon 2xCPU
L2-N2	2.4 GHz	2Gb	Ubuntu 10.10, virtual machine

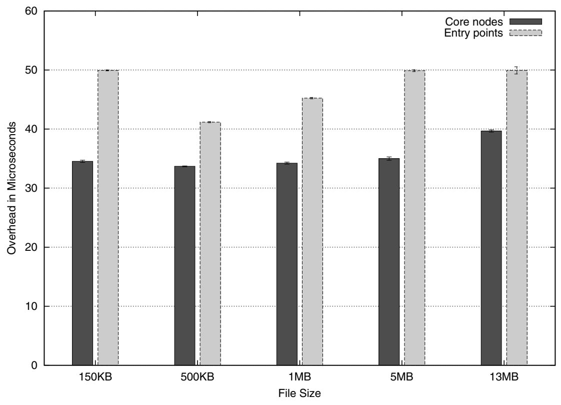


Fig. 7. Average processing time introduced by entry points and core nodes in MON

node is negligible. However, it should be noted that the end-to-end delay is affected both from the number of MON layers, and the latency of the underlying network where the MON nodes are located. In the extreme case, we can optimize performance by reducing the system to one level of indirection similar to [29].

5.2 Qualitative Analysis

MON introduces little delay to end user communication with a protected service, however, its effectiveness as a protection mechanism against DoS attacks must be evaluated as well for its ability to identify and punish DoS packets. Thus, we employ a single source attack scenario in which the malicious user generates 1000 HTTP requests per second for a web page of 100 bytes. Regardless of the simplicity of this attack, the main goal of this experiment is to demonstrate MON's efficiency during such an attack, by monitoring the number of requests received by the protected service.

On the one hand, if the service is protected through traditional centralized DoS mechanisms, all the attack requests will arrive at least up to the protected service network causing a congestion either at the edge of the network or in the service itself. On the other hand, if MON is enabled it will forward all the packets belonging to the attack flow toward the service until the DDME is triggered. In the worst case, MON will deliver to the service $Tw \times Number\ of\ Packets\ Per\ Second$, however, the number of packets received by the service depends on the Tw that the DDME is triggered, and the punishment model. Since, the DDM is triggered the attack will be identified and traffic will be throttled. Particularly, under our attack scenario the number of requests received by the service is on average 107 HTTP requests, which corresponds to 323 packets. The majority of these packet are delivered to the service because the DDME had not triggered yet. Using MON we achieve a reduction on the attack traffic receive by the service up to 85%. The number of times a flow exceeds the legitimate threshold affects the punishment delay introduced on a flow exponentially, as described in Sect. 3. So if a malicious user exceeds more than 10 times the threshold the introduced delay reaches to 1200 seconds.

Similar is the case of multiple sources DoS attack, as we can consider it as N separate single source DoS attacks. The only difference is the amount of traffic that will be delivered by MON towards to the service until the DDME is triggered, which is $N \times Tw \times Number\ of\ Packet\ Per\ Second$. Note that a malicious user can still spoof an IP address and get a valid MON ticket, but he is not able to affect the communications of other users, as MON distinguishes flows on a per ticket basis. However, to effectively shield a service against these types of DoS attacks, MON nodes should be widely deployed across the Internet. Also, additional IP spoofing protection mechanisms would further fortify the system.

All in all, MON can be used to defend against DoS attacks targeting service network bandwidth or server resources (briefly described in Sect. 2) by exploiting the autonomously operating MON nodes to detect and discard DoS traffic.

6 Related Work

Overlay networks emerged as a solution to overcome the limitations of the Internet architecture, providing new services and applications in a distributed end-to-end fashion. To that end, previous work build on overlay networks as an alternative underlying mechanism to enhance Internet service security and availability. The very first work exploiting overlay networks to enhance service availability is RON [4]. RON introduces an application-level routing and packet forwarding service that enables end-points to monitor the quality of links in the network, and detect path failures in order to choose an alternative path in a few seconds. However, the predominant work exploiting overlay networks to improve network service security is presented in [18]. The SOS architecture routes traffic to the protected service only from a set of pre-determined authorized users, based on protocols such as IPsec [26] and SSL [31]. A similar approach is presented in [3].

Several variations of SOS have been proposed in literature [28,29,20,21,6,5]. [28] extends its functionality to defend against botnets using CAPTCHA [32], while [29] introduces an architecture for enhancing service availability build on multi-path and stateless tokens. In ODON [20], users are verified through credentials to access the overlay network. They establish a session token among end-users, ODON nodes, and the final service used to verify traffic, and provide integrity and confidentiality services. [21] proposes a federated overlay network (FONet) architecture to protect services from DoS attacks. FONet forwards only the traffic originating from the federation to the protected service, and filters the other traffic on the edge of the domains participating in the FONet. [5] introduces an intermediate interface by overlay nodes to hide the location of application sites during DoS attacks. Similarly, this solution allows communication only among confirmed entities; meaning that any packet must be authenticated through the proposed architecture. [6] protects a target using a filter that drops any packet whose source address is not approved. In the case of a DoS attack, rather than processing all arriving packets, the target's filter processes only a subset of received packets, and drops all the remaining.

Phalanx [11] follows a similar approach to SOS, leveraging the power of swarms to combat DoS. A client communicating with a destination sends its packets through a random sequence of entities called "mailboxes". Mailboxes discard user packets that do not include an authentication token, or a solution to a cryptographic puzzle. A protected destination receives packets only from specific mailboxes, which were authorized in a previous communication. The only approach utilizing a distributed intrusion detection system (IDS) is presented in DefCOM [24]. DefCOM nodes collaborate during an attack to spread alerts, and protect legitimate traffic based on local classifier, which limits attack traffic.

Most of the existing overlay solutions, that were discussed above, have been influenced by SOS [18]. These solutions rely on authentication mechanisms, requiring pre-established trust or user interaction, to filter unauthorized traffic and defend against DoS attacks. None of them except SOS [18], WebSOS [28] and [29] operate transparently to end-users, and build on existing network infrastructure. While [24] is the only solution exploiting an IDS mechanism, but assumes that such a protection mechanism already exists.

7 Conclusions and Future Work

In this paper, we proposed and implemented a distributed and transparent to end-users architecture overlay network to counter DoS attacks, that does not require modifications to the underlying network infrastructure. The proposed architecture is based on a multi-layered semistructured overlay network, which is DoS resistant by design, and uses filtering to stop DoS attacks close to their source. We believe that our work makes the use of overlays for DoS protection more feasible compared with previous work.

We evaluated MON using the Apache web server as the protected service. Results shows that it has little effect on user experience, and it can effectively detect and mitigate DoS attacks against the WWW and similar Internet services like FTP and e-mail. However, additional analysis should be done for real time services. For future extensions to MON, we are considering additional protection mechanisms that can be incorporated into our DPM to also identify and prevent other types of DoS attacks.

Acknowledgements. This work was supported by the National Science Foundation through Grant CNS-07-14277. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or the NSF.

References

1. Abatishchev: Low orbit ion cannon, `http://sourceforge.net/projects/loic/`
2. Abramson, N.: THE ALOHA SYSTEM: another alternative for computer communications. In: AFIPS 1970 (Fall): Proceedings of the fall Joint Computer Conference, November 17-19, pp. 281–285. ACM (1970)

3. Andersen, D.G.: Mayday: Distributed Filtering for Internet Services. In: Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems, Seattle, WA (March 2003)
4. Andersen, D.G., Balakrishnan, H., Kaashoek, M.F., Morris, R.: The case for resilient overlay networks. In: Proceedings of the 8th Workshop on Hot Topics in Operating Systems, p. 152. IEEE Computer Society (2001)
5. Beitollahi, H., Deconinck, G.: An overlay protection layer against denial-of-service attacks. In: Proceeding of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1–8 (April 2008)
6. Beitollahi, H., Deconinck, G.: FOSeL: filtering by helping an overlay security layer to mitigate DoS attacks. In: Proceedings of the IEEE International Symposium on Network Computing and Applications, pp. 19–28. IEEE Computer Society (2008)
7. Chee, W.O., Brennan, T.: Slow HTTP POST DoS attacks. OWASP AppSec DC (2010), `http://www.owasp.org/images/4/43/Layer_7_DDOS.pdf` (November 2010)
8. Chellapilla, K., Simard, P.Y.: Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). In: Advances in Neural Information Processing Systems (NIPS), vol. 17, pp. 265–272. MIT Press (2005)
9. Cheswick, W.R., Bellovin, S.M., Rubin, A.D.: Firewalls and Internet security: repelling the wily hacker. Addison-Wesley (2003)
10. Cretu-Ciocarlie, G.F., Stavrou, A., Locasto, M.E., Stolfo, S.J.: Adaptive anomaly detection via self-calibration and dynamic updating. In: Kieda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 41–60. Springer, Heidelberg (2009)
11. Dixon, C., Anderson, T., Krishnamurthy, A.: Phalanx: withstanding multimillion-node botnets. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2008, pp. 45–58 (2008)
12. Gil, T.M., Poletto, M.: MULTOPS: a data-structure for bandwidth attack detection. In: Proceedings of the 10th USENIX Security Symposium (August 2001)
13. GNU: The GNU multiple precision arithmetic library, `http://gmplib.org/`
14. Hovemeyer, D., Pugh, W.: Finding more null pointer bugs, but not too many. In: Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE), pp. 9–14 (2007)
15. Ioannidis, J., Bellovin, S.M.: Implementing Pushback: Router-based defense against DDoS attacks. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (February 2002)
16. Ioannidis, S., Keromytis, A.D., Bellovin, S.M., Smith, J.M.: Implementing a distributed firewall. In: Proceedings of the 7th ACM Conference on Computer and Communications Security, CCS 2000, pp. 190–199. ACM (2000)
17. Jula, H., Tralamazza, D., Zamfir, C., Candea, G.: Deadlock immunity: enabling systems to defend against deadlocks. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI), pp. 295–308 (2008)
18. Keromytis, A.D., Misra, V., Rubenstein, D.: SOS: secure overlay services. In: Proceedings of the 2002 SIGCOMM Conference, pp. 61–72 (August 2002)
19. Krasnyansky, M.: Virtual point-to-point (TUN) and ethernet (TAP) devices, `http://vtun.sourceforge.net/tun/`
20. Kurian, J., Kulkarni, A., Vu, H.T., Sarac, K.: ODON: an On-Demand security overlay for Mission-Critical applications. In: Proceedings of the International Conference on Computer Comm. and Netw., pp. 1–6. IEEE Computer Society (2009)

21. Kurian, J., Saraç, K.: Provider provisioned overlay networks and their utility in dos defense. In: Proceeding of the IEEE GLOBECOM, pp. 474–479 (2007)
22. Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: Internet Denial of Service: Attack and Defense Mechanisms, illustrated edn. Prentice Hall (January 2005)
23. Mirkovic, J., Reiher, P.: D-ward: A source-end defense against flooding denial-of-service attacks. IEEE Trans. Dependable Secur. Comput. 2, 216–232 (2005)
24. Oikonomou, G., Mirkovic, J., Reiher, P., Robinson, M.: A framework for a collaborative ddos defense. In: Proceedings of the 22nd Annual Computer Security Applications Conference, pp. 33–42. IEEE Computer Society (2006)
25. Patcha, A., Park, J.M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Comput. Netw. 51, 3448–3470 (2007)
26. Kent, S., Seo, K.: Security architecture for the internet protocol. RFC 4301 (December 2005)
27. Siris, V.A., Papagalou, F.: Application of anomaly detection algorithms for detecting syn flooding attacks. Comput. Commun. 29, 1433–1442 (2006)
28. Stavrou, A., Cook, D.L., Morein, W.G., Keromytis, A.D., Misra, V., Rubenstein, D.: WebSOS: an overlay-based system for protecting web servers from denial of service attacks. Computer Networks 48(5), 781–807 (2005)
29. Stavrou, A., Keromytis, A.D.: Countering dos attacks with stateless multipath overlays. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 249–259. ACM, New York (2005)
30. Titz, O.: Why tcp over tcp is a bad idea, http://sites.inka.de/bigred/devel/tcp-tcp.html
31. Viega, J., Messier, M., Chandra, P.: Network Security with OpenSSL, 1st edn. O'Reilly Media (June 2002)
32. Von Ahn, L., Blum, M., Langford, J.: Telling humans and computers apart automatically. Commun. ACM 47, 56–60 (2004)
33. Wang, H., Zhang, D., Shin, K.G.: Change-point monitoring for the detection of dos attacks. IEEE Trans. Dependable Secur. Comput. 1, 193–208 (2004)
34. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
35. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199 (2004)
36. Yaar, A., Perrig, A., Song, D.: SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 130–143 (2004)
37. Yang, X., Wetherall, D., Anderson, T.: A DoS-limiting network architecture. In: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Comm., pp. 241–252 (2005)

Mitigation of Malicious Modifications by Insiders in Databases

Harini Ragavan and Brajendra Panda

Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, Arkansas 72701, USA
{hragavan,bpanda}@uark.edu

Abstract. Insider threat is considered as a serious issue in all organizations. Sophisticated insiders can override threat prevention tools and carry on their attacks with new techniques. One such technique which remains to be an advantage for insiders to attack a database is dependency relationship among data items. This paper investigates the ways by which an authorized insider detects dependencies in order to perform malicious write operations. The paper introduces a new term 'threshold', which defines the constraints and limits a write operation could take. Having threshold as the key factor, the paper proposes two different attack prevention systems which involve log and dependency graphs that aid in monitoring malicious activities and ultimately secure the data items in a database. Our proposed systems continuously monitor all the data items to prevent malicious operations, but the priority is to secure the most sensitive data items first since any damage to them can hinder the functions of critical applications that use the database. By prioritizing the data items, delay of the system is reduced in addition to mitigating insider threats arising from write operations.

Keywords: Database, Threshold, Log, Dependency Graph, Insider Threat.

1 Introduction

Most of the existing organizations make use of computer systems to store, process and distribute information regarding their day to day activities. These information systems which contain valuable data are confronted with a variety of threats originating from both inside and outside the organization. It is therefore highly essential for every organization to maintain high level of security since failure of security could lead to unauthorized disclosure, modification, or interruption of information. Various research works show that even though attacks such as hacking, viruses etc. arise from the outside and cause heavy damage, insider threats pose a significantly higher level of risk than outsiders do [1].

Basically, insiders are trusted persons. They have knowledge of the information systems they use and the services used in the organization. They also have knowledge about the security measures that have been taken to protect valuable information. Since they are aware of these measures and policies, they have the ability to violate or go around them. Thus certain attacks would go undetected for some time.

This paper investigates the problem of insider threat in database systems. It mainly deals with the threats posed by insiders who have authority to change and modify information in the database. Generally, insiders with write permissions can input any data into the database which at times becomes a serious threat for the organization. Our proposed system catches those malicious write operations by associating a new term called 'threshold' with every data item in the database. Threshold defines the limit to which a data item could be modified. One may argue that an insider with the knowledge of a threshold for a data item can modify the data item maliciously by keeping the changes within the threshold. However, the thresholds are determined in such a way that, any value less than the threshold is within acceptable risk and causes no problem to the system. Furthermore, thresholds can be adjusted as needed. These concepts are discussed in the paper later.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 discusses our proposed model and explains the terms used in the paper. Section 4 and 5 contains the algorithms and working of our models. Section 6 is a comparison between the models followed by section 7 which gives the conclusion and states our future work.

2 Background and Related Work

The term 'insider' carries many definitions and few among them are discussed here. In a paper [2], the authors define an insider as *"someone with legitimate access to an organization's computers and networks. For instance, an insider might be a contractor, auditor, ex-employee, temporary business partner, or more".* In another paper by Bishop and Gates [3], they address an insider as *"a trusted entity that is given the power to violate one or more rules in a given security policy... the insider threat occurs when a trusted entity abuses that power."* In [4], an insider is defined as, *"Anyone with access, privilege, or knowledge of information systems and services".* Also, numerous methods to prevent insider threats have been introduced till date. In a paper by Spitzner [5], honeypots have been used to detect insider threat. They discuss the ways of indicating insider threats by combining honeypots with honeytokens and honeynets. Apart from this, various mechanisms like attack graphs, trees have been proposed in many papers. One such paper [6] uses attack trees as a framework to monitor the activities of users and also to catch them if their target node is found along the tree. Adding to this, in [7] use of attack decomposition trees and attack vs. defense matrices for insider threat defense is discussed.

A paper by Yaseen and Panda [8] discusses how an unauthorized insider can acquire knowledge about a relational database by detecting the dependencies between objects. Sufficient amount of work has been performed in preventing insiders, who build their knowledge by exploring dependencies in order to access sensitive information. Our model also deals with dependencies, but in contrast to their work, we deal with malicious write operations. Similar work has been done in paper [9] which discusses few methods to predict malicious activities by insiders who combine object dependencies and their knowledge gained in order to launch an attack. In [10], the authors proposed a model called a key challenge graph, which describes various paths an insider could take to reach the target node. They say that every time a node is compromised, additional information is gained which helps in continuing the attack.

Since dependencies remain to be a major issue in insider threat mitigation, many researchers have discussed about it extensively in [11] [12] [13] [14]. In [11][12] the authors talk about the problem of combining non-sensitive data to derive sensitive data. They present a survey of the current and emerging research works in data inference controls. In [13] and [14] the authors investigate the problem of inference channels which arises when one combines non sensitive data to get sensitive information. An integrated architecture to detect various types of insiders was proposed by Maybury et al. in [15]. This paper summarizes few works that have been carried out for counter attacks on insiders. Bradford and Hu [16] combined intrusion detection mechanisms with forensics tools to detect insiders in a layered approach. They have employed intrusion detection systems to drive the forensic tools. Morgenstern [17] formulated a function to compute the amount of knowledge an insider could attain by detecting dependencies. In the paper [18], the authors advocate a model called the Capability Acquisition Graph (CAG) which analyzes the individual actions of users to evaluate their cumulative effects and detect possible violations. To complement these existing works, our proposed models aim at preventing malicious write operations. The following section describes our attack prevention system and explains few key terms used in the paper.

3 The Attack Prevention System

Insiders are trusted entities of an organization who have the authority to perform various operations, and when they misuse their permissions and violate rules, it turns out to be a threat. As mentioned earlier, good amount of work exists in preventing malicious read operations; so this paper focuses on preventing malicious write operations in databases. A write operation can modify a data item by one of the following ways:

- Direct access.
- Modifying some data item which will trigger an indirect update[Indirect access].

Direct Access. Here, an insider will have write permissions on the data item he/she is trying to modify. So as an authorized user, he will be trusted and he can make modifications directly on the object.

Indirect Access. In this case, an insider might not have direct permissions to modify the data item as such, but he can still work on it by figuring out its dependencies. This means, when the dependent data item is changed, it makes a change in the target item or might change few intermediate data items which get reflected in the target node. Thus, changing one value can produce a transitive change in the data items.

For example, let us assume that someone orders an item online and money gets deducted from their account twice for the same item by mistake. To notify this, one might inform the concerned person and they will take necessary actions. The vendor who is responsible for this will have to manually deal with this situation. So, in cases like these where actions have to be directly taken, insiders will be given permissions and would be trusted. This is an example of insiders directly accessing data items.

A simple example to understand an indirect access would be to consider three attributes namely *Date of Birth (DOB), Age* and *Vote. Vote* denotes if the person has the right for voting (varies depending on the country). When there is a change in the *DOB,* it gets reflected in the *age* as well as the vote column. This is an indirect change, i.e. change in one data item produces a change in the other one too. This is a simple example and in cases like this, the insider may be prohibited from modifying *DOB.* However, there may be complex situations that may not make the dependencies so obvious and insiders can take advantage of that.

3.1 Working of the Models

With these classifications in mind, our attack prevention system has been modeled to forbid malicious writes specifically in two scenarios.

– Prevent malicious writes in the presence of log.
– Prevent malicious writes in the absence of log (dependency graph).

A new term called 'threshold' is introduced in the paper which sets the limit to which a data item can be modified. Every data item in the database is associated with a 'threshold'. When a change crosses the threshold, it signals a threat and a check is made to verify the validity of the write operation. There might be cases where one would need more information about the write operation in order to proceed with the investigation. So, to get a clear picture of the write operation, the log is examined for determining the complete set of data items that were accessed during that entire transaction. By doing so, one can pull out the corrupt data items and can clean (recover) them from damages.

Tracing the entire log for identifying a malicious update would cause more delay when there are numerous transactions getting recorded. So, an alternate approach which works faster by employing 'dependency graphs' is proposed as the next model in the paper. Here, the process begins by developing a graph called the 'dependency' graph, whose nodes are various data objects of the database. An edge between two nodes means that there is a dependency relationship between the two data items (explained in next section). This means that the write operation performed on one data item might affect the other in some way. Our system works by monitoring every write operation on nodes and when the changes go beyond the threshold, necessary security checks are performed. The main difference between log and the graph is that, log records each activity of every transaction in a database and it is tedious to distinguish a particular sequence of interest; whereas, the graphs get dynamically built when the data items are accessed and it is easy to go back and trace which data item was accessed before or after a particular one. In the following section each term introduced in the paper is defined with suitable examples.

3.2 Definitions

Definition 1. *In a database, the data items which are very sensitive and whose changes are to be highly monitored are addressed as Critical Data Items (CDI).* For example, in a healthcare database, the report of a patient is a CDI. In a university database the grade of a student is a CDI. Malicious changes to these data items will cause a major damage to the database, and so they have to be monitored with high security.

Definition 2. *A Regular Data Item (RDI) in a database is a non-sensitive data item and changes to such data items do not cause an immediate problem.* Few data items like address, gender etc. in a company or university database may not matter much to the organization as compared to the CDI's. So changes to these might affect the database in little amounts without any serious damage.

Definition 3. *A dependency between two data items x and y (denoted as $x \rightarrow y$) can be defined as a relationship such that, y is a function of x; i.e., y is calculated using the value of x.* For example, in a student database, if the data item *grade* is calculated as *score1+score2,* then changing either of the values produces a change in *grade.* This means that a dependency relationship exists between*grade* and *score1, score2.*

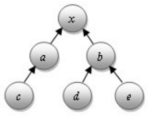


Fig. 1. An Example Dependency Graph

The figure above is an example of how a dependency graph would look like. The dependency graph is defined as V, E where the vertices V are various data items and edges E between two vertices convey that a dependency relationship exists between them.

In fig. 1, the data item x is dependent on two data items a and b. Similarly a and b are dependent on c and d, e respectively. The dependency graph is built when a transaction modifies a data item. Before the changes get committed, appropriate security checks are performed to verify the validity of write operations

Table 1. A University Database

NAME	ATTENDANCE	ASSIGNMENT	GRADE
Alisa	90%	98%	A
Jack	92%	88%	A
Jim	80%	85%	B

and also to ensure the security of data items. If they are valid, then execution of such operations is allowed.

To understand how the dependency graph is built, let us consider a scenario where the *pay* of an employee is calculated by multiplying the *number of hours* he worked per week with *pay* per hour. When his *number of hours* is entered into the database every week, a node corresponding to it gets added to the graph. At the end of the month when his *pay* is calculated from the *number of hours* and *pay* per hour, both the nodes are appended to the graph. The graph simply shows that *pay* is dependent on the both the nodes and if some threat is encountered with *pay* the other two nodes also have to be investigated. Here is an example that clarifies the concept of dependency better. A University database which maintains the academic information of a student may contain fields like the ones listed above. Here the *grade* of a student depends both on his *attendance* as well as his *assignments.* He gets an A grade when his *assignments* are between 89% and 100%. Also, an extra credit is given for the *attendance* in such a way that if his grade is along the border of A (88% is a B) from the *assignments* and he holds more than 90% in *attendance,* then a credit gets added and he gets an A. This simply means that both *attendance* and *assignment* play an important role in the value of *grade.* So, *attendance* and *assignment* are two data items which are the dependencies of *grade.*

A CDI can be dependent on another CDI or an RDI. So changing an RDI can also affect a CDI which is the main concern of our paper. As long as these dependencies exist, it will be easier for insiders to attack a CDI by working on RDI's. In spite of preventing insiders from accessing CDI's without authorization, they can alter other data items and achieve their goal.

Definition 4. *The term Δ_i denotes the amount of change (in numerals) that is made to a data item 'i' by a write operation.* To understand it better, let us take the bank balance as a data item. When we deposit, withdraw or transfer money, the value of the balance changes. The difference between the initial value and the value after the change (new value) is defined as $\Delta_{balance}$.

Definition 5. *Every data item 'i' in the dependency graph is associated with a Threshold denoted as T_i. It can be defined as a numerical value that sets the limit to which a data item could be modified without causing a threat.* Threshold may take different kinds of values depending on the data item to which it is related. If T_i=0, changes to i are highly critical to the database and if $T_i=\infty$, the changes are trivial. Thus, every data item acquires a threshold value that ranges between 0 and ∞. These values can be changed by security personnel as and when required.

Here are few numerical examples:-

1. For a company database, *salary* of employees is very sensitive and the threshold takes a value in such a way that, any change to the *salary* till it crosses the threshold is trivial. For example, a + or - $10 change to the *salary* may not make much harm whereas when one figures more than $1000 change it has to be investigated. So, depending upon the risk the company is willing to take, in the described situation, the threshold value can be set between $10 and $1000.
2. Let us consider a database which stores the *SSN* of people. It is highly sensitive and the threshold for *SSN* will monitor the number of times it gets altered. So, in this case threshold is not a mere number, it also tracks the changes.

A Non-numerical Example

Threshold for data items that are not numeric will have to be defined in a different manner than the numeric ones. It is easy to note the changes in numbers, but when the data items have values that are in alphabets, it is inefficient to monitor every single letter for a change. In those cases, initially a predefined set contains all the values a data item could take. Then a Δ value is associated for changes from one value to the other in the set. If they cross the threshold then it is considered as a violation of security. Here is an example to make it clearer.

Table 2. A Company Database

NAME	JOB TITLE	YEARS OF EXP.	SALARY
Tom	Associate Prof.	3	20k
Kelly	Professor	10	50k
John	Teaching Assistant	5	9k

Consider a company database as shown in Table 2. An insider might target the *job title* and could try to change the designation. The threshold for *job title* is complicated and a numeric threshold cannot help catching the malicious change. So, for data items like this which are non numeric, a predefined set is employed which holds the possible values the data item could take. In this case, *job title* would have a set with values such as Professor, Associate professor, Assistant Professor, Office Staff, Teaching Assistant. A shift from one position to another takes a unique Δ value which helps us in identifying threats. An immediate change from Teaching Assistant to Associate Professor is not acceptable; same way one cannot become a Professor from being an Office Staff. Thus, for each combination a Δ value is attached accordingly. For example, for the position of a Teaching Assistant, the change could be defined as (Assistant Professor, Associate Professor, Professor) in that sequence. If the Δ is made as 1 for a shift to the next position, then anything greater than 1 is considered to be invalid in this example. So the threshold for *job title* can have the condition as Δ greater than 1 and any changes that have Δ more than 1 will go for verification.

Listing out all the possibilities a data item would take is little tedious. This information could be taken from the domain defined for every data item while building the database. This method helps in identifying threats associated with any type of data item, especially that are non-numeric. The set is dynamic and have to be updated whenever the domain of a data item changes.

In practical terms, threshold can be defined for an entire table, for an attribute or for every data item. As mentioned, threshold value 0 signifies that the data item is highly sensitive and every change made to it has to be monitored. For example, let us consider *SSN* again. It very rarely gets changed and so even a one-time change has to be validated, thus it takes a threshold of 0. Similarly a data item with threshold infinity denotes that it can take any number of changes and the changes do not constitute a risk. Assigning proper threshold to the data items is a very important task. The selection of threshold should be judicious enough and it must cover all the possible scenarios by which a data item could be attacked. It should also be dynamic so that if the risk level of the data item increases or decreases, our threshold must also change accordingly. In this paper, for simplicity, we have focused on threshold values that operate on numeric data items.

Definition 6. *Every time a transaction operates on a data item 'i' and makes a change, the Δ value gets collected in a variable called 'unverified Δ' denoted as Δ_u.*

If the changes are below threshold, they get added to the existing values in the variable and the point at which Δ_u exceeds T_i it signifies a threat. A value of 0 to Δ_u denotes that the data item is secure and there are no unchecked write operations. Initially when the database is built, the Δ_u values of all the data items are initialized to 0. Δ_{ui} refers to the unverified Δ value for the data item 'i'. Let us consider the balance attribute of a bank database. Initially let's assume that the Δ_u has a value 100. When someone makes a deposit or withdrawal of 50 dollars, it becomes 150 or 50, respectively. Same is the case for any other data item; the amount of change made to the data item is recorded and summed up in the variable. The main goal of the proposed idea is to make $\Delta_u = 0$ most of the times so that, the data items remain secured.

Irrespective of the sensitivity, the changes made to every single data item in a database is important to the organization. But, it is time consuming to check all the transactions to verify each and every write operation. Thus, the most important data items, the CDIs are alone investigated every time it changes. We do not claim that the changes of an RDI are immaterial, but checking each change to each RDI will slow down system performance significantly. So, our priority is to safeguard the critical data items first, and then periodically track the RDI's to assure they are free from threats.

When security checks are made, it might require manual verification of transactions. For example, if there is a vast change in a bank account, then during the check one needs to manually affirm that the change was valid. This checking is real time and might involve some delay, but eventually it satisfies our goal of securing data items. When security is a top priority, delays are small prices one must be willing to pay. The two models introduced in the paper are explained in the following sections.

4 Log Based Model

A log file keeps record of each operation and the data items accessed during every transaction that gets executed. Although traditional logs do not store read operations, in our model we require them to be stored. Furthermore, we also require that, the type as well as the amount of changes ($\varDelta_u$) made to each data item during a transaction must also be recorded. With all these information in hand, one can trace the log to figure the dependencies and the sequence of data items involved during a transaction.

4.1 Identifying Threats

According to our proposed idea, when a write operation does something that goes beyond the threshold, the system sends an alert. As stated earlier, if a critical data item is getting modified then the corresponding write operation is immediately monitored. There might be cases where the current operation modifying a data item is determined to be correct during investigation where as one or more of the past operations that contributed to the present value may involve malicious activities. Hence, all past operations have to be investigated as well. According to the proposed model, there are two scenarios during which a log would be examined.

– Changes exceeding threshold.
– Sum of values in $\varDelta_u$ exceeding threshold.

Whenever the write operation makes a change that exceeds the threshold, the transaction is verified and if it requires more information the log is examined. Same is the case when the sum of values in $\varDelta_u$ exceeds the threshold. Below is an example.

Table 3 shows how a modified log used in this model would look like. It contains the data items getting involved in a transaction and the type of action performed on each data item by a transaction. If the data items are written, the ways in which they get modified are also recorded. The 'value' field stores the unverified changes for every item and it gets updated once they are verified. In the figure below, initially, a transaction t1 performed a write operation on data item b which is recorded as b^2, after which the value of b was read by t2. Following this, t2 updated the value of a by adding b to it. Then a transaction t3 read the updated value of a and modified x as $x+$ a^3. Now, if the changes made to x cross the threshold, verification is triggered. If the write made by t3 appears to be wise, then other data items which contributed to the value of x might be wrong. By tracing the log, it is easy to figure out which data items were involved in the modification made to x. The $\varDelta_u$ value for every data item shows the amount of changes that are still unverified and are less than the threshold. By validating those changes for a and b, one could figure out which update was wrong. This shows the strength of dependencies. Even though the changes made to a and b did not cross their threshold, they indirectly affected x which turned out to be malicious. Once the checking is done, the Δ_u values of all the data items involved could be set to 0, which states there are no pending changes to be verified and the data items are out of threats so that the forthcoming transactions can avoid checking those items.

Table 3. Modified Log

Data item	b	a	x
Actions	written by t_1 (b^2)		
	read by t_2	written by t_2 $(a+b)$	
		read by t_3	written by t_3 $(x+a^3)$
Value	Δ_{ub}	Δ_{ua}	Δ_{ux}

The log modeled in the paper avoids all aborted transactions and works on the committed ones. The corresponding algorithm is provided below.

Algorithm 1. Preventing malicious writes with Logs

```
Input: $\Delta_i$, $\Delta_{ui}$, Data item $i$, Threshold $T_i$
Output: Allow only valid write transactions

1. For each write operation on a data item $i$
2. $\Delta_{ui}=\Delta_{ui}+\Delta_i$
3. If $i$ a CDI
4.     Check the validity of the write operation
5. Else
6.   If $\Delta_{ui}>T_i$,
7.     If $\Delta_i$ valid
8.       For the set of data items $\{j\}$ that affected $i$
9.         If $j$ is empty
10.           Set $\Delta_u$ of parent and children to zero
11.        If $\Delta_{uj}=0$ then
12.           Remove $j$; goto step 8
13.        Else assign $j$ as the parent node($i$); goto step 7
14.   Else Check the current transaction
15.      Assign $\Delta_{ui}=0$
16.   End for
```

Comments: Step 7 checks if the current write operation is valid or not, Step 8 reads the log and pulls out all the read dependencies into a subset $\{j\}$ where j denotes a dependent data item. Step 13 starts a recursive search by assigning the child node as the parent node. Thus, our algorithm tries to find which transaction is the reason for a malicious activity. This method introduces some delay since one has to back track a substantial portion of the log and find the right transaction which was malicious. To overcome this delay, we have introduced the dependency graph model which is discussed in the following section.

5 Dependency Graph Based Model

Generally in all the databases, numerous dependencies will exist between data items, which continue to be an advantage for insiders. Insiders who are not authorized to access the critical data items tend to use dependencies as a means of attacking CDI's. They try to guess the dependency between regular data items and the critical data items, as a result of which they can successfully modify the CDI's by modifying the RDI's.

Initially, when the database is built, numerous dependencies will be created among data items. Based on those, we build our dependency graph immediately after a transaction starts its operations. The dependency graph has the data items getting accessed as nodes and their relationship with other items contributing to the edges.

Every time a transaction generates a write operation, the Δ value gets added to its Δ_u. Then, depending on the type of the data item under modification, certain actions are performed to ensure its security. There are three cases:

– Data item is a CDI.
– Data item is an RDI.
– Data item is a CDI or an RDI, affecting a CDI immediately.

If the data item getting modified is a CDI, then irrespective of its Δ, security checks are made. This is to ensure that the CDI's in a database are always free from malicious activities. Information such as SSN, medical reports etc. are highly confidential and have to be under supervision continuously. Next, when an RDI is modified, if the changes are less than the threshold they get accumulated in Δ_u. If they cross the threshold then security checks are triggered. If the RDI or CDI getting modified is found to impact another CDI then it is immediately checked, since the goal of our system is to prioritize the security of CDIs. This is the basic idea of the dependency graph model. In all these cases, dependency graphs are built but the process depends on few other conditions as listed:

– Write operation on a data item immediately affecting the dependents.
– Write operation on a data item subsequently affecting the dependents.

The first scenario addresses the write operations which modify a data item, which in turn automatically changes few other data items. As a consequence, all the data items getting involved in the change are added as nodes to the graph with corresponding edges denoting the manner in which the data items affect each other. This signifies that there is a dependency between them. The *DOB, age* and *vote* example discussed previously falls under this category. Firstly let's assume vote is a CDI. When *DOB* is changed, all the three nodes are added to the graph since they get affected automatically. As the CDI *vote* is getting changed here, the write operation on *DOB* is immediately verified. Now let us imagine that *vote* is an RDI. When *DOB* is changed, the graph adds all the three nodes, but since there is no CDI getting changed here, the operation is continued without any checks. The transaction is checked only when any data item among them has a change that goes beyond its threshold. Also, if *vote* remains the same for a change in *DOB,* then only the first two nodes are added to the graph. Thus, only the transactions that have happened in the database will be recorded in the graph.

The next scenario deals with write operations on data items which may affect other items, but not immediately. Since the dependents are not altered immediately, only the node currently getting modified is added to the graph. To make it clearer, let us consider that the *grade* of a student depends on the scores of four *tests.* The *grade* will not be calculated until all the four *test scores* are received. So, even though *grade* is dependent on various data items, a change will be done to it only when all the *test scores* have received a value. So here, every time a value is entered for a *test score,* only that corresponding node gets added to the graph. Finally while calculating *grade,* a *grade* node will be appended to the graph and if there are any issues encountered, then the graph is traced to find which *test score* carries a wrong value. The entire idea is that, when a data item is found to be invalid, then all its dependents are also checked and cleared. This will leave the data items in the database free from malicious activities. The proposed idea is explained in terms of an algorithm below.

Algorithm 2. Preventing malicious writes with Dependency graph

```
Input: $\Delta_i$, $\Delta_{ui}$, Data item $i$, Threshold $T_i$
Output: Allow only valid transactions
1. For each write operation on a data item $i$
2. $\Delta_{ui} = \Delta_{ui} + \Delta_i$
3. If $i$ a CDI OR any CDI dependent on $i$ OR $\Delta_i > T_i$,
4.     Check the validity of the write operation
5.        If valid
6.            Check the dependents of $i$
7.     Set $\Delta_u$ of data items in the sequence to 0
8.  Else proceed with the transaction
9.  End for
```

Comments. Step 4 checks the validity of the current write operation. If valid, then Step 6 figures out the dependencies from the graph built and checks them also. In step 7, Δ_u values of the checked data items are re-initialized to 0 which means they are perfect.

Anytime a check is triggered for a data item, it terminates by making its $\Delta_u = 0$. This ensures that the data items are secure. A transaction accesses its own set of data items, so every transaction starts building its own dependency graph. The dependency graphs may be disjoint for each transaction since each might have different sequence and the data items may not be related. But once a dependency is found between two data items in two different graphs, the graphs will then be merged by delineating an edge between the nodes.

6 Comparison of the Two Models

As stated earlier, both the models aim at preventing a database from insider threats. But there are few factors which make one better than the other at times. Some of them are speed, complexity, accuracy and efficiency.

In terms of speed, tracing the log to find a malicious transaction is time consuming than figuring it out from the dependency graph. Dependency graphs simplify the work done by a log, by automatically building the transaction sequence; whereas, the log records numerous operations every day and it is practically hard to check every data item to pull out the invalid ones. But, though log method puts in delay, it maintains all the information we need. So, one can be confident about catching the malicious operations by looking through the log.

Adding to speed, dependency graphs are less complex than logs in figuring out threats, but are more difficult to build when the database is huge and has numerous dependencies. Both the log and graphs are accurate, but logs are trusted more because, the entries in a log cannot be changed or hacked. Certain organizations might refrain from building the graphs since it requires more effort, but it is certainly an improvement over the logs in all aspects. To compare the efficiency of the models, we intend to develop a simulation of both the models. Since the ultimate goal of mitigating insider threat is achieved by both of them, they are efficient in their own terms.

7 Conclusions and Future Work

Insider threat is a problem that is standing for long and it sometimes becomes unpreventable due to the lack of proper tools and mechanisms. In addition to this, attacking a database and evading detection has become easy for an insider who successfully discovers the dependencies among data items. So establishing appropriate techniques for catching insiders is essential in enhancing and creating a secured database system. Thus, in this paper we discussed various possibilities through which an insider can perform malicious writes. We introduced an idea of using threshold to limit the amount of change an operation could perform without requiring a validation. We also proposed two different attack prevention models that can detect and prevent malicious write operations in various scenarios. For systems which don't create dependency graphs, we explained how logs could be a counterpart. The two models were also compared and analyzed in terms of various factors. Various works exist to prevent malicious read operations (unauthorized disclosure of information) in databases. To complement those models, an attack prevention model concentrating mainly on write operations was proposed. As a future work, we plan to carry out experiments to prove the effectiveness of the models for both numeric and non-numeric data.

Acknowledgement. This work has been supported in part by US AFOSR under grant FA 9550-08-1-0255. We are thankful to Dr. Robert. L. Herklotz for his support, which made this work possible.

References

1. Schultz, E.E.: A framework for understanding and predicting insider attacks. Computers & Security 21(6), 526–531 (2002)
2. Predd, J., Pfleeger, S.L., Hunker, J., Bulford, C.: Insiders Behaving Badly. IEEE Security & Privacy 6(4), 66–70 (2008)
3. Bishop, M., Gates, C.: Defining the Insider Threat. In: Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research, Tennessee, vol. 288 (2008)
4. Brackney, R., Anderson, R.: Understanding the insider threat. In: Proceedings of a March 2004 workshop. Technical report, RAND Corporation. Santa Monica, CA (2004)
5. Spitzner, L.: Honeypots: Catching the Insider Threat. In: Proceedings of the 19th Annual Computer Security Applications Conference, Washington (2003)
6. Ray, I., Poolsapassit, N.: Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 231–246. Springer, Heidelberg (2005)
7. Franqueira, V., van Eck, P.: Defense against Insider Threat: A Framework for Gathering Goal-based Requirements. In: Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2007), Trondheim, Norway (June 2007)
8. Yaseen, Q., Panda, B.: Knowledge Acquisition and Insider Threat Prediction in Relational Database Systems. In: Proceedings of the International Workshop on Software Security Processes, Vancouver, Canada, pp. 450–455 (2009)
9. Althebyan, Q., Panda, B.: A knowledge-base model for insider threat prediction. In: Proceedings of the IEEE Workshop on Information Assurance and Security, West Point, NY, pp. 239–246 (2007)
10. Chinchani, R., Iyer, A., Ngo, H.Q., Upadhyaya, S.: Towards a Theory of Insider Threat Assessment. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), June 28-July 1, pp. 108–117 (2005)
11. Farkas, C., Jajodia, S.: The Inference Problem: A Survey. ACM SIGKDD Explorations 4, 6–11 (2002)
12. Farkas, C., Toland, T., Eastman, C.: The Inference Problem and Updates in Relational Databases. In: Proceedings of the 15th IFIP WG11.3 Working Conference on Database and Application Security, pp. 181–194 (2001)
13. Brodsky, A., Farkas, C., Jajodia, S.: Secure Databases: Constraints, Inference Channels and Monitoring Disclosures. In: Proceedings of the IEEE Trans. on Knowledge and Data Engineering, vol. 12, pp. 900–919 (2000)
14. Yip, R., Levitt, K.: Data Level Inference Detection in Database Systems. In: Proceedings of the 11th Computer Security Foundations Workshop, Rockport, MA, pp. 179–189 (1998)

15. Maybury, M., Chase, P., Cheikes, B., Brackney, D., Matznera, S., Hetherington, T., Wood, B., Sibley, C., Marin, J., Longstaff, T.: Analysis and Detection of Malicious Insiders. In: Proceedings of the International Conference on Intelligence Analysis, VA (2005)
16. Bradford, P., Hu, N.: A Layered Approach to Insider Threat Detection and Proactive forensics. In: Proceedings of the Twenty-First Annual Computer Security Applications Conference, Tucson, AZ (December 2005)
17. Morgenstern, M.: Security and Inference in Multilevel Database and Knowledge-Base Systems. In: ACM SIGMOD Record, NewYork, USA, pp. 357–373 (1987)
18. Mathew, S., Upadhyaya, S., Ha, D., Ngo, H.Q.: Insider abuse comprehension through capability acquisition graphs. In: Proceedings of 11th IEEE International Conference on Information Fusion, pp. 1–8 (2008)

A Semantic Hierarchy for Erasure Policies

Filippo Del Tedesco[1], Sebastian Hunt[2], and David Sands[1]

[1] Chalmers University of Technology, Sweden
[2] City University London

Abstract. We consider the problem of logical data erasure, contrasting with physical erasure in the same way that end-to-end information flow control contrasts with access control. We present a semantic hierarchy for erasure policies, using a possibilistic knowledge-based semantics to define policy satisfaction such that there is an intuitively clear upper bound on what information an erasure policy permits to be retained. Our hierarchy allows a rich class of erasure policies to be expressed, taking account of the power of the attacker, how much information may be retained, and under what conditions it may be retained. While our main aim is to specify erasure policies, the semantic framework allows quite general information-flow policies to be formulated for a variety of semantic notions of secrecy.

1 Introduction

Erasing data can be difficult for many reasons. As an example, recent research on SSD-drives has shown that the low-level routines for erasing data often inadvertently leave data behind [30]. This is due to the fact that information on an SSD (in contrast to a more conventional magnetic hard drive) gets copied to various parts of memory in order to even out wear. The naive firmware sanitisation routines do not have access to the movement-history of data, and so leave potentially large amounts of data behind.

This paper is not focused on low-level erasure *per se*. The requirement that data is used but not retained is commonplace in many non hardware-specific scenarios. As an everyday example consider the credit card details provided by a user to a payment system. The expectation is that card details will be used to authorise payment, but will not be retained by the system once the transaction is complete.

An erasure policy describes such a limited use of a piece of data. But what does it mean for a system to correctly erase some piece of data? One natural approach taken here is to view erasure as an information-flow concept – following [7]. To erase something means that after the point of erasure there is no information flowing from the original data to observers of the system. This gives a natural generalisation of the low-level concept of *physical erasure* to what one might call *logical erasure*. Logical erasure specifies that a system behaves *as if* it has physically erased some data *from the viewpoint of a particular observer*. The observer viewpoint is more than just a way to model erasure in a multi-level security context (as in [7]). To understand the importance of the attacker viewpoint, consider a system which receives some data subject to an erasure policy. The system then receives a random key from a one-time pad and XORs it with the secret. The key is then overwritten with a constant. Does such a system erase the data? The answer, from an information-flow perspective, depends on what the observer (a.k.a. *the attacker*) can see/remember about the execution. An attacker who can see the exact final state of the system (including the encrypted data) and nothing more, cannot deduce anything about the subject data, and so we can conclude that it is erased for that attacker. But if the attacker could also observe the key that was provided, then the system is not erasing. Different situations may need to model different attacker powers.

In practice the concept of erasure is a subtle one in which many dimensions play a role. This is analogous to the various "dimensions" of declassification [28]. In this paper we develop a semantic model for erasure which can account for different *amounts* of erasure, covering the situation where some but not necessarily all information about the subject is removed, and different varieties of *conditional erasure*, which describes both what is erased, and under what conditions.

The contribution of this work is to identify (Section 2) and formalise (Section 4) a hierarchy of increasingly expressive erasure policies which captures various dimensions of erasure. To do this we build on a new possibilistic information-flow model (Section 3) which is parameterised by (i) the *subject* of the information flow policy (e.g. the data to be erased), (ii) the attacker's observational power. This is done taking into account the *facts* that an attacker might be interested to learn, and the *queries* which he can or will be able to answer about the subject.

Proofs of main results can be found in the extended version of the paper [13].

2 Erasure Case Studies

We consider a series of examples of erasing systems which differ according to the way they answer the following questions:

1. *How much* of the erasure subject is erased?
2. *Under which conditions* is erasure performed?

The examples are presented via simple imperative pseudocode. We emphasise that the examples themselves are not intended to be realistic programs – they serve to motivate simply and intuitively various types of erasure policy that we will formalise in a more abstract setting in Section 4.

2.1 Total Erasure

Consider a ticket vending machine using credit cards as the payment method. A partial implementation, in simplified form, is shown in Listing 1.1. Line 1

```
1 get(cc_number);
2 charge(ticket_cost,cc_number);
3 log(current_time());
4 cc_number=null;
```

Listing 1.1. Ticket vending machine, total and unconditional erasure

inputs the card number; line 2 executes the payment transaction; line 3 writes the transaction time to a log for audit purposes; line 4 deletes the card number.

This is an example of an erasing program: once line 4 is executed, the card number has been erased from the system. This statement can be refined further with respect to our original questions: 1) the system is *totally* erasing (no information about the card number is retained) and 2) erasure occurs *unconditionally*, since control flow always reaches line 4.

2.2 Partial Erasure

Consider a variant of the vending machine (Listing 1.2) which logs the last four digits of the card number of each transaction, enabling future confirmation of transactions in response to user queries. The difference to Listing 1.1 is in line 3, where additional data is written to the log.

```
1 get(cc_number);
2 charge(ticket_cost,cc_number);
3 log(current_time(),last4(cc_number));
4 cc_number=null;
```

Listing 1.2. Ticket vending machine, partial and unconditional erasure

With this change, line 4 no longer results in total erasure since, even after cc_number is overwritten, the last four digits of the card number are retained in the log.

2.3 Low Dependent Erasure

Consider a further elaboration of the vending machine example (Listing 1.3) which allows the user to *choose* whether the last four digits are retained. In line 3 the program acquires the user choice, then it either proceeds as Listing 1.2 or as Listing 1.1, according to the choice. Now the question about how much information is erased has two different answers, depending on the second user input. Since this dependency is not related to the erasure subject itself, we call this *low dependent* erasure.

```
get(cc_number);                                1
charge(ticket_cost,cc_number);                 2
get(choice);                                   3
if choice="Allow"                              4
 then log(current_time(),last4(cc_number));    5
 else log(current_time());                     6
cc_number=null;                                7
```

Listing 1.3. Ticket vending machine, low dependent erasure

2.4 High Dependent Erasure

Suppose there is a brand of credit card, StealthCard, which only allows terminals enforcing a strict confidentiality policy to be connected to their network. This requires a further refinement of the program (Listing 1.4), since StealthCard users are not permitted a choice for the logging option. At line 3 the credit card number is inspected and, if it is a StealthCard, the system proceeds like 1.1.

```
get(cc_number);                                  1
charge(ticket_cost ,cc_number);                  2
if (cc_number is in StealthCard)                 3
  then log(current_time());                      4
  else get(choice);                              5
    if choice="Allow"                            6
      then log(current_time(),                   7
                last4(cc_number));               8
      else log(current_time())                   9
cc_number=null;                                  10
```

Listing 1.4. Ticket vending machine, high dependent erasure

Compared to the previous case, this example has an additional layer of dependency, since the amount of data to be erased is itself dependent on the erasure subject. We refer to this as *high dependent* erasure.

3 An Abstract Model of Information Flow

We formalise erasure policies as a particular class of information flow policies. In this section we define the basic building blocks for describing such policies. We consider trace-based (possibilistic) models of system behaviour and we interpret information flow policies over these models. We make the standard conservative assumption that the attacker has perfect knowledge of the system model.

Our definitions are based directly on what an attacker can deduce about an erasure subject from observations of system behaviour. In this respect our model is close in spirit to Sutherland's multi-level security property of nondeducibilty [29]. However, we are not directly concerned with multi-level security and, in a number of ways, our model is more abstract than non-deducibility. For example, McLean's criticism [22] of nondeducibility (that it fails to incorporate an appropriate notion of causality) does not apply, since our notion of the "subject" of a policy is general enough to incorporate temporal dependency if required. On the other hand, our model *is* open to the criticism of nondeducibilty made by Wittbold and Johnson [31] with regard to interactive environment behaviours. Adapting our work using the approach of [31] (explicit modelling of user strategies) remains a subject for future work. A more radical departure from the current work, though still possibilistic, would be to take a process-calculus approach [16].

3.1 Trace Models

The behavioural "atom" in our framework is the *event* (in our examples this will typically be an input ($?v$) or output ($!v$) but internal computation steps can be modelled in the same way). Traces, ranged over by s, t, s_1, t_1, etc, are finite or countably infinite sequences of events. We write $t.e$ for the trace t extended with event e and we write $s.t$ for the concatenation of traces s and t. In what follows we assume given some set T of traces.

A *system* is considered to be a set $S \subseteq T$ (the assumption is that S is the set of maximal traces of the system being modeled). Certain parts of system behaviour will be identified as the *subject* of our policies and we define these parts by a function $\Phi : T \to D$, for some set D (typically, Φ will be a projection on traces). For a confidentiality property the subject might represent the secret that we are trying to protect (an input or a behaviour of a classified agent). For erasure the subject will be the input which is to be erased.

Given a system S, we denote by $\Phi(S)$ the subset of D relevant for S:

$$\Phi(S) = \{\Phi(t) | t \in S\}$$

We call this the *subject domain* of S. Let $\mathrm{Sys}(V)$ be the set of all systems with subject domain V. Our flow policies will be specific to systems with a given subject domain.

3.2 Equivalence Relations and Partitions

The essential component of a flow policy is a *visibility* policy which specifies how much an attacker should be allowed to learn about the subject of a system by observing its behaviour. Following a standard approach in the information flow literature – see, for example [20,27] – we use equivalence relations for this purpose. A flow policy for systems in $\mathrm{Sys}(V)$ is $R \in \mathrm{ER}(V)$, where $\mathrm{ER}(V)$ denotes the set of all equivalence relations on V. The intention is that attackers should not be able to distinguish between subjects which are equivalent according to R. An example is the "have the same last four digits" relation, specifying that the most an attacker should be allowed to learn is the last four digits of the credit card number (put another way, all cards with the same last four digits should look the same to the attacker).

In what follows we make extensive use of two key, well known facts about equivalence relations:

- The set of equivalence relations on V, ordered by inclusion of their defining sets of pairs, forms a complete lattice, with the identity relation (which we denote Id_V) as the bottom element, and the total relation (which we denote All_V) as the top.
- The set of equivalence relations on V is in one-one correspondence with the set of *partitions* of V, where each of the disjoint subsets making up a partition is an equivalence class of the corresponding equivalence relation. We write $\mathrm{PT}(V)$ for the set of all partitions of V. Given $P \in \mathrm{PT}(V)$, we write $\mathcal{E}(P)$ for the corresponding equivalence relation: $v_1 \mathcal{E}(P) v_2$ iff $\exists X \in P. v_1, v_2 \in X$. In the other direction, given $R \in \mathrm{ER}(V)$ and $v \in V$ we write $[v]_R$ for the R-equivalence class of v: $[v]_R = \{v' \in V | v' \; R \; v\}$. We write $[R]$ for the partition corresponding to R: $[R] = \{[v]_R | v \in V\}$.

In the current context, the significance of $R_1 \subseteq R_2$ is that R_1 is more discriminating - i.e., has smaller equivalence classes - than R_2. Hence, as visibility policies, R_1 is more permissive than R_2. The lattice operation of interest on $\mathrm{ER}(V)$ is *meet*, which is given by set intersection. Given a family of equivalence relations $\{R_i\}_{i \in I}$, we write their meet as $\bigwedge_{i \in I} R_i$ (the least permissive equivalence relation which is nonetheless more permissive than each R_i).

The order relation on partitions corresponding to subset inclusion on equivalence relations will be written $\preceq_{\mathrm{ER}}$, thus $[R_1] \preceq_{\mathrm{ER}} [R_2]$ iff $R_1 \subseteq R_2$. We overload the above notation for meets of partitions in this isomorphic lattice: $\bigwedge_{i \in I} P_i = [\bigwedge_{i \in I} \mathcal{E}(P_i)]$.

3.3 Attacker Models and K-Spaces

As discussed in the introduction, whether or not a system satisfies a policy will depend on what is observable to the attacker. We specify an *attacker model* as an equivalence relation on traces, $A \in \mathrm{ER}(T)$. Note that this is a passive notion of attacker - attackers can observe but not interact with the system.

To compare what the attacker actually learns about the subject with what the visibility policy permits, we define, for each attacker observation $O \in [A]$, the corresponding *knowledge set* $\mathrm{K}_S(O) \subseteq V$, which is the set of possible subject values which the attacker can deduce from making a given observation[1]: $\mathrm{K}_S(O) = \{\Phi(t) | t \in O \cap S\}$.

The *K-space of A for S*, denoted $\mathcal{K}_S(A)$, is the collection of all the attacker's possible (ie non-empty) knowledge sets when observing S:

$$\mathcal{K}_S(A) = \{\mathrm{K}_S(O) | O \in [A], O \cap S \neq \emptyset\}$$

Lemma 1. *Let $S \in Sys(V)$ and $A \in ER(V)$. Then the K-space of A for S* covers *V, by which we mean that every member of $\mathcal{K}_S(A)$ is non-empty and $\bigcup \mathcal{K}_S(A) = V$.*

From now on, for a given V, we use the term K-space to mean any collection of sets which covers V.

In the special case that a system's behaviour is a *function* of the subject, each K-space will actually define an equivalence relation on V:

Proposition 1. *Say that $S \in Sys(V)$ is* functional *just when, for all $t, t' \in S$, $t \neq t' \Rightarrow \Phi(t) \neq \Phi(t')$. In this case, for all $A \in ER(T)$, $\mathcal{K}_S(A)$ partitions V.*

When S is functional, the K-space $\mathcal{K}_S(A)$, being a partition, can be interpreted as the equivalence relation $\mathcal{E}(\mathcal{K}_S(A))$. So, in the functional case there is a straightforward way to compare a visibility policy with an attacker's K-space: we say that the policy R is satisfied just when R is more discriminating than this induced equivalence relation. Formally, when S is functional, *S satisfies R for attacker A*, written $S \vdash_A R$, just when $R \subseteq \mathcal{E}(\mathcal{K}_S(A))$ or, equivalently:

$$S \vdash_A R \text{ iff } [R] \preceq_{\mathrm{ER}} \mathcal{K}_S(A) \tag{1}$$

We now consider how to extend this definition to the general case, in which a system has other inputs apart from the policy subject.

[1] A more reasonable but less conventional terminology would be to call this an *uncertainty set*.

3.4 Comparing K-Spaces: Facts and Queries

In general, a system's behaviour may depend on events which are neither part of the policy subject nor visible to the attacker. In this case, the attacker's knowledge of the subject need not be deterministic, resulting in a K-space which is not a partition. This raises the question: when is one K-space "more discriminating" than another?

Here we motivate a variety of orderings by considering some basic modes in which an attacker can use observations to make deductions about the subject of a system:

Facts. A fact F is just a set of values. A given knowledge set X *confirms fact* F just when $X \subseteq F$. Dually, X *has uncertainty* F when $F \subseteq X$. For example a fact of interest (to an attacker) might be the set of "Platinum" card numbers. In this case an observation might confirm to the attacker that a card is a Platinum card by also revealing exactly which platinum card it is. For a given K-space K we then say that
- K *can confirm* F if there exists some $X \in K$ such that X confirms F.
- K *can have uncertainty* F if there exists some $X \in K$ such that X has uncertainty F.

Queries. A query Q is also just a set of values. We say that a given knowledge set X *answers query* Q just when either $X \subseteq Q$ or $X \subseteq V \setminus Q$. For a given K-space K we then say that
- K *will answer* Q if for all $X \in K$, X answers Q, and
- K *can answer* Q if there exists some $X \in K$ such that X answers Q.

In a possibilistic setting it is natural to focus on those "secrets" which it is *impossible* for a given system to reveal, where revealing a secret could mean either confirming a fact or answering a query. Two of the four K-space properties defined above have an immediate significance for this notion of secrecy:

- Say that S *keeps fact* F *secret from attacker* A iff there are no runs of S for which A's observation confirms F, i.e., iff: $\neg(\mathcal{K}_S(A)$ can confirm $F)$.
- Say that S *keeps query* Q *secret from attacker* A iff there are no runs of S for which A's observation answers Q, i.e., iff: $\neg(\mathcal{K}_S(A)$ can answer $Q)$.

The possibilistic secrecy significance of "has uncertainty" and "will answer" is not so clear. However, as we will show, we are able to define flow policies and a parameterized notion of policy satisfaction which behaves well with respect to all four properties.

Using the ability of a K-space to confirm facts and answer queries, we can order systems in different ways, where a "smaller" K-space (ie one lower down in the ordering) allows the attacker to make more deductions (and so the system may be regarded as less secure). Define the following orderings between K-spaces:

Upper: $K_1 \preceq_{\mathrm{U}} K_2$ iff $\forall F. K_2$ can confirm $F \Rightarrow K_1$ can confirm F. Note that $K_1 \preceq_{\mathrm{U}} K_2$ iff K_2 keeps more facts secret than K_1.

Lower: $K_1 \preceq_{\mathrm{L}} K_2$ iff $\forall F. K_1$ can have uncertainty $F \Rightarrow K_2$ can have uncertainty F.

Convex (Egli-Milner): $K_1 \preceq_{\mathrm{EM}} K_2$ iff $K_1 \preceq_{\mathrm{U}} K_2 \wedge K_1 \preceq_{\mathrm{L}} K_2$.
Can-Answer: $K_1 \preceq_{\mathrm{CA}} K_2$ iff $\forall Q. K_2$ can answer $Q \Rightarrow K_1$ can answer Q. Note that $K_1 \preceq_{\mathrm{CA}} K_2$ iff K_2 keeps more queries secret than K_1.
Will-Answer: $K_1 \preceq_{\mathrm{WA}} K_2$ iff $\forall Q. K_2$ will answer $Q \Rightarrow K_1$ will answer Q.

It is straightforward to verify that these orders are reflexive and transitive, but not anti-symmetric. The choice of names for the upper and lower orders is due to their correspondence with the powerdomain orderings [25]:

Proposition 2

$$K_1 \preceq_U K_2 \textit{ iff } \forall X_2 \in K_2. \exists X_1 \in K_1. X_1 \subseteq X_2$$
$$K_1 \preceq_L K_2 \textit{ iff } \forall X_1 \in K_1. \exists X_2 \in K_2. X_1 \subseteq X_2$$

We can compare the K-space orders 1) unconditionally, 2) as in the case of policy satisfaction, when we are comparing a partition with a K-space, and, 3) when the K-spaces are both partitions, yielding the following results:

Proposition 3. *1.* $\preceq_{EM} \subsetneq \preceq_L \subsetneq \preceq_{WA}$ *and* $\preceq_{EM} \subsetneq \preceq_U \subsetneq \preceq_{CA}$.
2. Additionally, when P is a partition: $P \preceq_{CA} K \Rightarrow P \preceq_{WA} K$ (the reverse implication does not hold in general).
3. $\preceq_{ER}$, $\preceq_{EM}$, $\preceq_L$, and $\preceq_{WA}$ all coincide on partitions. Furthermore, when P_1 and P_2 are partitions: $P_1 \preceq_{ER} P_2 \Rightarrow P_1 \preceq_U P_2 \Rightarrow P_1 \preceq_{CA} P_2$ (the reverse implications do not hold in general).

These orderings give us a variety of ways to extend the definition of policy satisfaction from functional systems (Equation 1) to the general case. The choice will depend on the type of security condition (eg protection of facts versus protection of queries) which we wish to impose.

4 The Policy Hierarchy

We specify a three-level hierarchy of erasure policy types. All three types of policy use a structured collection of equivalence relations on the subject domain to define what information should be erased. A key design principle is that, whenever a policy permits part of the erasure subject to be retained, this should be *explicit*, by which we mean that it should be captured by the conjunction of the component equivalence relations.

For each type of policy, we define a satisfaction relation, parameterized by a choice of K-space ordering $o \in \{U, L, EM, CA, WA\}$.

Assume a fixed policy subject function $\Phi : T \to D$. Given a subset $V \subseteq D$, let $T_V = \{t \in T | \Phi(t) \in V\}$. Note that if S belongs to $\mathrm{Sys}(V)$ then $S \subseteq T_V$.

Type 0 Policies

Type 0 policies allow us to specify *unconditional* erasure, corresponding to the two examples shown in Section 2 in Listings 1.1 and 1.2.

A Type 0 erasure policy is just a visibility policy. We write Type-0(V) for the set of all Type 0 policies for systems in Sys(V) (thus Type-0(V) = ER(V)). The definition of satisfaction for a given attacker model A and system S uses a K-space ordering (specified by parameter o) to generalise the satisfaction relation of Equation 1 to arbitrary (i.e., not-necessarily functional) systems:

$$S \vdash_A^o R \text{ iff } [R] \preceq_{\rm o} \mathcal{K}_S(A)$$

For functional systems note that, by Proposition 3, choosing o to be any one of EM, L or WA yields a notion of satisfaction equivalent to Equation 1, while U and CA yield strictly weaker notions.

Example. Consider the example in Listing 1.2. The subject domain is CC, the set of all credit card numbers, and (since the erasure subject is the initial input) the subject function is the first projection on traces. The policy we have in mind for this system is that it should erase all but the last four digits of the credit card number. We extend this example so that it uses a method call `erased()` to generate an explicit output event η (signalling that erasure should have taken place) followed by a dump of the program memory (thus revealing all retained information to a sufficiently strong attacker).

```
get(cc_number);
charge(ticket_cost ,cc_number);
log(current_time(),last4(cc_number));
cc_number=null;
erased();
dump();
```

Listing 1.5. Ticket vending machine, partial and unconditional erasure: extended

If we restrict attention to systems (such as this one) where each run starts by inputting a credit card number and eventually outputs the erasure signal exactly once, we can assume a universe of traces T such that all $t \in T$ have the form $t = ?cc.s.\eta.s'$, where s, s' are sequences not including η. Let S be the trace model for the above system. The required visibility policy is the equivalence relation L4 $\in$ ER(CC) which equates any two credit card numbers sharing the same last four digits. An appropriate attacker model is the attacker who sees nothing before the erasure event and everything afterwards. Call this the *simple erasure attacker*, denoted AS:

$$\text{AS} = \{(t_1, t_2) \in T \times T | \exists s_1, s_2, s_3.\ t_1 = s_1.\eta.s_3 \ \wedge\ t_2 = s_2.\eta.s_3\}$$

Informally, it should be clear that, for each run of the system, AS will learn the last four digits of the credit card which was input, together with some other log data (the transaction time) which is independent of the card number. Thus the knowledge set on a run, for example, where the card number ends 7016, would be the set of all card numbers ending 7016. The K-space in this example will actually be exactly the partition [L4], hence S does indeed satisfy the specified policy: $S \vdash_{\rm AS}^o$ L4 for all choices of o. From now on, we write just $S \vdash_A R$ to mean that it holds for all choices of ordering (or, equivalently, we can consider $\vdash_A$ to be shorthand for $\vdash_A^{EM}$, since EM is the strongest ordering).

Type 1 Policies

Type 1 policies allow us to specify "low dependent" erasure (Section 2, Listing 1.3), where different amounts may be erased on different runs, but where the erasure condition is independent of the erasure subject itself.

For systems in $\mathrm{Sys}(V)$ the erasure condition is specified as a partition $P \in \mathrm{PT}(T_V)$. This is paired with a function $f : P \rightarrow \text{Type-0}(V)$, which associates a Type 0 policy with each element of the partition. Since the domain of f is determined by the choice of P, we use a dependent type notation to specify the set of all Type 1 policies:

$$\text{Type-1}(V) = \langle P : \mathrm{PT}(T_V), P \rightarrow \mathrm{ER}(V) \rangle$$

Because we want to allow only low dependency – i.e., the erasure condition must be independent of the erasure subject – we require that P is *total for* V, by which we mean:

$$\forall X \in P . \Phi(X) = V$$

This means that knowing the value of the condition will not in itself rule out any possible subject values. To define policy satisfaction we use the components $X \in P$ to partition a system S into disjoint sub-systems $S \cap X$ and check both that each sub-system is defined over the whole subject domain V (again, to ensure low dependency) and that it satisfies the Type 0 policy for sub-domain X. So, for a Type 1 policy $\langle P, f \rangle \in \text{Type-1}(V)$, an attacker model A, and system $S \in \mathrm{Sys}(V)$, satisfaction is defined thus:

$$S \vdash^{o}_{A} \langle P, f \rangle \text{ iff } \forall X \in P . S_X \in \mathrm{Sys}(V) \wedge S_X \vdash^{o}_{A} f\ X$$

where $S_X = S \cap X$.

Example. Consider the example of Listing 1.3 extended with an erasure signal followed by a memory dump (as in our discussion of Type 0 policies above). Let S be the system model for the extended program. We specify a conditional erasure policy where the condition depends solely on the user choice. The erasure condition can be formalised as the partition $\mathrm{Ch} \in \mathrm{PT}(T)$ with two parts, one for traces where the user answers "Allow" (which we abbreviate to a) and one for traces where he doesn't: $\mathrm{Ch} = \{Y, \overline{Y}\}$, where $Y = \{t \in T | \exists s, s_1, s_2.\ t = s.?a.s_1.\eta.s_2\}$ and $\overline{Y} = T \setminus Y$. For runs falling in the Y component, the intended visibility policy is L4, as in the Type 0 example above. For all other runs, the intended policy is $\mathrm{All}_{\mathrm{CC}}$, specifying complete erasure. The Type 1 policy is thus $\langle \mathrm{Ch}, g \rangle$ where $g : \mathrm{Ch} \rightarrow \mathrm{ER}(\mathrm{CC})$ is given by:

$$g(X) = \begin{cases} \text{L4} & \text{if } X = Y \\ \text{All} & \text{if } X = \overline{Y} \end{cases}$$

Intersecting Y and $\overline{Y}$, respectively, with the system model S gives disjoint subsystems S_Y (all the runs in which the user enters "Allow" to permit retention of the last four digits) and $S_{\overline{Y}}$ (all the other runs). Since the user's erasure choice is input independently of the card number, it is easy to see that both sub-systems are in Sys(CC), that $S_Y \vdash_{\mathrm{AS}}$ L4, and $S_{\overline{Y}} \vdash_{\mathrm{AS}}$ All. Thus $S \vdash_{\mathrm{AS}} \langle \mathrm{Ch}, g \rangle$.

The following theorem establishes that our "explicitness" design principle is realised by Type 1 policies:

Theorem 1. *Let* $\langle P, f \rangle \in$ *Type*-$1(V)$ *and* $S \in$ *Sys*(V) *and* $A \in$ *ER*(T)*. Let* $o \in \{U, L, EM, CA, WA\}$*. If* $S \vdash^o_A \langle P, f \rangle$ *then:*

$$[\bigwedge_{X \in P} (f\ X)] \preceq_o \mathcal{K}_S(A)$$

Example. Consider instantiating the theorem to the policy $\langle \mathrm{Ch}, g \rangle$ described above. Here the policy is built from the two equivalence relations All and L4; the theorem tells us that the knowledge of the attacker is bounded by the meet of these components (and hence nothing that is not an explicit part of the policy) i.e., All $\wedge$ L4, which is equivalent to just L4.

Type 2 Policies

Type 2 policies are the most flexible policies we consider, allowing dependency on both the erasure subject and other properties of a run.

Recall the motivating example from Section 2 (Listing 1.4) in which credit card numbers in a particular set (the StealthCards) SC $\subseteq$ CC are always erased, while the user is given some choice for other card numbers. In this example, the dependency of the policy on the erasure subject can be modelled by the partition HC $= \{\mathrm{SC}, \overline{\mathrm{SC}}\}$. For each of these two cases, we can specify sub-policies which apply only to card numbers in the corresponding subsets. Since these sub-policies do not involve any further dependence on the erasure subject, they can both be formulated as Type 1 policies for their respective sub-domains. In general then, we define the Type 2 policies as follows:

$$\text{Type-2}(V) = \langle Q : \mathrm{PT}(V), W : Q \rightarrow \text{Type-1}(W) \rangle$$

To define satisfaction for Type 2 policies, we use the components $W \in Q$ to partition a system S into sub-systems (unlike the analogous situation with Type 1 policies, we cannot intersect S directly with W; instead, we intersect with T_W). To ensure that the *only* dependency on the erasure subject is that described by Q, we require that each sub-system $S \cap T_W$ is defined over the whole of the subject sub-domain W. So, for a Type 2 policy $\langle Q, g \rangle \in$ Type-2(V), an attacker model A, and system $S \in \mathrm{Sys}(V)$, satisfaction is defined thus:

$$S \vdash^o_A \langle Q, g \rangle \text{ iff } \forall W \in Q . S_W \in \mathrm{Sys}(W) \wedge S_W \vdash^o_A g\ W$$

where $S_W = S \cap T_W$.

To state the appropriate analogue of Theorem 1 we need to form a conjunction of all the component parts of a Type 2 policy:

- In the worst case, the attacker will be able to observe which of the erasure cases specified by Q contains the subject, hence we should conjoin the corresponding equivalence relation $\mathcal{E}(Q)$.
- Each Type 1 sub-policy determines a worst case equivalence relation, as defined in Theorem 1. To conjoin these relations, we must first extend each one from its sub-domain to the whole domain, by appending a single additional equivalence class comprising all the "missing" elements: given $W \subseteq V$ and $R \in \mathrm{ER}(W)$, define $R^\dagger \in \mathrm{ER}(V)$ by $R^\dagger = R \cup \mathrm{All}_{V \setminus W}$.

Theorem 2. *Let $\langle Q, g\rangle \in \mathit{Type}\text{-}2(V)$ and $S \in \mathit{Sys}(V)$ and $A \in \mathit{ER}(T)$. For any Type 1 policy $\langle P, f\rangle$, let $R_{\langle P,f\rangle} = \bigwedge_{X \in P}(f\ X)$. Let $o \in \{U, L, \mathit{EM}, \mathit{CA}, \mathit{WA}\}$. If $S \vdash^o_A \langle Q, g\rangle$ then:*

$$[\mathcal{E}(Q) \wedge \bigwedge_{W \in Q} R^\dagger_{(g\ W)}] \preceq_o \mathcal{K}_S(A)$$

Example. We consider a Type 2 policy satisfied by Listing 1.4, namely $\langle \mathrm{HC}, h\rangle$ where HC is the partition into Stealth and non-Stealth cards (as above), and h is defined as follows.

$$\begin{array}{ll} h(\mathrm{SC}) = \langle\{T_{\mathrm{SC}}\}, \lambda x.\mathrm{All}_{\mathrm{SC}}\rangle & \qquad h_1(Y) = \mathrm{L4}_{\overline{\mathrm{SC}}} \\ h(\overline{\mathrm{SC}}) = \langle \mathrm{Ch}, h_1\rangle & \qquad h_1(\overline{Y}) = \mathrm{All}_{\overline{\mathrm{SC}}} \end{array}$$

The term T_{SC} denotes the set of traces which input a Stealth card number as first action. As in the example of Type 1 policy above, Y is the set of (non-stealth) traces where the user gives permission ("Yes") to retain the last digits, $\overline{Y}$ is its complement (relative to the set of non-stealth traces), and Ch is the partition $\{Y, \overline{Y}\}$. The term $\mathrm{L4}_{\overline{\mathrm{SC}}}$ denotes the restriction of $L4$ to elements in $\overline{\mathrm{SC}}$. Instantiating Theorem 2 to this example tells us that the attacker knowledge is bounded by $\mathcal{E}(\mathrm{HC}) \wedge \mathrm{All}^\dagger_{\mathrm{SC}} \wedge \mathrm{L4}^\dagger_{\overline{\mathrm{SC}}} \wedge \mathrm{All}^\dagger_{\overline{\mathrm{SC}}}$, which is just $\mathrm{L4}^\dagger_{\overline{\mathrm{SC}}}$.

4.1 Varying the Attacker Model

The hierarchy deals with erasure policies independently of any particular attacker model. Here we make some brief remarks about modelling attackers. Let us take the example of the erasure notion studied in [18] where the systems are simple imperative programs involving IO on public and secret channels. Then the *implicit* attacker model in that work is unable to observe any IO events prior to the erasure point, and is able to observe just the public inputs and outputs thereafter. (We note that [18] also considers a policy *enforcement* mechanism which uses a stronger, state-based non-interference property.)

Now consider the example of the one-time pad described in the introduction, codified in Listing 1.6. Let system S be the set of traces modelling the possible runs of the program and let the subject be the first input in each trace. For the simple erasure attacker AS (Section 4), unable to observe the key provided in line 2, the K-space will be $\{V\} = [\mathrm{All}]$, hence $S \vdash_{\mathrm{AS}} \mathrm{All}$. This is because the value of `data` in the output does not inform the attacker about the initial value.

```
get(data);
get(key);
data := data XOR key;
key := null;
erased();
output(data);
```

Listing 1.6. Key Erasure

On the other hand, the attacker who can also observe the key learns everything about the data from its encrypted value.[2] So for this stronger attacker, using encryption to achieve erasure does not work, and indeed policy satisfaction fails for this particular system.

If the attacker is strengthened even further, we arrive at a point where *no* system will be able to satisfy the policy. Intuitively, if an attacker can see the erasure subject itself (or, more specifically, more of the erasure subject than the policy permits to be retained) no system will be able to satisfy the policy. In general, we say that a policy p with subject domain V (where p may be of any of Types 0,1,2) is *weakly o-compatible* with attacker model A iff there exists $S \in \text{Sys}(V)$ such that $S \vdash_A^o p$ (we call this weak compatibility because it assumes that all $S \in \text{Sys}(V)$ are of interest but in general there will be additional constraints on the admissible systems). Clearly, to be helpful as a sanity check on policies we need something a little more constructive than this. For the special case of Type 0 policies and the upper ordering we have the following characterisation:

Lemma 2. *R is weakly U-compatible with A iff $\forall v \in V. \exists O \in [A].[v]_R \subseteq \Phi(O)$.*

Deriving analogues of this result (or at least sufficient conditions) of more general applicability remains a subject for further work.

Finally, we note that, while our main aim has been to specify erasure policies, by varying the attacker model appropriately, we can specify quite general information-flow properties, not just erasure policies. For example, by classifying events into High and Low and defining the attacker who sees only Low events, we can specify non-interference properties.

5 Related Work

We consider related work both directly concerned with erasure and more generally with knowledge based approaches to information flow policies.

Erasure. The information-flow perspective on erasure was introduced by Chong and Myers [7] and was studied in combination with confidentiality and declassification. Their semantics is based on an adaptation of two-run noninterference definitions, and does not have a clear attacker model. They describe conditional erasure policies where the condition is independent of the data to be erased. Although this appears similar to Type 1 policies (restricted to total erasure), it is more accurately viewed as a form of Type 0 policy in which the condition defines the point in the trace from which the attacker begins observation.

[2] Note, however, that we cannot model the fact that certain functions are not (easily) invertible, so our attackers are always endowed with unbounded computational power.

The present paper does not model the behaviour of the user who interacts with an erasing system. This was studied in [15] for one particular system and attacker model. We believe that it would be possible to extend the system model with a user-strategy parameter (see [31,24,24] which consider explicit models of user strategies). Neither do we consider here the verification or enforcement of erasure policies; for specific systems and attacker models this has been studied in a programming language context in [18,8,9,14,23].

Knowledge Based Approaches. Our use of knowledge sets was inspired by Askarov and Sabelfeld's *gradual release* definitions [2]. This provides a clear attacker-oriented perspective on information-flow properties based on what an attacker can deduce about a secret after making observations. A number of recent papers have followed this approach to provide semantics for richer information flow properties, e.g. [4,5]. Our use of knowledge sets to build a K-space, thus generalising the use of equivalence relations/partitions, is new. The use of partitions in expressing a variety of information flow properties was studied in early work by Cohen [10]. The use of equivalence relations and more generally partial equivalence relations as models for information and information flow was studied in [20] and resp. [27].

Recent work [3] uses an epistemic temporal logic as a specification language for information flow policies. Formulae are interpreted over trace-based models of programs in a simple sequential while language (without input actions), together with an explicit observer defined via an observation function on traces. Our work looks very similar in spirit to [3], though this requires further investigation, and it appears that our modelling capabilities are comparable. The use of temporal logic in [3] is attractive, for example because of the possibility of using off the shelf model-checking tools. However, our policy language allows a more intuitive reading and clear representation of the information leakage.

Alur *et al* [1], study preservation of secrecy under refinement. The information flow model of that work bears a number of similarities with the present work. Differences include a more concrete treatment of traces, and a more abstract treatment of secrets. As here, equivalence relations are used to model an attacker's observational power, while knowledge models the ability of an attacker to determine the value of trace predicates. Their core definition of secrecy coincides with what we call secrecy of queries (viz., negation of "can answer"), although they do not consider counterparts to our other knowledge-based properties.

Abstract Non-interference. Abstract Non-Interference [17] has strong similarities with our use of K-spaces. In abstract non-interference, *upper closure operators* (uco's) are used to specify non-interference properties. The similarities with the current work become apparent when a uco is presented as a *Moore family*, which may be seen as a K-space closed under intersection.

[17] starts by defining the intuitive notion of *narrow* abstract non-interference (NANI) parameterized by two upper closure operators η (specifying what the attacker can observe of low inputs) and ρ (ditto low outputs). A weakness of

NANI is that it suffers from "deceptive flows", whereby a program failing to satisfy NANI might still be non-interfering. From our perspective, the deceptive flows problem arises because η fails to distinguish between what an attacker can *observe* of low inputs and what he should be allowed to *deduce* about them (i.e., everything). Since we specify the attacker model independently from the flow policy, the deceptive flows problem does not arise for us.

The deceptive flows problem is addressed in [17] by defining a more general notion of abstract non-interference (ANI) which introduces a third uco parameter ϕ. The definition of ANI adapts that of NANI by lifting the semantics of a program to an abstract version in which low inputs are abstracted by η and high inputs by ϕ. A potential criticism of this approach is that an intuitive reading is not clear, since it is based on an abstraction of the original program semantics. On the other hand, being based on Abstract Interpretation [12,11], abstract non-interference has the potential to leverage very well developed theory and static analysis algorithms for policy checking and enforcement. It would therefore be useful to explore the connections further and to attempt an analysis of the ANI definitions (see also additional variants in [21]) relating them to more intuitive properties based on knowledge sets. A starting point could be [19] which provides an alternative characterisation of NANI using equivalence relations.

Provenance. A recent abstract model of *information provenance* [6] is built on an information-flow foundation and has a number of similarities with our model, including a focus on an observer model as an equivalence relation, and a knowledge-based approach described in terms of queries that an observer can answer. Provenance is primarily concerned with a providing *sufficient* information to answer provenance-related questions. In secrecy and erasure one is concerned with *not* providing more than a certain amount.

6 Conclusions and Further Work

We have presented a rich, knowledge-based abstract framework for erasure policy specification, taking into account both quantitative and conditional aspects of the problem. Our model includes an explicit representation of the attacker. The knowledge-based approach guarantees an intuitive understanding of what it means for an attacker to deduce some information about the secret, and for a policy to provide an upper bound to these deductions.

Our work so far suggests a number of possible extensions. At this stage, the most relevant ones on the theoretical side are:

- Develop a logic defined on traces, both to support policy definition and to give the basis for an enforcement mechanism (as is done in [3]).
- Model multilevel erasure, based on the fact the attacker might perform observations up-to a certain level in the security lattice. It would be interesting to investigate different classes of such attackers and to analyse their properties.
- Generalise policy specifications to use K-spaces in place of equivalence relations. This would allow specification of disjunctive policies such as "reveal

the key or the ciphertext, but not both". Non-ER policies may also be more appropriate for protection of facts, rather than queries, since ER's are effectively closed under complementation and so cannot reveal a fact without also revealing its negation (for example, we may be prepared to reveal "not HIV positive" to an insurance company, but not the negation of this fact).
- Extend the scope of the approach along the following key *dimensions* (defined in the same spirit as [28]):
  What: Our model is possibilistic but it is well known that possibilistic security guarantees can be very weak when non-determinism is resolved probabilistically (see the example in Section 5 of [26]). A probabilistic approach would be more expressive and provide stronger guarantees.
  When: Our policies support history-based erasure conditions but many scenarios require reasoning about the future ("erase this account in 3 weeks"). This would require a richer semantic setting in which time is modelled more explicitly.
  Who: We do not explicitly model the user's behaviour but it is implicit in our possibilistic approach that the user behaves non-deterministically and, in particular, that later inputs are chosen independently of the erasure subject. Modelling user behaviour explicitly would allow us to relax this assumption (which is not realistic in all scenarios) and also to model active attackers.
- Understand the interplay between erasure and cryptographic concepts. To make this possible some refinements of the theory are needed. Firstly, it would be natural to move to a probabilistic system model. Secondly, the present notion of knowledge assumes an attacker with computationally unlimited deductive power; instead we would need a notion of feasibly computable knowledge.

We have focussed on characterising expressive erasure policies, but not on their verification for actual systems. As a step towards bridging this to more practical experiments in information erasure, it would be instructive to explore the connections to the rich policies expressible by the enforcement mechanism for Python programs we describe in our earlier work [14].

Acknowledgment. Alejandro Russo provided valuable input throughout the development of this work. Many thanks to the anonymous referees for useful comments and observations. This work was partially financed by grants from the Swedish research agencies VR and SSF, and the European Commission EC FP7-ICT-STREP WebSand project.

References

1. Alur, R., Černý, P., Zdancewic, S.: Preserving Secrecy Under Refinement. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 107–118. Springer, Heidelberg (2006)

2. Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy, SP 2007, pp. 207–221. IEEE Computer Society, Washington, DC, USA (2007)
3. Balliu, M., Dam, M., Le Guernic, G.: Epistemic temporal logic for information flow security. In: ACM SIGPLAN Sixth Workshop on Programming Languages and Analysis for Security (June 2011)
4. Banerjee, A.: Expressive declassification policies and modular static enforcement. In: Proc. IEEE Symp. on Security and Privacy, pp. 339–353 (2008)
5. Broberg, N., Sands, D.: Flow-sensitive semantics for dynamic information flow policies. In: ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security (PLAS 2009), June 15. ACM (2009)
6. Cheney, J.: A formal framework for provenance security. In: The 24th IEEE Computer Security Foundations Symposium (June 2011)
7. Chong, S., Myers, A.: Language-based information erasure. In: 18th IEEE Workshop on Computer Security Foundations, CSFW-18 2005, pp. 241–254 (June 2005)
8. Chong, S.: Expressive and Enforceable Information Security Policies. Ph.D. thesis, Cornell University (August 2008)
9. Chong, S., Myers, A.C.: End-to-end enforcement of erasure and declassification. In: CSF, pp. 98–111. IEEE Computer Society (2008)
10. Cohen, E.S.: Information transmission in sequential programs. In: DeMillo, R.A., Dobkin, D.P., Jones, A.K., Lipton, R.J. (eds.) Foundations of Secure Computation, pp. 297–335. Academic Press (1978)
11. Cousot, P.: Semantic foundations of program analysis. In: Muchnick, S., Jones, N. (eds.) Program Flow Analysis: Theory and Applications, ch.10, pp. 303–342. Prentice-Hall, Inc., Englewood Cliffs (1981)
12. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 238–252 (January 1977)
13. Del Tedesco, F., Hunt, S., Sands, D.: A semantic hierarchy for erasure policies (extended version). In: International Conference on Information System Security (2011), http://arxiv.org/abs/1109.6914
14. Del Tedesco, F., Russo, A., Sands, D.: Implementing erasure policies using taint analysis. In: Aura, T. (ed.) The 15th Nordic Conference in Secure IT Systems. LNCS. Springer, Heidelberg (October 2010)
15. Del Tedesco, F., Sands, D.: A user model for information erasure. In: 7th International Workshop on Security Issues in Concurrency (SECCO 2009), pp. 16–30 (2009)
16. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. J. Computer Security 3(1), 5–33 (1995)
17. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: Parameterizing non-interference by abstract interpretation. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 186–197 (January 2004)
18. Hunt, S., Sands, D.: Just Forget it – The Semantics and Enforcement of Information Erasure. In: Gairing, M. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 239–253. Springer, Heidelberg (2008)
19. Hunt, S., Mastroeni, I.: The Per Model of Abstract Non-Interference. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 171–185. Springer, Heidelberg (2005)
20. Landauer, J., Redmond, T.: A lattice of information. In: Proc. IEEE Computer Security Foundations Workshop, pp. 65–70 (June 1993)

21. Mastroeni, I.: On the Rôle of Abstract Non-Interference in Language-Based Security. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 418–433. Springer, Heidelberg (2005)
22. McLean, J.: Security models and information flow. In: Proc. IEEE Symp. on Security and Privacy, pp. 180–187 (May 1990)
23. Nanevski, A., Banerjee, A., Garg, D.: Verification of information flow and access control policies with dependent types. In: Proc. IEEE Symp. on Security and Privacy (2011)
24. O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW 2006: Proceedings of the 19th IEEE Workshop on Computer Security Foundations, pp. 190–201. IEEE Computer Society, Washington, DC, USA (2006)
25. Plotkin, G.D.: A powerdomain construction. SIAM J. Comput. pp. 452–487 (1976)
26. 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. 40–58. Springer, Heidelberg (1999)
27. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation 14(1), 59–91 (2001)
28. Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. Journal of Computer Security 15(5), 517–548 (2009)
29. Sutherland, D.: A model of information. In: Proc. National Computer Security Conference, pp. 175–183 (September 1986)
30. Wei, M.Y.C., Grupp, L.M., Spada, F.E., Swanson, S.: Reliably erasing data from flash-based solid state drives. In: 9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, pp. 105–117. USENIX (2011)
31. Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symposium on Security and Privacy, pp. 144–161 (1990)

A Universal Semantic Bridge for Virtual Machine Introspection

Christian Schneider, Jonas Pfoh, and Claudia Eckert

Technische Universität München
Munich, Germany
{schneidc,pfoh,eckertc}@in.tum.de

Abstract. All systems that utilize virtual machine introspection (VMI) need to overcome the disconnect between the low-level state that the hypervisor sees and its semantics within the guest. This problem has become well-known as the *semantic gap*. In this work, we introduce our tool, *InSight*, that establishes a semantic connection between the guest and the hypervisor independent of the application at hand. InSight goes above and beyond previous approaches in that it strives to expose *all* kernel objects to an application with as little human effort as possible. It features a shell interface for interactive inspection as well as a scripting engine for comfortable and safe development of new VMI-based methods. Due to this flexibility, InSight supports a wide variety of VMI applications, such as intrusion detection, forensic analysis, malware analysis, and kernel debugging.

1 Introduction

Bridging the semantic gap requires knowledge about the virtual hardware architecture, the guest operating system (OS), or both. A VMI component that bridges the semantic gap by applying such knowledge is referred to as a *view generating component* (VGC) [6]. Most VGCs target exactly that portion of the VM state necessary for a particular VMI application, for example, protection of system hooks [7,8], monitoring of critical data structures [3,7], or comparison of data collected from the hypervisor and from within the guest [3,4,5]. Recently, cutting edge research has begun to focus on stand-alone view generation [2]. However, such work lacks the completeness and flexibility necessary for the wide range of possible VMI applications.

For our VMI research, we have implemented a VGC called *InSight* [1] to bridge the semantic gap on the Intel x86 and AMD64 platforms. In contrast to the aforementioned approaches, the goal of InSight is to make *all* kernel objects and their associated values available with as little human effort or expert knowledge as possible. InSight strives to interpret the entire kernel memory of a running guest OS in the same way the guest kernel does. It is very flexible in that it provides access to the VM state through a powerful command line interface as well as through a scripting engine to directly interact with kernel objects as JavaScript objects. This way, the actual VMI application is completely decoupled from the view generation process, making InSight a universal tool for various applications such as intrusion detection and forensic analysis, as well as for kernel debugging.

2 Challenges in View Generation

Reconstructing a VM's state from the vantage point of the hypervisor requires the following knowledge of the guest OS as well as the virtual hardware architecture: (1) the layout of the kernel address space, (2) the layout and size of kernel data structures, (3) the location of kernel objects in the kernel's address space, and (4) the function of the virtual-to-physical address translation. This knowledge is delivered to the VGC out-of-band—that is, before the introspection starts [6]—in the form of, for example, kernel debugging symbols and a software-based memory management unit. The view is then generated by applying this knowledge to the guest physical memory. When put into practice, re-creating the VM's state with this knowledge at hand provides several challenges, as we will outline in the following. Up until now, these challenges have been solved by manually applying expert knowledge on a case-by-case basis.

OS kernels organize objects in efficient data structures such as, for example, linked lists or hash tables. A common implementation of these lists and tables works by defining generic "node" data structures along with auxiliary functions and macros to operate on them. These nodes must then be embedded as a member of the kernel object's definition to be organized in a list or table. To retrieve objects stored in such data structures, the auxiliary functions operate only on the embedded nodes but still make the embedding object available through address manipulations and type casts. When a VGC aims to make the objects stored in such a list or table available, it faces several problems. First, the head to such data structures is often a plain node itself. There is no hint in the debugging symbols as to which object type is stored in a particular list. Second, once the VGC has access to an object stored in a list or table, the type information does not indicate which object type lurks behind the pointers of the embedded node member. To make matters worse, many structures contain several node members as heads to other lists holding various types, making it impossible to guess the correct type based on heuristics.

Further uncertainties in inferring the correct object type and address result from untyped pointers, pointer casts from integer values, offset pointers, pointers with status flags encoded in their least significant bits, pointers that are used as arrays, and ambiguous types such as unions.

OS kernels perform various operations atomically or in critical sections to keep their state consistent. Detecting that the virtual CPU is currently in a critical section is a difficult task, as the debugging symbols do not provide enough information to discover such situations. In case a view is generated while the CPU is in a critical section, the VGC might encounter inconsistent data structures; this is even more likely for a VM with multiple CPUs. It must then be able to cope with such inconsistencies in a reliable way.

3 Implementation

For our VMI research, we need a view-generating component that automatically provides access to *all* kernel objects in memory, not only to a small number of selected data structures. In addition, we require a high-level, flexible, and powerful interface to these objects in oder to be able to perform sophisticated analysis of the guest OS state. Our tool, InSight, is such a view-generating component for the x86 and AMD64 platform that addresses many of the aforementioned challenges. It is written in C++ and operates on physical memory. Thus, it works with memory dump files as well as with any hypervisor that provides access to its guest's physical memory. For Linux guests, InSight fully supports the reading of arbitrary kernel objects for both 64-bit and 32-bit addressing schemes with and without physical address extension (PAE). Multiple memory files can be processed simultaneously (for example, to compare the current "live" state of a running VM to a previous snapshot).

We designed InSight to be as straightforward to use as possible. Whenever the member of a kernel object is accessed, the user receives a new kernel object of the type that this member represents; all pointers are automatically dereferenced. With regard to efficient lists and hash tables that are implemented as detailed in Section 2, InSight detects these "node" data structures and marks the particular members of the embedding structure accordingly. If the user accesses these members, he receives an object of the embedding type with its correct address, just as the kernel macros for operating on those data structures would do.

The current approach enables an easy enumeration of running processes and loaded kernel modules, for example. In situations in which total automation is not achieved, the flexible design of InSight allows the user to address this by encapsulating expert knowledge in JavaScript functions that are reusable for all applications. In an effort to reduce the amount of human intervention even further, we are already working on an improved solution which parses the kernel's source code in order to automatically resolve virtually all data types of pointers and type casts without requiring expert knowledge.

The InSight shell provides an interactive interface to analyze the state of a VM. It allows inspection of the kernel's data structures and global variables. For each loaded memory file, the kernel objects referenced by the global variables as well as their members can be accessed using the common notation `object.member`. In addition, the user may choose to read an object from an arbitrary virtual or physical address as any known data type, allowing the user to further follow any members of that type.

The true power of InSight lies in its included JavaScript engine which enables users to interact with kernel objects and to automate complex analysis tasks. The user can access global variables by requesting an `Instance` object of a variable by its name. If the resulting object represents a structure, its members can be accessed by their name using the "dot" notation just like any other property of a JavaScript object. In addition, an `Instance` object provides methods to access meta data of the corresponding kernel object, change its type, manipulate its address, and access array elements. Using all of these features, writing a function that prints out a list of loaded kernel modules requires less than ten lines of JavaScript code. Besides its simple application, another substantial benefit of the scripting engine in comparison to any low-level approach is the fact that any sort of runtime error will result in a JavaScript exception rather than a segmentation fault. All errors are contained within the scripting engine and do not propagate to the VGC itself. The user can modify and re-run the script instantly without having to recompile the source code, restart InSight, or restart the monitored VM. This is a major advantage of InSight over other approaches and greatly eases the development of new VMI mechanisms.

4 Conclusion

InSight focuses on the well-known semantic gap issue faced by all VMI applications. While other approaches strive to bridge the semantic gap for a particular problem, InSight takes a general approach. We strive to provide a VGC that gives a complete view of the guest kernel, is decoupled from the VMI component itself, and remains flexible enough to be applicable for forensics, intrusion detection, malware analysis, and kernel debugging. Our efforts have resulted in InSight which is an extremely powerful and flexible VGC offering several interfaces including a scripting engine for further automation of complex tasks. It has already proven to be very useful and is successfully applied in several projects within our research group. We have released InSight as an open source tool [1] to enable the fast and intuitive development of new VMI and forensic approaches.

References

1. InSight project website, `https://code.google.com/p/insight-vmi/`
2. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W.: Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland) (May 2011)
3. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. of NDSS, pp. 191–206 (2003)
4. Litty, L., Lagar-Cavilla, H.A., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: Proc. of the 17th Conf. on Security Symp., pp. 243–258. USENIX, Berkeley (2008)
5. Martignoni, L., Fattori, A., Paleari, R., Cavallaro, L.: Live and Trustworthy Forensic Analysis of Commodity Production Systems. In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 297–316. Springer, Heidelberg (2010)
6. Pfoh, J., Schneider, C., Eckert, C.: A formal model for virtual machine introspection. In: Proc. of 2nd Workshop on VM Sec. ACM, New York (2009)
7. Riley, R., Jiang, X., Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 1–20. Springer, Heidelberg (2008)
8. Wang, Z., Jiang, X., Cui, W., Ning, P.: Countering kernel rootkits with lightweight hook protection. In: Proc. of 16th Conf. on Computer and Communications Security, CCS 2009, pp. 545–554. ACM (2009)

A Signature-Based Approach of Correctness Assurance in Data Outsourcing Scenarios

Morteza Noferesti, Mohammad Ali Hadavi, and Rasool Jalili

Data and Network Security Lab,
Sharif University of Technology, Tehran, Iran
{mnoferesti,mhadavi}@ce.sharif.edu, jalili@sharif.edu

Abstract. Correctness assurance of query results in data outsourcing scenarios includes authenticity, completeness, and freshness of the results. Utilizing signature chaining and aggregation, this paper proposes a method to verify the correctness of results returned from an untrusted server. An MHT constructed over attribute values of a tuple is used to provide the authenticity, and timestamp is used to provide the freshness verifiability of results. While our approach supports a wide range of queries, simulation results indicate its efficiency in comparison with some existing methods in terms of communication and computation overhead imposed to execute a query.

Keywords: data outsourcing, correctness, freshness.

1 Introduction

Database outsourcing is a paradigm to reduce data management costs in which data along with its management is outsourced to a third party service provider. A generic model for data outsourcing consists of a data owner *OWNER*, a service provider *SP*, and some users *U* who submit their queries to the system. *OWNER* outsources data to *SP* and updates them whenever necessary.

Due to the untrustworthiness of the service provider *SP*, a lazy service provider or a malicious attacker who has compromised the server may violate the correctness of query results. So, a data outsourcing scheme is required to make a correctness proof for the client. To this purpose, the untrusted server sends some extra information to the client as a verification object together with the query results. Subsequently, the client uses the verification object to verify the query results correctness. The returned result is correct if it satisfies authenticity, completeness, and freshness.

In a dynamic environment where data changes frequently, *OWNER* must be able to update the outsourced data efficiently and *U* must receives the latest version of data. To cope with such a requirement, in reply to the received query *Q* from *U*, *SP* calculates the result *RST* and *VO*. Upon receiving the result, *U* verifies the *RST* based on the *VO* through a verification algorithm. The algorithm should be sound and complete. The soundness means if *RST* is a correct result to *Q*, the verification algorithm identifies it as a correct result.

The verification algorithm is complete if *SP* can generate *VO* of a correct result for every query where the algorithm identifies it as a correct result.

Our proposed approach enables users to verify correctness of query result. This approach imposes low overhead depending on the size of the query result. Considering a typical relational data model environment and queries formulated through SQL, our approach supports different types of query.

The existing approaches for correctness assurance are mainly based on authentication data structures or digital signatures. Merkel Hash Tree (MHT) is an authentication data structure, which has been used in several proposals [1,2]. It is constructed on a data set such that its root maintains the integrity of the whole data set. In digital signature-based approaches[3,4], the data owner sings data, and outsources the data along with their signatures. These signatures are used to verify the correctness of the returned result.

2 The Proposed Approach

Using digital signature schemes, we provide the correctness verifiability for clients at the granularity of a tuple. In the proposed approach, *OWNER* signs each tuple. The *OWNER*'s public key is used to verify authenticity of the returned results. Signature chaining is used to verify the result completeness. For this purpose, each tuple is linked to its previous and next ones by a collision-free hash function.

We provide online verification of the result freshness without involving the data owner. For this goal, we propose a scheme to verify freshness of returned result based on the fact that update frequency in a database is not the same for all attributes of a relation.

For example the table schema $SENSOR(SensorID, Loc, Tmp)$ provided by a weather forecasting company. Each sensor has a unique $SensorID$ which does not change during its lifetime, so it is persistent. Each sensor has been located at a specific location Loc and it may be changed with low frequent updates. These sensors periodically sample the temperature Tmp, so $Tmp's$ value changes with high frequency. We consider this example in the rest of this paper. we go into depth of our approach in the three following paragraphs:

Initial construction at the owner side: To outsource a table, some attributes are inserted into its schema. For example $SENSOR$ schema changes to $SENSOR(SensorID, Loc, Tmp, H(Loc-Prev), H(Loc-Next), H(Tmp-Prev), H(Tmp-Next), Time_{Loc-Expired}, Time_{Tmp-Expired})$. In this schema, $H(Loc-Prev)$ and $H(Loc-Next)$ presents the hash values of the previous and next location values of each tuple. $Time_{Loc-Expired}$ describes the freshness time interval of the Loc value. $H(Tmp-Prev)$, $H(Tmp-Next)$, and $Time_{Tmp-Expired}$ do the same for Tmp value. Finally, *OWNER* sings each tuple and stores it beside the tuple. The signature of an arbitrary tuple r is computed as equation 1 where $\|$ denotes concatenation, H(.) is a cryptographic hash function, and SK is the private signing key of the data owner:

$$Sign(r) = H(H(SensorID||H(Loc)||H(Tmp)||H(Loc-Prev)|| \\ H(Loc-Next)||H(Time_{Loc-Expired)}||H(Tmp-Prev)|| \\ H(Tmp-Next)||H(Time_{Tmp-Expired}))_{SK} \qquad (1)$$

Building Verification Object at the Server Side: For an arbitrary query, *SP* computes a set RST of satisfying tuples based on the query conditions. Then, it finds their associated signature and aggregates them as an aggregated signature σ as the verification object. *SP* sends RST and σ to the user.

Verifying Correctness at the User Side: The user assures the authenticity through verifying σ, ensuring that all members of RST are originated by $OWNER$. Hash values are used to verify the completeness of the returned result. To this end, a collision-free hash function, same as [5] is used, such that for each two distinct values m and n ($m \geq n \Rightarrow hash(m) \geq hash(n)$). The result is fresh if and only if the current time is less than or equal to the returned timestamps. We suppose that all the parties in the data outsourcing model have a synchronized clock.

3 Query Processing

SELECT: The SELECT operation is denoted by $\sigma_{<selection\ condition>}(R)$. It is used to select a subset from a relation R such that satisfying the <*selection condition*>. Consider a query selects entire rows of the SENSOR table where its *Tmp* value is in the range $[Tmp_{low}, Tmp_{high}]$. *SP* sends $RST = \{r_m, ..., r_n\}$, and its aggregated signature $\sigma = Aggregated(sign(r_m), ..., sign(r_n))$ to the client. U uses to verify the authenticity of the returned tuples. Hash values and timestamps in the returned tuples are used to verify the completeness and freshness of RST, respectively.

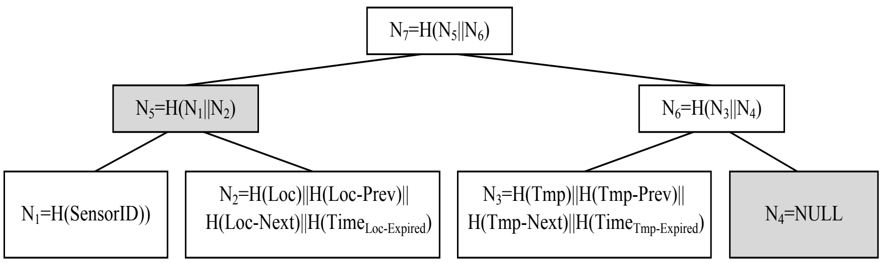

N_7=H(N_5||N_6)
N_5=H(N_1||N_2)
N_6=H(N_3||N_4)
N_1=H(SensorID))
N_2=H(Loc)||H(Loc-Prev)|| H(Loc-Next)||H($Time_{Loc-Expired}$)
N_3=H(Tmp)||H(Tmp-Prev)|| H(Tmp-Next)||H($Time_{Tmp-Expired}$)
N_4=NULL

Fig. 1. MHT is build over a tuple in the *SENSOR* table. Shades nodes represent the *VO* for *Tmp* value.

PROJECT: The PROJECT operation is indicated by $\pi_{<attribute\ list>}(R)$ where <*attribute list*> is the desired list of attributes from the attributes of relation

R. For a PROJECT operation, the overhead of verification process should be correspondent to the size of the projected attributes, not to the size of the entire tuple. To verify correctness of a PROJECT query result, $OWNER$ builds an MHT over values of a tuple and signs its root. as depicted in figure 1, SP sends with each attribute value, its co-path to the root and the root signature.

INSERT: The INSERT command is used to add a single tuple into a relation. For this, SP finds the next and previous tuples for each searchable attribute of the new tuple and sends them with an appropriate VO to $OWNER$. After that, $OWNER$ resigns the adjacent tuples with respect to the values of the new tuple. He also sets the timestamps of the new tuple, makes an MHT over it, and signs the MHT′s root. Then the updated signatures and the new tuple together with its signature are sent back to SP. Regarding this process, the cost of inserting a new tuple into a relation with n searchable attributes is calculated as equation 2 where S_x denotes the size of x, $T_{verification}$ denotes the time of signature verification, and $T_{signature}$ denotes the time of producing signature:

$$\begin{aligned} communication\ cost &= O((2n+1)*S_{tuple} + (2n+2)*S_{signature} \\ computation\ cost &= O(T_{verification} + (2n+1)*T_{signature} \end{aligned} \qquad (2)$$

Due to space restriction, we just mentioned the cost of UPDATE operation to modify m searchable attributes of a tuple in equation 3.

$$\begin{aligned} communication\ cost &= O((2n+1)*S_{tuple} + (2n+2)*S_{signature} \\ computation\ cost &= O(T_{verification} + (2n+1)*T_{signature} \end{aligned} \qquad (3)$$

4 Implementation and Result

We simulated our approach and compared the results with Goodrich et al.s MHT-based and Narasimha and Tsudiks signature-based methods. We executed a simple SELECT query to retrieve appropriate random tuples from the $SENSOR$ table which had been filled with 100000 random records.

Figure 2(a) indicates that the execution time of our approach is less than the others. In the Narasimha and Tsudiks approach, SP sorts database per each searchable attribute in the table schema, but in our approach database is sorted according to the searchable attributes that are presented in the query condition. Result verification in MHT-based approach contains executing several hash and concatenation functions and verifying the root signature. On the contrary, result verification in our approach consists of verifying an aggregated signature which makes ours faster than the MHT-based approach.

Figure 2(b) compares the communication cost of different approaches. Communication cost refers to the volume of VO transferred from SP to the user while processing a SELECT query. In the Narasimha and Tsudiks signature based approach VO contains two boundary tuples for each tuple in the result size, but in our approach there are two hash values for each tuple. VO in the MHT-based approach contains its co-path to the root and the root signature, so its size is higher than the size of VO in our approach.

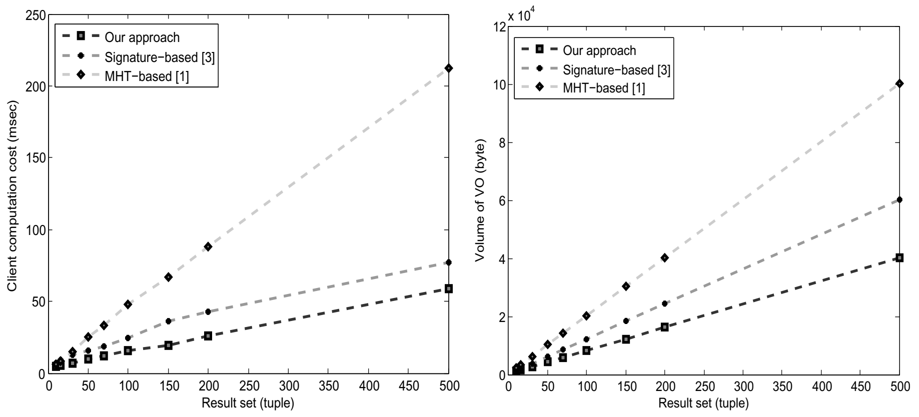


Fig. 2. Client computation cost(a) and communication cost(b) of different approaches

5 Conclusion

This work investigated the correctness problem of query results when data is outsourced to an untrusted service provider. The query results are correct if and only if they were authentic, complete, and fresh. We use signature aggregation schemes to verify authenticity of query results. The tuples in the outsourced database are chained with a collision free hash function. These chains are used to verify the completeness of query results. We used timestamps to provide online freshness verification of a query result. This approach allows the data owner to categorize its data and update them based on their change frequency. Moreover, according to the philosophy of data outsourcing, in our approach the cooperation of the data owner does not required in the verification process.

References

1. Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Super-Efficient Verification of Dynamic Outsourced Databases. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 407–424. Springer, Heidelberg (2008)
2. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data - SIGMOD 2006, p. 121 (2006)
3. Narasimha, M., Tsudik, G.: DSAC: An Approach to Ensure Integrity of Outsourced Databases using Signature Aggregation and Chaining. In: ACM Conf. on Information and Knowledge Management, pp. 420–436 (2005)
4. Pang, H.H., Zhang, J., Mouratidis, K.: Scalable verification for outsourced dynamic databases. In: Proceedings of the VLDB Endowment, vol. 2, pp. 802–813 (2009)
5. Agrawal, D., Abbadi, A.E., Emekci, F., Metwally, A.: Database Management as a Service: Challenges and Opportunities. In: 2009 IEEE 25th International Conference on Data Engineering, pp. 1709–1716 (March 2009)

Towards Access Control Model Engineering

Winfried E. Kühnhauser and Anja Pölck

Ilmenau University of Technology, Ilmenau, Germany
{winfried.kuehnhauser,anja.poelck}@tu-ilmenau.de

Abstract. Formal security models have significantly improved the understanding of access control systems. They have influenced the way access control policies are specified and analyzed, and they provide a sound foundation for a policy's implementation.

While their merits are many, designing security models is not an easy task, and their use in commercial systems is still far from everyday practice. This paper argues that model engineering principles and tools supporting these principles are important steps towards model based security engineering. It proposes a model engineering approach based on the idea that access control models share a common, model-independent core that, by core specialization and core extension, can be tailored to a broad scope of domain-specific access control models.

Keywords: security engineering, security policies, security models, security model engineering, access control models.

1 Introduction

IT systems with advanced security requirements increasingly apply problem-specific security policies for describing, analyzing, and implementing security properties, e.g. [3,9]. In order to precisely describe security policies, security models like [6,10] are applied, allowing for a formal analysis of security properties and serving as specifications from which policy implementations are engineered [11,12].

Since the appearance of the first security models, the scope of security-relevant applications has grown rapidly, and many application-specific security models have emerged. Models were tailored to specific application domains (such as work flow management or health care), and their basic paradigms (such as access control matrices or role hierarchies) reflect such specialization.

Domain-specific security models have both advantages as well as disadvantages: model paradigms that are well-tuned to application-specific security requirements result in small, elegant, and simple models. On the other hand, from a model engineering point of view, model diversity is an obstacle to engineering efficiency (e.g. by model reuse) and model-independent tool support (such as model analysis).

This paper motivates and identifies basic, model-independent engineering principles for access control (AC) security models. The idea is to identify a domain-independent model core shared by a broad scope of access control models that can be tailored to domain-specific access control models by core specialization and core extension.

2 Idea

This section first sketches engineering principles shared by many contemporary AC models. The idea of core-based model engineering is introduced afterwards.

Butler Lampson, the pioneer of AC models, published one of the first AC models in 1971 [8]. The goal of his model was to provide precise rules about which subjects in a computer system (users or processes) are allowed to execute specific operations (such as read or write) on the objects (e.g. files or printers) in that system. Lampson used a simple *access control function (ACF)*: $subjects \times objects \times operations \rightarrow \{true, false\}$ (often represented by an isomorphic access control matrix (ACM)) to model a static snapshot of the protection state.

In order to describe real-world policies, for which modifying privileges or adding/deleting users are important features, Lampson's model was augmented by a deterministic state automaton [5,6] with the objective to model dynamic changes of the protection state. In the following years, innumerable models have emerged from this approach, e.g. [7,11,13]. All of these AC models are based on an automaton (Q, Σ, δ, q_0) where Q is the state space, Σ is a finite set of inputs, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, and $q_0 \in Q$ is the initial state. They all have in common that each $q \in Q$ represents a policy's protection state containing a derivative of Lampson's ACM and other components like attribute or role assignments. For example, the ABAM model [13] combines the ACM with subject and object attributes; and both the ACM and the attribute assignments are state components. Hence, many dynamic AC models share a state automaton to describe policy dynamics. For all of them holds that a state of the automaton represents a policy's protection state, e.g. a system's right or role assignments, and policy dynamics are represented by changing right or role assignments via state transitions.

The central idea of core-based model engineering is that these design principles provide a model-independent fundament – referred to as *model core* – for a model engineering method. The model core builds a universal foundation for a broad scope of AC models, and domain-specific AC models are engineered by reusing the core and adapting it to domain-specific model paradigms. The objective is to facilitate the complex process of model engineering by model reuse and thus provide a common specification for AC models. This again enables model-independent tool-support for model engineering and model analysis.

The model core is a deterministic state automaton which is already known from the HRU model [6]. However, it generalizes the HRU automaton in such a way that it supports a broad scope of AC models and then allows for its specialization in order to engineer domain-specific AC models. Models are derived from the model core using core specialization and core extension inspired by generalization/specialization concepts of object-oriented software design.

In object-oriented design, a subclass is derived from a super class by inheriting attributes, operations, and constraints of the super class with the goal to reuse software. Subclasses are then allowed to add new features, e.g. new attributes and operations, or to override inherited operations in order to express individual class characteristics. In the context of security models, the model core corresponds to a super class and any security model corresponds to a subclass that can be derived from the model core. Consequently, models inherit the core's components Q, Σ, δ, and q_0 which may then be specialized. Result is a model-specific state automaton to model a policy's dynamic components, i.e. components of the protection state, and rules to modify them. The latter is modeled by the state transition function whose responsibility is to make policy decisions and alter a policy's protection state.

While core specialization focuses on dynamic model components, AC security models also require static model components that do not change during policy runtime, e.g. a system's right set, subject/object typification, or attribute tuples. Thus, the model core can be further tailored to a domain-specific model by core extensions representing such static components. In order to consider static extensions in policy decisions, interrelations between static and dynamic model components can be defined by the transition function δ.

Result is a methodical, straightforward, and manageable security model engineering approach whose contributions are as follows. Core-based model engineering provides the basis for a common specification of AC models and AC policies resulting in general advantages like usability and maintainability. More precisely, the unifying model core contributes to facilitate the typically burdensome process of model engineering and policy modeling by (i) establishing general semantics for AC models and (ii) providing basic model-engineering principles for AC models. In this way the contributions of core-based model engineering are similar to [2]. In contrast to [2], the main merit of core-based model engineering is, however, that it allows for tool support for model engineering and model analysis. That is, once a policy has been described by a model in core notation, it is unlocked to a family of core-based analyzing methods that implements state reachability analyses by heuristic search algorithms [1,4].

The approach of core-based model engineering also supports the activity of policy implementation. It is ongoing work to design a system's trusted computing base (TCB) by deriving its functionality from core-based security models. The objective is to establish a comprehensive and tool-supported security engineering process for modeling, analyzing, and implementing security policies.

3 Conclusion

This paper argues that contemporary access control models share common paradigms that form a model-independent fundament for methods and tools of model engineering. The paper identified a unified model core shared by a broad scope of access control models. Inspired by object-oriented software engineering, model engineering is based on deriving models from the common core by specializing and extending the core. This sets the course for a straightforward and methodical security engineering approach enabling tool support for model engineering, model analysis, and model implementation.

Acknowledgements. This work was partially supported by Carl-Zeiss-Stiftung.

References

1. Amthor, P., Kühnhauser, W.E., Pölck, A.: Model-based Safety Analysis of SELinux Security Policies. In: Samarati, P., Foresti, S., J.H.G. (eds.) Proc. of 5th Int. Conference on Network and System Security, pp. 208–215. IEEE (2011)
2. Barker, S.: The Next 700 Access Control Models or a Unifying Meta-Model? In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, pp. 187–196. ACM, New York (2009)
3. Benats, G., Bandara, A., Yu, Y., Colin, J.N., Nuseibeh, B.: PrimAndroid: Privacy Policy Modelling and Analysis for Android Applications. In: 2011 IEEE International Symposium on Policies for Distributed Systems and Networks (Policy 2011), pp. 129–132. IEEE (2011)
4. Fischer, A., Kühnhauser, W.E.: Efficient Algorithmic Safety Analysis of HRU Security Models. In: Katsikas, S., Samarati, P. (eds.) Proc. International Conference on Security and Cryptography (SECRYPT 2010), pp. 49–58. SciTePress (2010)
5. Graham, G.S., Denning, P.J.: Protection: Principles and Practice. In: AFIPS 1972 (Spring): Proceedings of the Spring Joint Computer Conference, May 16-18, pp. 417–429. ACM, New York (1972)
6. Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: On Protection in Operating Systems. Operating Systems Review, special issue for the 5th Symposium on Operating Systems Principles 9(5), 14–24 (1975)
7. Jha, S., Li, N., Tripunitara, M., Wang, Q., Winsborough, W.: Towards Formal Verification of Role-Based Access Control Policies. IEEE Transactions on Dependable Secure Computing 5, 242–255 (2008)
8. Lampson, B.W.: Protection. In: Fifth Annual Princeton Conference on Information Sciences and Systems, pp. 437–443 (March 1971); Protection. Operating Systems Review 8(1), 18–24 (reprinted January, 1974)
9. Loscocco, P.A., Smalley, S.D.: Integrating Flexible Support for Security Policies into the Linux Operating System. In: Cole, C. (ed.) Proc. 2001 USENIX Annual Technical Conference, pp. 29–42 (2001)
10. Marinovic, S., Craven, R., Ma, J., Dulay, N.: Rumpole: a Flexible Break-glass Access Control Model. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT 2011, pp. 73–82. ACM (2011)
11. Sandhu, R.S.: The Typed Access Matrix Model. In: Proc. IEEE Symposium on Security and Privacy, pp. 122–136. IEEE (May 1992)
12. Zanin, G., Mancini, L.V.: Towards a Formal Model for Security Policies Specification and Validation in the SELinux System. In: Proc. of the 9th ACM Symposium on Access Control Models and Technologies, pp. 136–145. ACM (2004)
13. Zhang, X., Li, Y., Nalla, D.: An Attribute-based Access Matrix Model. In: Proc. of the 2005 ACM Symposium on Applied Computing, pp. 359–363. ACM (2005)

IFrandbox - Client Side Protection from Malicious Injected Iframes

Tanusha S. Nadkarni, Radhesh Mohandas, and Alwyn R. Pais

Information Security Research Lab,
Department of Computer Science and Engineering,
National Institute of Technology Karnataka, Surathkal, India
{tanushanadkarni,radhesh,alwyn.pais}@gmail.com

Abstract. Drive-by downloads are currently one of the most popular methods of malware distribution. Widely visited legitimate websites are infused with invisible or barely visible Iframes pointing to malicious URLs, causing silent download malware on users system. In this paper, we present a client side solution for protection from such malevolent hidden Iframes. We have implemented our solution as an extension to Mozilla Firefox browser. The extension will check every Iframe loaded in the browser for properties emblematic of malicious Iframes such as hidden visibility styles and 0-pixel dimensions. These Iframes are then blocked by using browser content policy mechanism, hence alleviating the possibility of the malicious download taking place.

Keywords: Iframes, Malicious JavaScript, Obfuscation, nsIContentPolicy, Drive-by Downloads, Malware, Iframe Injection Attack, Mozilla Firefox Browser, Extension.

1 Introduction

Malicious code is either hosted directly on rogue web sites or injected using Iframes into legitimate web sites. The victim sites are compromised either partially or completely by cyber criminals to serve the malicious content. Hackers bring into play various social engineering techniques to lure users to visit rogue sites. But this requires attackers to trick users to open doors for malware to descend on their systems. To broaden their techniques of distributing malware hackers continuously work on inventing ways of downloading malware without users consent. Most Drive-by downloads happen when a user visits a website that contains a malicious Iframe. The Iframe may load the exploit script, or redirect to another malevolent site[1]. This script targets vulnerabilities in the browser or one of its plugins, successful exploitation of which results in the automatic execution of the spiteful code, triggering a drive-by download. The downloaded executable is then automatically installed and started on the infected system. Attackers use a number of obfuscation techniques to evade detection and complicate forensic analysis. The beginning point of a drive-by download attack is an Iframe, hence the main focus of our work is to utilize the power provided to browser extensions to identify potentially harmful Iframes and prevent them from loading in the browser.

2 Related Work

NoScript [8] an extension for Mozilla's Firefox browser, selectively manages Iframes. Iframes are blocked based on same origin policy and list of trusted sites. In our extension, we look for different properties of Iframes such as dimensions and visibility properties before blocking them.

3 Iframe Injection Attack

The key problem addressed in this paper is protection from malicious Iframes injected in legitimate websites, which are invisible to the visitors of the website. The <Iframe> tag is an HTML element that contains another document. A hidden Iframe is an Iframe which is present in the HTML of a webpage, but is not visible in the browser. Iframe Injection Attack threat injects an invisible Iframe into legitimate websites [2]. The Iframe source will be a malicious URL hosting an exploit or containing code which redirects to a malicious page. Malicious Iframes are made invisible by setting their attributes appropriately, and are usually injected in the footer or header. Invisible Iframes allow silently loading of malware. To evade detection, obfuscated JavaScript is infused into legitimate sites, which after decoding embeds malevolent Iframe. Many fall victims to such attacks as it is easy to inject such malicious Iframe into a legitimate webpages if the hosting server or FTP accounts are compromised or using Cross Site Scripting and SQL Injection attacks.

3.1 Techniques for Hiding Iframe

Iframe are rectangular elements and they occupy some space on web pages. Hackers use several techniques [3], such as dimension tricks, and visibility styles, to make them invisible for the compromised website visitor. Initially, either or both of the dimensions of the Iframes were set to 0. Since scanners searching for zeros, hackers started to use Iframes with very small dimensions, which makes an Iframe appear like a dot. To by pass scanners looking for dimensions, techniques such as setting visibility to 'hidden', or display to 'none', made the Iframes completely unseen, irrespective of their dimensions. The trick was to place visible Iframes within any parent node with hidden styles e.g. invisible <DIV> element. This hid Iframes despite not containing any code which made them invisible. A real example: <Iframe src="hxxp://google-analyz .cn/ count.php?o=1" width=0 height=0 style="visibility: hidden"> </Iframe>. JavaScript onload trick, is another technique, in which, hackers inject Iframe that does not have a src, style and dimension parameters that can make Iframe invisible. A script is specified for the "onload" event of Iframe which assigns values to Iframe's src, height and width on the fly when is its loaded. For example: <Iframe onload="if (!this.src) this.src='hxxp://iqmon .ru:8080/ index.php'; this.height='0'; this.width='0';"> </Iframe>. Execution of the script makes the Iframe hidden, loading a malicious page at the same time.

4 Motivation

Hackers are always in search of vulnerable popular sites. A very recent victim of Iframe injection attack is the Lenovo site. According to reports from security vendor from Zscaler[7], a website of Lenovo India was compromised and injected with a malicious invisible Iframe. This Iframe was redirecting visitors silently to the Incognito exploit kit. The Iframe had zero height and width, and its display property is set to none, making it invisible to the visitors of the website. Other real attack cases in 2011 include the frequently visited Geek.com [7], ICFAI site [7], BBC website [6], Goal.com [5]. Hundreds of such attacks are seen each and every day.

5 IFrandbox - Sandbox for Iframes

Deriving incite from the real examples in the motivation section above, an extension was created for Mozilla Firefox, to assuage the damage caused by Iframe injection attack. This extension will observe every Iframe being loaded, for properties typical of malicious Iframes. If an Iframes falls in the malicious category it will be blocked. When a web page is rendered by browser, all the elements loaded are monitored using nsIContentPolicy XPCOM interface [4], an interface for content policy mechanism. This interface can observe and control content that is being loaded into the browser. When an Iframe is being loaded, using Document Object Model functions, the source and style properties of the Iframe are checked against a set of rules. If the Iframe element passes the check, then it is loaded, else it is blocked. The rules are based on tricks used to hide Iframe elucidated in Iframe Injection Attack section. The rules include checking if Iframe visibility style is hidden or collapse or display style is set to none, if Iframe has zero or negligible dimensions, if any of the parent nodes (till the root node) of Iframe posess an invisibility property that can render the Iframe hidden. In addition, Iframes are also checked for the onload trick. These rules are configured considering various methods practiced by hackers and can be improvised as the hackers improvise. We have provided a framework to add new policies to the extension to keep up with the new methods to carry out attacks that hackers keep innovating.

6 Testing of IFrandbox

The extension was tested against malicious URLs from Malware Domain List [9]. The Iframe attributes are checked when they are loaded and not by statically scanning the source code of a web page. Hence even if the Iframe is injected using obfuscated JavaScript, it does not prevent the attribute check and this is the foremost advantage. For example, <Iframe src=http://www.malsite.com width=0 height=0> </Iframe> when injected using obfuscated JavaScript appears as document.write(unescape("%3CIfr ame%20src%3Dhttp%3A//www.malsite.com %20width%3D0%20height%3D0%3E%3C/Iframe%3E)); The Iframe was encoded using the escape() function and decoded just before being injected into the page using document.write. When this webpage was rendered in the browser this Iframe was blocked by the extension for having zero dimensions. Thus the obfuscation does not prevent the Iframe from being undetected. Even in the case when the onload trick is used, the source is available when Iframe is loading, and the dimensions are retrieved by checking the content of onload trick.

One disadvantage of IFrandbox is that the rules will filter out those sites using invisible Iframes for valid purpose, leading to false positives. But false positives can be alleviated by adding exceptions to the rules. For this purpose, this addon was tested against more than 5000 URLs, to see if it blocked any genuine Iframes. It was observed that among the sites blocked, the most common were Facebook 'like' plugin, Google AdSense program (http://googleads.g.doubleclick.net), http: //ad.doubleclick.net, http://s7.addthis.com/, Twitter button etc. Our extension allows users to manage an exception list.

7 Conclusion

In this paper we have presented a solution to protect users from threat posed by malicious Iframes injected into legitimate websites. An extension to Firefox has been created which will block all Iframes having properties indicating their maliciousness, including those injected using obfuscation techniques. Using the extension will help in preventing Drive-by downloads initiated by malicious invisible Iframes, hence making user's browsing experience more safe and secure.

References

1. Provos, N., Mavrommatis, P., Abu, M., Monros, R.F.: All Your Iframes Point to Us In: Google Technical Report provos-2008a
2. Hidden iframe injection attacks—Diovo, http://diovo.com/2009/03/hidden-iframe-injection-attacks/
3. Evolution of Hidden Iframes—Unmask Parasites Blog, http://blog.unmaskparasites.com/2009/10/28/evolution-of-hidden-iframes/
4. nsIContentPolicy, https://developer.mozilla.org/en/XPCOM_Interface_Reference/nsIContentPolicy
5. Goal.com Riddled with Malware-Serving Code, http://news.softpedia.com/news/Goal-com-Riddled-with-Malware-Serving-Code-198040.shtml
6. Infosecurity, http://www.infosecurity-magazine.com/view/15993/bbc-6-music-and-1xtra-websites-infected-by-phoenix-exploit-kit-hack
7. Zscaler Research, http://research.zscaler.com
8. NoScript, https://addons.mozilla.org/en-US/firefox/addon/noscript/
9. Malware Domain List, http://www.malwaredomainlist.com

Author Index