Lecture Notes in Computer Science 2502
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

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

Dieter Gollmann Günter Karjoth
Michael Waidner (Eds.)

Computer Security – ESORICS 2002

7th European Symposium on Research in Computer Security
Zurich, Switzerland, October 14-16, 2002
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Dieter Gollmann
Microsoft Research
7 J J Thomson Avenue, Cambridge CB3 0FB, UK
E-mail: diego@microsoft.com

Günther Karjoth
Michael Waidner
IBM Zurich Research Lab
Säumerstr. 4, 8803 Rüschlikon, Switzerland
E-mail: {gka/wmi}@zurich.ibm.com

Cataloging-in-Publication Data applied for

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

CR Subject Classification (1998): D.4.5, E.3, C.2.0, H.2.0, K.6.5, K.4.4

ISSN 0302-9743
ISBN 3-540-44345-2 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10870813 06/3142 5 4 3 2 1 0

Preface

ESORICS, the European Symposium on Research in Computer Security, is the leading research-oriented conference on the theory and practice of computer security in Europe. It takes place every two years, at various locations throughout Europe, and is coordinated by an independent Steering Committee.

ESORICS 2002 was jointly organized by the Swiss Federal Institute of Technology (ETH) and the IBM Zurich Research Laboratory, and took place in Zurich, Switzerland, October 14-16, 2002.

The program committee received 83 submissions, originating from 22 countries. For fans of statistics: 55 submissions came from countries in Europe, the Middle East, or Africa, 16 came from Asia, and 12 from North America. The leading countries were USA (11 submissions), Germany (9), France (7), Italy (7), Japan (6), and UK (6). Each submission was reviewed by at least three program committee members or other experts. Each submission coauthored by a program committee member received two additional reviews. The program committee chair and cochair were not allowed to submit papers. The final selection of papers was made at a program committee meeting and resulted in 16 accepted papers. In comparison, ESORICS 2000 received 75 submissions and accepted 19 of them.

The program reflects the full range of security research: we accepted papers on access control, authentication, cryptography, database security, formal methods, intrusion detection, mobile code security, privacy, secure hardware, and secure protocols.

We gratefully acknowledge all authors who submitted papers for their efforts in maintaining the standards of this conference.

It is also my pleasure to thank the members of the program committee, the additional reviewers, and the members of the organization committee for their work and support.

Zurich, October 2002 Michael Waidner

Organization Committee

Endre Bangerter (IBM Research), Dieter Gollmann (Microsoft Research, UK; Publication Chair) Günter Karjoth (IBM Research, Switzerland; General Chair), Jürg Nievergelt (ETH Zurich, Switzerland; General Co-chair), Floris Tschurr (ETH Zurich)

Steering Committee

Joachim Biskup (University of Dortmund, Germany), Frédéric Cuppens (ONERA, France), Yves Deswarte (LAAS-CNRS, France), Gerard Eizenberg (CERT, France), Simon Foley (University College Cork, Ireland), Dieter Gollmann (Microsoft Research, UK), Franz-Peter Heider (debis IT Security Services, Germany), Jeremy Jacob (University of York, UK), Socratis Katsikas (University of the Aegean, Greece), Helmut Kurth (atsec, Germany), Peter Landrock (Cryptomathic, UK), Emilio Montolivo (FUB, Italy), Roger Needham (Microsoft Research, UK), Jean-Jacques Quisquater (UCL, Belgium), Peter Ryan (University of Newcastle, UK: Steering Committee Chair), Pierangela Samarati (University of Milan, Italy), Einar Snekkenes (Norwegian Computer Center, Norway), Michael Waidner (IBM Research, Switzerland).

Table of Contents

Computational Probabilistic Non-interference

Michael Backes[1] and Birgit Pfitzmann[2]

[1] Saarland University, Saarbrücken, Germany
mbackes@cs.uni-sb.de
[2] IBM Zurich Research Laboratory, Rüschlikon, Switzerland
bpf@zurich.ibm.com

Abstract. In recent times information flow and non-interference have become very popular concepts for expressing both integrity and privacy properties. We present the first general definition of probabilistic non-interference in reactive systems which includes a computational case. This case is essential to cope with real cryptography since non-interference properties can usually only be guaranteed if the underlying cryptographic primitives have not been broken. This might happen, but only with negligible probability. Furthermore, our definition links non-interference with the common approach of simulatability that modern cryptography often uses. We show that our definition is maintained under simulatability, which allows secure composition of systems, and we present a general strategy how cryptographic primitives can be included in information flow proofs. As an example we present an abstract specification and a possible implementation of a cryptographic firewall guarding two honest users from their environment.

1 Introduction

Nowadays, information flow and non-interference are known as powerful possibilities for expressing privacy and integrity requirements a program or a cryptographic protocol should fulfill. The first models for information flow have been considered for secure operating systems by Bell and LaPadula [3], and Denning [5]. After that, various models have been proposed that rigorously define when information flow is considered to occur. The first one, named *non-interference*, was introduced by Goguen and Meseguer [6, 7] in order to analyze the security of computer systems, but their work was limited to deterministic systems. Nevertheless, subsequent work was and still is based on their idea of defining information flow. After that, research focused on non-deterministic systems, mainly distinguishing between probabilistic and possibilistic behaviors. Beginning with Sutherland [19] the possibilistic case has been dealt with in [15, 20, 16, 22, 14], while the probabilistic and information-theoretic cases have been analyzed by Gray [9, 10] and McLean [17]. Clark et. al. have shown in [4] that possibilistic information flow analysis can be used to check for probabilistic interference.

Gray's definition of "Probabilistic Non-Interference" of reactive systems stands out. It is closely related to the perfect case of our definition, but it does not cover computational aspects which are essential for reasoning about systems using real cryptographic primitives. Thus, if we want to consider real cryptography we cannot restrict ourselves to perfect non-interference as captured by the definition of Gray (nor to any other definition mentioned before, because they are non-probabilistic and hence not suited to cope with real cryptography) because it will not be sufficient for most cryptographic purposes. As an example, consider an arbitrary public key encryption scheme. Obviously, an adversary with unlimited computing power can always break the scheme by computing all possible encryptions of every plaintext and comparing the results with the given ciphertext. Moreover, even polynomially bounded adversaries may have a very small, so-called *negligible* probability of success. Thus, cryptographic definitions usually state that every polynomially bounded adversary can only achieve its goal with a negligible probability. Adopting this notion we present the first general definition of non-interference for this so-called *computational* case. Besides our work, the paper of Laud [12] contains the only definition of non-interference including such a computational case. However, only encryption is covered so far, i.e., other important concepts like authentication, pseudo-number generators, etc. are not considered. Moreover, the definition is non-reactive, i.e., it does not comprise continuous interaction between the user, the adversary, and the system, which is a severe restriction to the set of considered cryptographic systems. Our definition is reactive and comprises arbitrary cryptographic primitives.

In contrast to other definitions, we will not abstract from cryptographic details and probabilism, e.g., by using the common Dolev-Yao abstraction or special type systems, but we immediately include the computational variant in our definition. This enables sound reduction proofs with respect to the security definitions of the included cryptographic primitives (e.g., reduction proofs against the security of an underlying public key encryption scheme), i.e., a possibility to break the non-interference properties of the system can be used to break the underlying cryptography. Moreover, we show that our definition behaves well under the concept of simulatability that modern cryptography often uses, i.e., non-interference properties proved for an abstract specification automatically carry over to the concrete implementation. This theorem is essential since it enables modular proofs in large systems, i.e., proofs done for ideal systems not containing any probabilism simply carry over to their corresponding real cryptographic counterparts. Moreover, properties of these ideal systems could quite easily be proved machine-aided, so our theorem additionally provides a link between cryptography and formal proof tools for non-interference. Thus, non-interference properties can be expressed for reactive systems containing arbitrary cryptographic primitives, which is of great importance for extensible systems like applets, kernel extensions, mobile agents, virtual private networks, etc.

Outline of the paper. In Section 2 we briefly review the underlying model of asynchronous reactive system introduced in [18]. The original contributions are presented in Sections 3, 4 and 5. In Section 3 we extend the underlying model to multiple users and we refer to as multi-party configurations; built on this definition we construct our definition of non-interference. In Section 4 we show that our definition behaves well under simulatability, hence refinement does not change the non-interference properties. In Section 5 we present an abstract specification and a possible implementation of a cryptographic firewall guarding two honest users from their environment, and we prove that they fulfill our definition of non-interference.

2 General System Model for Reactive Systems

In this section we briefly recapitulate the model for probabilistic reactive systems as introduced in [18]. All details of the model which are not necessary for understanding are omitted; they can be looked up in the original paper.

Systems mainly are compositions of different machines. Usually we consider real systems consisting of a set $\hat{M}$ of machines $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ and ideal systems built by one machine $\{\mathsf{TH}\}$, called *trusted host*. The machine model is probabilistic state-transition machines, similar to I/O automata as introduced in [13]. For complexity we consider every automata to be implemented as a probabilistic Turing machine; complexity is measured in the length of its initial state, i.e. the initial worktape content (often a security parameter k, given in unary representation).

Communication between different machines is done via ports. Inspired by the CSP-Notation [11], we write output and input ports as $\mathsf{p}!$ and $\mathsf{p}?$, respectively. The ports of a machine M will be denoted by $\mathsf{ports}(\mathsf{M})$. Connections are defined implicitly by naming convention, i.e., port $\mathsf{p}!$ sends messages to $\mathsf{p}?$. To achieve asynchronous timing, a message is not directly sent to its recipient, but it is first stored in a special machine $\widetilde{\mathsf{p}}$ called a buffer and waits to be scheduled. If a machine wants to schedule the i-th message of buffer $\widetilde{\mathsf{p}}$ (this machine must have the unique clock-out port $\mathsf{p}^{\triangleleft}!$) it simply sends i at $\mathsf{p}^{\triangleleft}!$. The i-th message is then scheduled by the buffer and removed from its internal list. In our case, most buffers are either scheduled by a specific master scheduler or the adversary, i.e., one of those has the corresponding clock-out port.

A *collection* $\mathcal{C}$ of machines is a finite set of machines with pairwise different machine names and disjoint sets of ports. The *completion* $[\mathcal{C}]$ of a collection $\mathcal{C}$ is the union of all machines of $\mathcal{C}$ and the buffers needed for every channel.

A *structure* is a pair $(\hat{M}, S)$, where $\hat{M}$ is a collection of machines and $S \subseteq \mathsf{free}([\hat{M}])$, the so called *specified ports*, are a subset of the free[1] ports of $[\hat{M}]$. Roughly speaking the ports of S guarantee special services to the users. We will always describe specified ports by their complements S^c, i.e., the ports honest users should have. A structure can be completed to a *configuration* by adding special machines H and A, modeling honest users and the adversary. The machine H is restricted to the specified ports S, A connects to the remaining free ports of the structure and both machines can interact. If we now consider sets of structures we obtain a *system* Sys.

Scheduling of machines is done sequentially, so we have exactly one active machine M at any time. If this machine has clock-out ports, it is allowed to select

[1] A port is called *free* if its corresponding port is not in the system. These port will be connected to the users and to the adversary.

the next message to be scheduled as explained above. If that message exists, it is delivered by the buffer and the unique receiving machine is the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none or the message does not exist, the special master scheduler is scheduled.

Altogether we obtain a runnable system which we refer to as a *configuration* and a probability space over all possible executions (also denoted as *runs* or *traces*) of the system. If we restrict runs to certain sets $\hat{M}$ of machines, we obtain the *view* of $\hat{M}$. Moreover we can restrict a run r to a set $\mathcal{S}$ of ports which is denoted by $r\lceil_{\mathcal{S}}$.

For a configuration *conf*, we furthermore obtain random variables over this probability space which are denoted by $run_{conf,k}$ and $view_{conf,k}$, respectively.

3 Expressing Non-interference and Multi-party Configurations

In this section we define non-interference for reactive systems as introduced in Section 2. At first we look at the more general topic of information flow. Information flow properties consist of two components: a *flow policy* and a *definition of information flow*. Flow policies are built by graphs with two different classes of edges. The first class symbolizes that information may flow between two users, the second class symbolizes that it may not. If we now want to define non-interference, we have to provide a semantics for the second class of edges.[2] Intuitively, we want to express that there is no information flow from a user H_H to a user H_L iff the view of H_L does not change for every possible behaviour of H_H, i.e., H_L should not be able to distinguish arbitrary two families of views induced by two behaviours of H_H. As we have seen in Section 2, we did not regard different honest users as different machines so far, we just combined them into one machine H. Obviously, this distinction is essential for expressing non-interference, so we first have to define multi-party configurations for the underlying model. Multi-party configurations are defined identically to usual configurations except that we have a set U of users instead of a one-user machine H.

3.1 Multi-party Configurations

Definition 1. *(Multi-Party Configurations)* A *multi-party configuration* $conf^{\mathsf{mp}}$ of a system Sys is a tuple $(\hat{M}, S, U, \mathsf{A})$ where $(\hat{M}, S) \in Sys$ is a structure, U is a *set* of machines called users without forbidden ports, i.e., $\mathsf{ports}(U) \cap \mathsf{forb}(\hat{M}, S) = \emptyset$ must hold and the completion $\hat{C} := [\hat{M} \cup U \cup \{\mathsf{A}\}]$ is a closed collection. The set of these configurations will be denoted by $\mathsf{Conf}^{\mathsf{mp}}(Sys)$, those with polynomial-time users and a polynomial-time adversary by $\mathsf{Conf}^{\mathsf{mp}}_{\mathsf{poly}}(Sys)$. We will omit the indices mp and poly if they are clear from the context. ◇

[2] We will not present a semantics for the first class of edges here, because we only focus on absence of information flow in this paper.

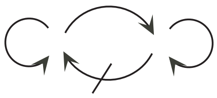


Fig. 1. A typical flow policy graph consisting of high and low users only

It is important to note that runs and views are also defined for multi-party configurations because we demanded the completion $\hat{C}$ to be closed.

3.2 Flow Policies

We start by defining the flow policy graph.

Definition 2. *(General Flow Policy)* A *general flow policy* is a pair $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ with $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{S} \times \{\rightsquigarrow, \not\rightsquigarrow\}$. Thus, we can speak of a graph $\mathcal{G}$ with two different kind of edges: $\rightsquigarrow, \not\rightsquigarrow \subseteq \mathcal{S} \times \mathcal{S}$. Furthermore we demand $(s_1, s_1) \in \rightsquigarrow$ for all $s_1 \in \mathcal{S}$, and every pair (s_1, s_2) of nodes should be linked by exactly one edge, so $\rightsquigarrow$ and $\not\rightsquigarrow$ form a partition of $\mathcal{S} \times \mathcal{S}$. ◇

Remark 1. The set $\mathcal{S}$ often consists of only two elements $\mathcal{S} = \{L, H\}$ which are referred to as low- and high-level users. A typical flow policy would then be given by $L \rightsquigarrow L$, $L \rightsquigarrow H$, $H \rightsquigarrow H$, and finally $H \not\rightsquigarrow L$, cf. Figure 1, so there should not be any information flow from high- to low-level users.

This definition is quite general since it uses an arbitrary set $\mathcal{S}$. If we want to use it for our purpose, we have to refine it so that it can be applied to a system *Sys* of our considered model. The intuition is to define a graph on the possible participants of the protocol, i.e., users and the adversary. However, this definition would depend on certain details of the users and the adversary, e.g., their port names, so we specify users by their corresponding specified ports of *Sys*, and the adversary by the remaining free ports of the system to achieve independency. After that, our flow policy only depends on the ports of *Sys*.

Definition 3. *(Flow Policy)* Let a structure $(\hat{M}, S)$ be given, and let $\Gamma_{(\hat{M},S)} = \{S_i \mid i \in \mathcal{I}\}$ denote a partition of S for a finite index set $\mathcal{I}$, so $\Delta_{(\hat{M},S)} := \Gamma_{(\hat{M},S)} \cup \{\bar{S}\}$ is a partition of $\mathsf{free}([\hat{M}])$. A flow policy $\mathcal{G}_{(\hat{M},S)}$ of the structure $(\hat{M}, S)$ is now defined as a general flow policy $\mathcal{G}_{(\hat{M},S)} = (\Delta_{(\hat{M},S)}, \mathcal{E}_{(\hat{M},S)})$.

The set of all flow policies for a structure $(\hat{M}, S)$ and a partition $\Delta_{(\hat{M},S)}$ of $\mathsf{free}([\hat{M}])$ will be denoted by $POL_{\hat{M},S,\Delta_{(\hat{M},S)}}$. Finally, a flow policy for a system *Sys* is a mapping

$$\begin{aligned} \phi_{Sys} : \quad & Sys \rightarrow POL_{\hat{M},S,\Delta_{(\hat{M},S)}} \\ & (\hat{M}, S) \mapsto \mathcal{G}_{(\hat{M},S)} \end{aligned}$$

that assigns each structure $(\hat{M}, S)$ a flow policy $\mathcal{G}_{(\hat{M},S)}$ which is defined on $(\hat{M}, S)$. ◇

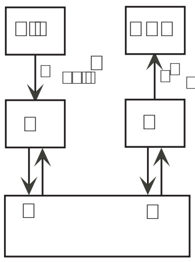


Fig. 2. Sketch of our non-interference definition

We will simply write $\mathcal{G}$, $\varDelta$, and $\mathcal{E}$ instead of $\mathcal{G}_{(\hat{M},S)}$, $\varDelta_{(\hat{M},S)}$, and $\mathcal{E}_{(\hat{M},S)}$ if the underlying structure is clear from the context. Additionally, we usually consider graphs with the following property: for blocks of ports $S_H, S_L \in \varDelta$ with $(S_H, S_L) \in \not\leadsto$ there should not be a path from S_H to S_L consisting of "$\leadsto$"-edges only. We will refer to this property as *transitivity property* and speak of a *transitive flow policy*.

The relation $\not\leadsto$ is the non-interference relation of $\mathcal{G}$, so for two arbitrary blocks $S_H, S_L \in \varDelta$, $(S_H, S_L) \in \not\leadsto$ means that no information flow must occur directed from the machine connected to S_H to the machine connected to S_L. The notion of a transitive flow policy is motivated by our intuition that if a user H_H should not be allowed to directly send any information to user H_L, he should also not be able to send information to H_L by involving additional users, similar for the adversary.

3.3 Definition of Non-interference

We still have to define the semantics of our non-interference relation $\not\leadsto$. Usually, expressing this semantics is the most difficult part of the whole definition. In our underlying model, it is a little bit easier because we already have definitions for runs, views, and indistinguishability that can be used to express the desired semantics.

Figure 2 contains a sketch of our definition of non-interference between two users H_H and H_L (one of them might also be the adversary). Mainly, we define a specific machine BIT_H, that simply chooses a bit $b \in \{0,1\}$ at random and outputs it to H_H. Non-interference now means that H_H should not be able to change the view of H_L, so it should be impossible for H_L to output the bit b at $\mathsf{p}^*_{L\_\mathsf{bit}}!$ with a probability better than $\frac{1}{2}$ in case of perfect non-interference. Statistical and computational non-interference now means that the advantage of H_L for a correct guess of b should be a function of a class *SMALL* or negligible, respectively, measured in the given security parameter k. The approach of "guessing a bit", i.e., including the machines BIT_H and OUT_L in our case, is essential to extend the notation of probabilistic non-interference to error probabilities and complexity-theoretic assumptions. Moreover, it is a fundamental concept in cryptography, so our definition additionally serves as a link between prior work in non-interference and real cryptographic primitives along with their security definitions.

These specific configurations including the special BIT_H- and OUT_L-machines will be called *non-interference configurations*. Before we turn our attention to the formal definition of these configurations we briefly describe which machines have to be included, how they behave, and which ports are essential for these sort of configurations, cf. Figure 3.

First of all, we have special machines BIT_H, OUT_L and $\mathsf{X}^{\mathsf{n\text{-}in}}$. As described above, BIT_H will uniformly choose a bit at the start of the run and output it to the user H_H, the second machine simply catches the messages received at port $\mathsf{p}^*_{L\_\mathsf{bit}}?$.

The machine $\mathsf{X}^{\mathsf{n\text{-}in}}$ is the master scheduler of the configuration. Its function is to provide liveness properties and to avoid denial of service attacks, so it ensures that every machine will be able to send messages if it wants to. Before we turn our attention to this machine, we briefly describe the ports of the users. In order to improve readability we encourage the reader to compare these descriptions with Figure 3.

At first, we demand that every user must not have any clock-out ports; instead they have additional output ports $\mathsf{p}^{\mathsf{s}}!$ connected to the master scheduler for every "usual" output port $\mathsf{p}!$. The master scheduler will use these ports to schedule outputs from ports $\mathsf{p}!$, so a user can tell the master scheduler which port it wants to be scheduled. This is essential for achieving liveness properties, because otherwise, there might be cycles inside of the system, so that neither the master scheduler nor some users will ever be scheduled. Therefore, we explicitly give the control to the master scheduler and define a suitable scheduling strategy in the formal definition.

We now focus on the ports of the master scheduler. First of all, it has the corresponding input ports $\mathsf{p}^{\mathsf{s}}?$ for receiving scheduling demands from the users, and the corresponding clock-out ports $\mathsf{p}^{\mathsf{s}\triangleleft}!$. Furthermore, it has clock-out ports $\mathsf{p}^{\triangleleft}!$ to schedule every buffer $\widetilde{\mathsf{p}}$ that delivers messages from the users to the actual system. Finally, it has special ports $\mathsf{master}_i!$ to schedule (or to give control to) the user H_i. The actual scheduling process, i.e., how and in what order users are scheduled will be described in the formal definition.

Definition 4. *(Non-Interference Configuration)* Let a finite index set $\mathcal{I}$ with $\mathsf{A} \notin \mathcal{I}$ and $H, L \in \mathcal{I} \cup \{\mathsf{A}\}$, $H \neq L$ be given. Furthermore, let a multi-party configuration $conf^{\mathsf{mp}}_{H,L} = (\hat{M}, S, U \cup \{\mathsf{BIT}_H, \mathsf{OUT}_L, \mathsf{X}^{\mathsf{n\text{-}in}}\}, \mathsf{A})$ of a system Sys with $U = \{\mathsf{H}_i \mid i \in \mathcal{I}\}$ and a partition $\Delta = \{S_i \mid i \in \mathcal{I}\} \cup \{\bar{S}\}$ of $\mathsf{free}([\hat{M}])$ be given. For naming convention we set $\mathsf{H}_\mathsf{A} := \mathsf{A}$ and $S_\mathsf{A} := \bar{S}$. We call this configuration a non-interference configuration of Sys if the following holds:

a) The ports of BIT_H are given by $\{\mathsf{master}_{\mathsf{BIT}_H}?, \mathsf{p}_{H\_\mathsf{bit}}!, \mathsf{p}_{H\_\mathsf{bit}}{}^{\triangleleft}!\}$.
The specific machine OUT_L has only one input port $\mathsf{p}^*_{L\_\mathsf{bit}}?$ connected to H_L.
The machine $\mathsf{X}^{\mathsf{n\text{-}in}}$ is the master scheduler of the configuration. Its ports are given by
   - $\{\mathsf{clk}^{\triangleleft}?\}$: The master clock-in port.

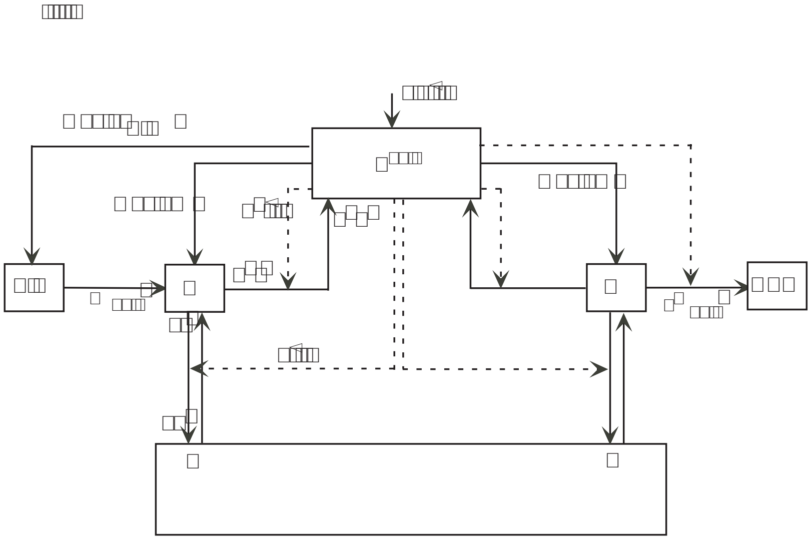


Fig. 3. Main Parts of a non-interference configuration. Additionally, the ports of the master scheduler $\mathsf{X}^{\mathsf{n\text{-}in}}$ and the two emphasized users H_H and H_L are sketched

- $\{\mathsf{p}^{\triangleleft}! \mid \mathsf{p}^{\triangleleft}! \in S_i^c, i \in \mathcal{I}\}$: The ports for scheduling buffers between users and the actual system.
- $\{\mathsf{p}^{\mathsf{s}}?, \mathsf{p}^{\mathsf{s}\triangleleft}! \mid \mathsf{p}^{\triangleleft}! \in S_i^c, i \in \mathcal{I}\}$: The ports connected to the users for receiving scheduling demands.
- $\{\mathsf{p}^{\mathsf{s}*}_{L\_\mathsf{bit}}?, {\mathsf{p}^{\mathsf{s}*}_{L\_\mathsf{bit}}}^{\triangleleft}!, {\mathsf{p}^{*}_{L\_\mathsf{bit}}}^{\triangleleft}!\}$: The ports for scheduling demands and outputs to machine OUT_L.
- $\{\mathsf{master}_i!, {\mathsf{master}_i}^{\triangleleft}! \mid i \in \mathcal{I} \cup \{\mathsf{A}\} \cup \{\mathsf{BIT}_H\}\}$: The ports for scheduling (that is giving control to) the users, the adversary and BIT_H.

b) For $i \in \mathcal{I}$ the ports of H_i must include $\{\mathsf{p}^{\mathsf{s}}! \mid \mathsf{p}^{\triangleleft}! \in S_i^c\} \cup \{\mathsf{master}_i?\}$, the ports of the adversary must include $\mathsf{master}_{\mathsf{A}}?$. Additionally, H_H must have an input port $\mathsf{p}_{H\_\mathsf{bit}}?$, H_L must have output ports $\mathsf{p}^*_{L\_\mathsf{bit}}!$ and $\mathsf{p}^{\mathsf{s}*}_{L\_\mathsf{bit}}!$. Furthermore, we demand that the remaining ports of every user and of the adversary connect exactly to their intended subset of specified ports. Formally,

$$\mathsf{ports}(\mathsf{H}_H) \setminus (\{\mathsf{p}_{H\_\mathsf{bit}}?\} \cup \{\mathsf{p}^{\mathsf{s}}! \mid \mathsf{p}^{\triangleleft}! \in S_H^c\} \cup \{\mathsf{master}_{\mathsf{H}}?\}) = S_H^c \setminus \{\mathsf{p}^{\triangleleft}! \mid \mathsf{p}^{\triangleleft}! \in S_H^c\},$$

$$\begin{aligned}&\mathsf{ports}(\mathsf{H}_L) \setminus (\{\mathsf{p}^*_{L\_\mathsf{bit}}!, \mathsf{p}^{\mathsf{s}*}_{L\_\mathsf{bit}}!\} \cup \{\mathsf{p}^{\mathsf{s}}! \mid \mathsf{p}^{\triangleleft}! \in S_L^c\} \cup \{\mathsf{master}_{\mathsf{L}}?\})\\ &= S_L^c \setminus \{\mathsf{p}^{\triangleleft}! \mid \mathsf{p}^{\triangleleft}! \in S_L^c\}\end{aligned}$$

must hold, respectively.
For the remaining users H_i with $i \in \mathcal{I} \cup \{\mathsf{A}\}$, $i \notin \{H, L\}$ we simply have to leave out the special bit-ports, i.e., the equation

$$\mathsf{ports}(\mathsf{H}_i) \setminus (\{\mathsf{p}^{\mathsf{s}}! \mid \mathsf{p}^{\triangleleft}! \in S_i^c\} \cup \{\mathsf{master}_i?\}) = S_i^c \setminus \{\mathsf{p}^{\triangleleft}! \mid \mathsf{p}^{\triangleleft}! \in S_i^c\}$$

must hold. Essentially this means that an arbitrary user H_i must not have any clock-out ports and its "usual" simple ports are connected exactly to the simple ports of S_i. The special ports $\mathsf{p}_{H\_\mathsf{bit}}?$, $\mathsf{p}^*_{L\_\mathsf{bit}}!$, $\mathsf{p^s}^*_{L\_\mathsf{bit}}!$ and ports of the form $\mathsf{master}_i?$, $\mathsf{p^s}!$ are excluded because they must be connected to the master scheduler and to the machines BIT_H and OUT_L.

If the adversary is one of the two emphasized users, i.e., $\mathsf{A} \in \{H, L\}$, we have to leave out the term $\{\mathsf{p^s}! \mid \mathsf{p}^{\triangleleft}! \in S^c_H\}$, or $\{\mathsf{p^s}! \mid \mathsf{p}^{\triangleleft}! \in S^c_L\}$, respectively. Alternatively, we can wlog. restrict our attention to those adversaries that do not have such port names which can easily be achieved by consistent port renaming.

c) The behaviour of the machine BIT_H is defined as follows. If BIT_H receives an arbitrary input at $\mathsf{master}_{\mathsf{BIT}_H}?$, it chooses a bit $b \in \{0, 1\}$ at random, outputs it at $\mathsf{p}_{H\_\mathsf{bit}}!$, and schedules it.

The machine OUT_L simply does nothing on inputs at $\mathsf{p}^*_{L\_\mathsf{bit}}?$. It just "catches" the inputs to close the collection. This ensures that runs and views of the configuration are still defined without making any changes.

d) The behaviour of the machine $\mathsf{X}^{\mathsf{n\text{-}in}}$ is defined as follows. Internally, it maintains two flags start and fl over $\{0, 1\}$, both initialized with 0, and a counter cnt over the finite index set $\mathcal{I} \cup \{\mathsf{A}\}$. Without loss of generality we assume $\mathcal{I} := \{1, \ldots, n\}$, so the counter is defined over $\{0, \ldots, n\}$, initialized with 0 (identifying the number 0 with A). Additionally, it has a counter $MaSc\_poly$ if the machine is demanded to be polynomial time, furthermore, a polynomial P must be given in this case that bounds the steps of $\mathsf{X}^{\mathsf{n\text{-}in}}$. If $\mathsf{X}^{\mathsf{n\text{-}in}}$ is scheduled, it behaves as follows:

Case 1: Start of the run. If $\mathsf{start} = 0$: Set $\mathsf{start} := 1$ and output 1 at $\mathsf{master}_{\mathsf{BIT}_H}!$, 1 at $\mathsf{master}_{\mathsf{BIT}_H}{}^{\triangleleft}!$.

Case 2: Schedule users. If $\mathsf{fl} = 0$ and $\mathsf{start} = 1$: If $\mathsf{X}^{\mathsf{n\text{-}in}}$ has to be polynomial time, it first checks $cnt = n$, increasing $MaSc\_poly$ in this case and checking whether $MaSc\_poly < P(k)$ holds for the security parameter k, stopping at failure. Now, it sets $cnt := cnt{+}1 \bmod (n{+}1)$, and outputs 1 at $\mathsf{master}_{cnt}!$, 1 at $\mathsf{master}_{cnt}{}^{\triangleleft}!$. If $cnt \neq 0$, i.e., the clocked machine is an honest user, it additionally sets $\mathsf{fl} := 1$ to handle the scheduling demands of this user at its next clocking.

Case 3: Handling scheduling demands. If $\mathsf{fl} = 1$ and $\mathsf{start} = 1$: In this case it outputs 1 at every port $\mathsf{p}^{\mathsf{s}\triangleleft}!$ with $\mathsf{p}^{\triangleleft}! \in S^c_{cnt}$ (for $cnt = L$ it also outputs 1 at $\mathsf{p^s}^*_{L\_\mathsf{bit}}{}^{\triangleleft}!$) and tests whether it gets a non-empty input at exactly one input port.[3] If this does not hold, it sets $\mathsf{fl} := 0$ and does nothing. Otherwise, let $\mathsf{p^s}?$ denote this port with non-empty input i. $\mathsf{X}^{\mathsf{n\text{-}in}}$ then outputs i at $\mathsf{p}^{\triangleleft}!$ and sets $\mathsf{fl} := 0$. This case corresponds to a valid scheduling demand of the user, so the corresponding buffer is in fact scheduled.

[3] More formally, it enumerate the ports and sends 1 at the first one. The buffer either schedules a message to $\mathsf{X}^{\mathsf{n\text{-}in}}$ or it does nothing. In both cases $\mathsf{X}^{\mathsf{n\text{-}in}}$ is scheduled again, so it can send 1 at the second clock-out port and so on. Every received message is stored in an internal array so the test can easily be applied.

We obtain a "rotating" clocking scheme on the set $\mathcal{I} \cup \{\mathsf{A}\}$, so every user and the adversary will be clocked equally often with respect to the special master-ports.

Non-interference configurations are denoted by $conf^{n\_in}_{H,L,\mathcal{I}} = (\hat{M}, S, U^{n\_in}, \mathsf{A})^{n\_in}_{\mathcal{I}}$ with $U^{n\_in} := U \cup \{\mathsf{BIT}_H, \mathsf{OUT}_L, \mathsf{X}^{\mathsf{n\_in}}\}$ but we will usually omit the index $\mathcal{I}$. $conf^{n\_in}_{H,L}$ is called polynomial-time if its underlying multi-party configuration $conf^{\mathsf{mp}}_{H,L}$ is polynomial-time. The set of all non-interference configurations of a system Sys for fixed H, L, and $\mathcal{I}$ will be denoted by $\mathsf{Conf}^{n\_in}_{H,L,\mathcal{I}}(Sys)$, and the set of all polynomial-time non-interference configurations by $\mathsf{Conf}^{n\_in}_{H,L,\mathcal{I},\mathsf{poly}}(Sys)$. $\diamond$

Definition 5. *(Non-Interference)* Let a flow policy $\mathcal{G} = (\Delta, \mathcal{E})$ for a structure $(\hat{M}, S)$ be given. Given two arbitrary elements $H, L \in \mathcal{I} \cup \{\mathsf{A}\}$, $H \neq L$ with $(S_H, S_L) \in \not\rightsquigarrow$, we say that $(\hat{M}, S)$ fulfills the non-interference requirement $NIReq_{H,L,\mathcal{G}}$

a) **perfectly** (written $(\hat{M}, S) \models_{\mathsf{perf}} NIReq_{H,L,\mathcal{G}}$) iff for any non-interference configuration $conf^{n\_in}_{H,L} \in \mathsf{Conf}^{n\_in}_{H,L,\mathcal{I}}(Sys)$ of this structure the inequality

$$P(b = b^* \mid r \leftarrow run_{conf^{n\_in}_{H,L},k}; b := r\lceil_{\mathsf{p}_{H\_\mathsf{bit}}!}; b^* := r\lceil_{\mathsf{p}^*_{L\_\mathsf{bit}}?}) \leq \frac{1}{2}$$

holds.

b) **statistically** for a class $SMALL$ ($(\hat{M}, S) \models_{SMALL} NIReq_{H,L,\mathcal{G}}$) iff for any non-interference configuration $conf^{n\_in}_{H,L} \in \mathsf{Conf}^{n\_in}_{H,L,\mathcal{I}}(Sys)$ of this structure there is a function $s \in SMALL$ such that that the inequality

$$P(b = b^* \mid r \leftarrow run_{conf^{n\_in}_{H,L},k}; b := r\lceil_{\mathsf{p}_{H\_\mathsf{bit}}!}; b^* := r\lceil_{\mathsf{p}^*_{L\_\mathsf{bit}}?}) \leq \frac{1}{2} + s(k)$$

holds. $SMALL$ must be closed under addition and with a function g also contain every function $g' \leq g$.

c) **computationally** ($(\hat{M}, S) \models_{\mathsf{poly}} NIReq_{H,L,\mathcal{G}}$) iff for any polynomial-time non-interference configuration $conf^{n\_in}_{H,L} \in \mathsf{Conf}^{n\_in}_{H,L,\mathcal{I},\mathsf{poly}}(Sys)$ the inequality

$$P(b = b^* \mid r \leftarrow run_{conf^{n\_in}_{H,L},k}; b := r\lceil_{\mathsf{p}_{H\_\mathsf{bit}}!}; b^* := r\lceil_{\mathsf{p}^*_{L\_\mathsf{bit}}?}) \leq \frac{1}{2} + \frac{1}{\mathsf{poly}(k)}$$

holds.

We write "$\models$" if we want to treat all cases together.

If a structure fulfills all non-interference requirements $NIReq_{H,L,\mathcal{G}}$ with $(S_H, S_L) \in \not\rightsquigarrow$, we say it fulfills the (global) requirement $NIReq_{\mathcal{G}}$ ($(\hat{M}, S) \models NIReq_{\mathcal{G}}$). A system Sys fulfills a flow policy ϕ_{Sys} if every structure $(\hat{M}, S) \in Sys$ fulfills its requirement $NIReq_{\phi_{Sys}(\hat{M},S)}$, and we consequently write $Sys \models NIReq_{\phi_{Sys}(\hat{M},S)}$, or $Sys \models \phi_{Sys}$ for short.

$\diamond$

4 Preservation of Non-interference under Simulatability

Simulatability essentially means that whatever might happen to an honest user H in a real system Sys_{real} can also happen to the same honest user in an ideal system Sys_{id}. Formally speaking, for every configuration $conf_1$ of Sys_{real} there is a configuration $conf_2$ of Sys_{id} with the same users yielding indistinguishable views of H in both systems [21]. We abbreviate this by $Sys_{\mathsf{real}} \geq_{\mathsf{sec}} Sys_{\mathsf{id}}$ (Sys_{real} is "at least as secure as" Sys_{id}), indistinguishability of the views of H is denoted by $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$. Usually only certain "corresponding" structures $(\hat{M}_1, S_1)$ of Sys_{real} and $(\hat{M}_2, S_2)$ of Sys_{id} are compared. Structures are called corresponding or *validly mapped* if their set of specified ports are equal. In this section we show that our definition of non-interference behaves well under simulatability. More precisely, we will show that the relation "at least as secure as" will not change the non-interference relation between two arbitrary users (one of them might also be the adversary). Usually, defining a cryptographic system starts with an abstract specification of what the system actually should do, possible implementations have to be proven to be at least as secure as this specification. Such a specification usually consists of a monolithic idealized machine that neither contains any cryptographic details nor any probabilism. Thus, it can be validated quite easily by formal proof systems, e.g., formal theorem proving or even automatic model checking, at least if it is not to complex. Hence, it would be of great use to also verify integrity and privacy properties for this idealized machine that will automatically carry over to the real system.

Our theorem states that non-interference properties are in fact preserved under the relation "at least as secure as". In the proof of the preservation theorem, the following lemma will be needed.

Lemma 1. *The statistical distance $\Delta(\phi(\mathsf{var}_k), \phi(\mathsf{var}'_k))$ for function ϕ of random variables is at most $\Delta(\mathsf{var}_k, \mathsf{var}'_k)$.* □

This is a well-known fact; a proof can be found in, e.g., [8].

Theorem 1. *(Preservation of Non-Interference Properties) Let a flow policy ϕ_{Sys_2} for a system Sys_2 be given, so that $Sys_2 \models \phi_{Sys_2}$ holds. Furthermore, let a system Sys_1 be given with $Sys_1 \geq^f Sys_2$ for a mapping f with $S_1 = S_2$ whenever $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$. Then $Sys_1 \models \phi_{Sys_1}$ for all ϕ_{Sys_1} with $\phi_{Sys_1}(\hat{M}_1, S_1) := \phi_{Sys_2}(\hat{M}_2, S_2)$ for an arbitrary structure $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$. This holds for the perfect, statistical, and the computational case.* □

Proof. We first show that ϕ_{Sys_1} is a well-defined flow policy for Sys_1 under our preconditions. Let an arbitrary structure $(\hat{M}_1, S_1) \in Sys_1$ be given. Simulatability implies that for every structure $(\hat{M}_1, S_1) \in Sys_1$, there exists $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$.

ϕ_{Sys_2} is a flow policy for Sys_2, so we have a flow policy $\mathcal{G}_{(\hat{M}_2, S_2)} = (\Delta, \mathcal{E})$ for $(\hat{M}_2, S_2)$. Furthermore, we have $S_1 = S_2$ by precondition, so we can indeed build the same set Γ of blocks on the specified ports and therefore the same partition

Δ of the free ports of $[\hat{M}_1]$.[4] Hence, $\phi_{Sys_1}(\hat{M}_1, S_1)$ is defined on $(\hat{M}_1, S_1)$, so ϕ_{Sys_1} is a well-defined flow policy for Sys_1.

We now have show that Sys_1 fulfills ϕ_{Sys_1}. Let a structure $(\hat{M}_1, S_1) \in Sys_1$ and two elements $H, L \in \mathcal{I} \cup \{\mathsf{A}\}$, $H \neq L$ with $(S_H, S_L) \in \not\rightsquigarrow$ (with respect to the flow policy $\phi_{Sys_1}(\hat{M}_1, S_1)$) be given. We have to show that $(\hat{M}_1, S_1)$ fulfills the non-interference requirement $NIReq_{H,L,\mathcal{G}}$.

Let now a non-interference configuration $conf^{n\text{-}in}_{H,L,1} = (\hat{M}_1, S_1, U^{n\text{-}in}, \mathsf{A})^{n\text{-}in} \in \mathsf{Conf}^{n\text{-}in}_{H,L,\mathcal{I}}(Sys_1)$ be given. Because of $Sys_1 \geq^f Sys_2$ there is a configuration $conf_{H,L,2} = (\hat{M}_2, S_2, U^{n\text{-}in}, \mathsf{A}') \in \mathsf{Conf}^{n\text{-}in}_{H,L,\mathcal{I}}(Sys_2)$ for $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$ with $view_{conf^{n\text{-}in}_{H,L,1}}(U^{n\text{-}in}) \approx view_{conf_{H,L,2}}(U^{n\text{-}in})$. Moreover, the honest users $U^{n\text{-}in}$ are unchanged by simulatability, so $conf_{H,L,2}$ is again a non-interference configuration; hence, we write $conf^{n\text{-}in}_{H,L,2}$ in the following instead of $conf_{H,L,2}$. As usual we distinguish between the perfect, statistical, and the computational case. In the computational case, both configurations must be polynomial-time.

In the perfect case, we have $view_{conf^{n\text{-}in}_{H,L,1}}(U^{n\text{-}in}) = view_{conf^{n\text{-}in}_{H,L,2}}(U^{n\text{-}in})$ because of $Sys_1 \geq^f_{\mathsf{perf}} Sys_2$. Now, both $b := r\lceil_{\mathsf{p}_{H\_\mathsf{bit}}!}$ and $b^* := r\lceil_{\mathsf{p}^*_{L\_\mathsf{bit}}?}$ are part of the view of $U^{n\text{-}in}$ because BIT_H and OUT_L are elements of $U^{n\text{-}in}$, so we obtain the same probabilities in both configurations. Our precondition $(\hat{M}_2, S_2) \models_{\mathsf{perf}} NIReq_{H,L,\mathcal{G}}$ and our arbitrary choice of $conf^{n\text{-}in}_{H,L,1}$ implies that $(\hat{M}_1, S_1)$ also fulfills $NIReq_{H,L,\mathcal{G}}$.

We will treat the statistical and the computational case together. In the statistical (computational) case we have $view_{conf^{n\text{-}in}_{H,L,1}}(U^{n\text{-}in}) \approx_{SMALL} view_{conf^{n\text{-}in}_{H,L,2}}(U^{n\text{-}in})$ $(view_{conf^{n\text{-}in}_{H,L,1}}(U^{n\text{-}in}) \approx_{\mathsf{poly}} view_{conf^{n\text{-}in}_{H,L,2}}(U^{n\text{-}in}))$. We assume for contradiction that $(\hat{M}_1, S_1)$ does not fulfill the non-interference requirement $NIReq_{H,L,\mathcal{G}}$, so the probability $p(k)$ of a correct guess for $b = b^*$ is not smaller than $\frac{1}{2} + s(k)$ for any $s \in SMALL$ in the statistical case (or $p(k) - \frac{1}{2}$ is not negligible in the computational case). Thus, the advantage $\epsilon(k) := p(k) - \frac{1}{2}$ of the adversary is not contained in $SMALL$ (or $\epsilon(k)$ is not negligible). $(\hat{M}_2, S_2)$ fulfills the non-interference requirement, so in this configuration, the advantage $\epsilon'(k)$ for a correct guess is a function of $SMALL$ in the statistical or negligible in the computational case.

We can then define a distinguisher D as follows. Given the views of $U^{n\text{-}in}$ in both configurations it explicitly knows the views of BIT_H and OUT_L. Now D outputs 1 if $b = b^*$ and 0 otherwise. Its advantage in distinguishing is

$$|P(\mathsf{D}(1^k, view_{conf^{n\text{-}in}_{H,L,1},k}(U^{n\text{-}in})) = 1) - P(\mathsf{D}(1^k, view_{conf^{n\text{-}in}_{H,L,2},k}(U^{n\text{-}in})) = 1)|$$
$$= |\frac{1}{2} + \epsilon(k) - (\frac{1}{2} + \epsilon'(k))| = \epsilon(k) - \epsilon'(k).$$

[4] More, precisely the block $\bar{S}_1$ is identified with $\bar{S}_2$. The ports of both sets may be different, but this does not matter because our definition of flow policies only uses whole blocks, so the different ports do not cause any trouble.

For the polynomial case, this immediately contradicts our assumption $Sys_1 \geq^f_{\mathsf{poly}} Sys_2$ because $\epsilon(k) - \epsilon'(k)$ is not negligible. For the statistical case, the distinguisher D can be seen as a function on the random variables, so Lemma 1 implies

$$\begin{aligned}&\Delta(view_{conf^{n\_in}_{H,L,1},k}(U^{n\_in}), view_{conf^{n\_in}_{H,L,2},k}(U^{n\_in}))\\ \geq\ & |P(\mathsf{D}(1^k, view_{conf^{n\_in}_{H,L,1},k}(U^{n\_in})) = 1) - P(\mathsf{D}(1^k, view_{conf^{n\_in}_{H,L,2},k}(U^{n\_in})) = 1)|\\ =\ & \epsilon(k) - \epsilon'(k).\end{aligned}$$

But $\epsilon(k) - \epsilon'(k) \notin SMALL$ must hold, because $\epsilon'(k) \in SMALL$ and $SMALL$ is closed under addition. Thus, $\Delta(view_{conf^{n\_in}_{H,L,1},k}(U^{n\_in}), view_{conf^{n\_in}_{H,L,2},k}(U^{n\_in})) \notin SMALL$ because $SMALL$ is closed under making functions smaller which yields the desired contradiction. $(S_H, S_L) \in \not\leadsto$ and $(\hat{M}_1, S_1)$ have been chosen arbitrary so $(\hat{M}_1, S_1) \models NIReq_{\mathcal{G}}$ and finally $Sys_1 \models \phi_{Sys_1}$ which finishes the proof. ■

5 A Cryptographic Firewall

In the following we present an example of a system that allows authenticated communications between two users and furthermore ensures that these two users cannot be affected by their environment. This yields a flow policy our system has to (and indeed will) fulfill.

5.1 Some Preliminaries

We start with a brief review on standard cryptographic systems and composition, cf. [18] for more details. In cryptographic protocols every user u usually has exactly one machine M_u and its machine is correct if and only if the user is honest.

The machine M_u of user u has special ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ for connecting to user u. A standard cryptographic system Sys can now be derived by a *trust model*. The trust model consists of an access structure $\mathcal{ACC}$ and a channel model χ. $\mathcal{ACC}$ is a set of subsets $\mathcal{H}$ of $\{1, \ldots, n\}$ and denotes the possible sets of correct machines. For each set $\mathcal{H}$ there will be exactly one structure built by the machines belonging to the set $\mathcal{H}$. The channel model classifies every connection as either secure (private and authentic), authenticated or insecure. In the considered model these changes can easily be achieved via port renaming (see [18]).

For a fixed set $\mathcal{H}$ and a fixed channel model we obtain modified machines for every machine M_u which we refer to as $\mathsf{M}_{u,\mathcal{H}}$. We denote their combination by $\hat{M}_{\mathcal{H}}$ (i.e., $\hat{M}_{\mathcal{H}} := \{\mathsf{M}_{u,\mathcal{H}} \mid u \in \mathcal{H}\}$), so real systems are given by $Sys_{\mathsf{real}} = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$. Ideal systems typically are of the form $Sys_{\mathsf{id}} = \{(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ with the same sets $S_{\mathcal{H}}$ as in the corresponding real system Sys_{real}.

We now briefly review what has already been proven about composition of reactive systems. What we actually want is the relation "at least as secure as" to be consistent with the composition of systems. Assume that we have already

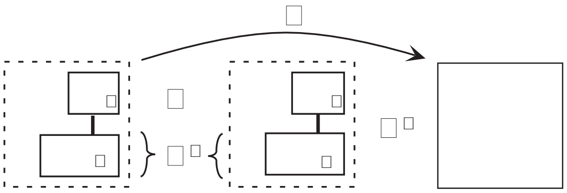


Fig. 4. Composition of systems

proven that a system Sys_0 is at least as secure as another system Sys'_0. Typically, Sys_0 is a real system whereas Sys'_0 is an ideal specification of the real system. If we now consider larger protocols that use Sys'_0 as an ideal primitive we would like to be able to securely replace it with Sys_0. In practice this means that we replace the specification of a system with its implementation.

Usually, replacing means we have another system Sys_1 using Sys'_0; we call this combination Sys^*. We now want to replace Sys'_0 with Sys_0 inside of Sys^* which gives a combination $Sys^\#$. Typically, $Sys^\#$ is a completely real system whereas Sys^* is at least partly ideal. This fact is illustrated in the left and middle part of Figure 4. The composition theorem now states that this replacement maintains security, i.e., $Sys^\#$ is at least as secure as Sys^* (see [18] for further details).

After this brief review we can turn our attention to the cryptographic firewall. The construction of both our ideal and our real system can be explained using Figure 4. Our ideal specification is based on an ideal specification for secure message transmission with ordered channels introduced in [2] which we will slightly modify to fit our requests. Mainly, we have to model reliable channels to avoid denial of service attacks. We will denote this modified ideal system by Sys'_0 following the notation of Figure 4. Furthermore, a possible implementation has also been presented in [2] which we have to modify similar to the ideal specification to maintain the *at least as secure as* relation.

Our cryptographic firewall will then be derived by defining a new system Sys_1 so that combination with Sys'_0 yields the ideal system, replacing Sys'_0 with Sys_0 finally yields a possible implementation. Sys_1 will be designed to filter messages sent by "wrong" senders that should not be allowed to influence the special users according to the flow policy shown in Figure 5. According to Figure 4, we denote our ideal system as Sys^* and our real implementation as $Sys^\#$.

We start with a brief review of the ideal system for secure message transmission with ordered channels and present our modifications of the system afterwards which will be essential for achieving non-interference. After that, we introduce our system Sys_1, and briefly sketch the concrete implementation. We then prove that our ideal system Sys^* fulfills its non-interference requirements. Finally, we apply our preservation theorem 1 and conclude that these non-interference requirements carry over to the concrete implementation, which successfully finishes our attempt to design a real example that fits our non-interference definition.

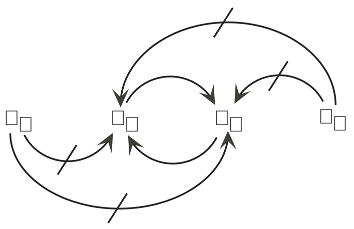


Fig. 5. Sketch of the flow policy of our system. Only one non-emphasized user S_1 is considered and some edges of the graph are omitted. Missing edges are of the form "$\rightsquigarrow$"

5.2 The Ideal System

Let n denote the number of participants, $\mathcal{I} := \{1, \dots, n\}$ the set of indices of the considered participants, and $\mathcal{I}_{\mathsf{A}} := \mathcal{I} \cup \{\mathsf{A}\}$ the set of participants including the adversary. In the following we will identify these indices with their corresponding user. Intuitively, we want a system that fulfills the flow policy shown in Figure 5. We consider two distinguished users a and b with $\{a, b\} \in \mathcal{I}$. We now have two blocks of specified ports S_a and S_b, so that information must not flow to one of these ports from the outside. More precisely, we have non-interference requirements $\mathit{NIReq}_{i_1,i_2,\mathcal{G}}$ for every pair (i_1, i_2) with $i_1 \in \mathcal{I}_{\mathsf{A}} \setminus \{a, b\}$, $i_2 \in \{a, b\}$.

We start with a brief description of the ideal specification for secure message transmission with ordered channels. The specification is of the typical form $\mathit{Sys}'_0 = \{(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \subseteq \mathcal{M}\}$, i.e., there is one structure for every subset of the machines, denoting the honest users.

The ideal machine $\mathsf{TH}_{\mathcal{H}}$ models initialization, sending and receiving of messages. A user u can initialize communications with other users by inputting a command of the form $(\mathsf{snd\_init})$ to the port $\mathsf{in}_u?$ of $\mathsf{TH}_{\mathcal{H}}$. In real systems, initialization corresponds to key generation and authenticated key exchange. Sending of messages to a user v is triggered by a command (send, m, v). If v is honest, the message is stored in an internal array $\mathit{deliver}^{\mathsf{spec}}_{u,v}$ of $\mathsf{TH}_{\mathcal{H}}$ together with a counter indicating the number of the message. After that, a command $(\mathsf{send\_blindly}, i, l, v)$ is output to the adversary, l and i denote the length of the message m and its position in the array, respectively. This models that the adversary will notice in the real world that a message has been sent and he might also be able to know the length of that message. We speak of tolerable imperfections that are explicitly given to the adversary. Because of the underlying asynchronous timing model, $\mathsf{TH}_{\mathcal{H}}$ has to wait for a special term $(\mathsf{receive\_blindly}, v, i)$ or $(\mathsf{rec\_init}, u)$ sent by the adversary, signaling, that the message stored at the ith position of $\mathit{deliver}^{\mathsf{spec}}_{u,v}$ should be delivered to v, or that a connection between u and v should be initialized. In the first case, $\mathsf{TH}_{\mathcal{H}}$ reads $(m, j) := \mathit{deliver}^{\mathsf{spec}}_{u,v}[i]$ and checks whether $\mathit{msg\_out}^{\mathsf{spec}}_{u,v} \leq j$ holds for a message counter $\mathit{msg\_out}^{\mathsf{spec}}_{u,v}$. If the test is successful the message is delivered and the counter is set to $j + 1$, otherwise $\mathsf{TH}_{\mathcal{H}}$ outputs nothing. The condition $\mathit{msg\_out}^{\mathsf{spec}}_{u,v} \leq j$ ensures that messages can only be delivered in the order they have been received by $\mathsf{TH}_{\mathcal{H}}$, i.e., neither replay attacks

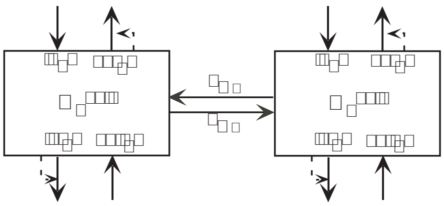


Fig. 6. Sketch of system Sys_1

nor reordering messages is possible for the adversary; cf. [2] for details. The user will receive inputs of the form $(\mathsf{receive}, u, m)$ and $(\mathsf{rec\_init}, u)$, respectively. If v is dishonest, $\mathsf{TH}_{\mathcal{H}}$ will simply output (send, m, v) to the adversary. Finally, the adversary can send a message m to a user u by sending a command $(\mathsf{receive}, v, m)$ to the port $\mathsf{from\_adv}_u?$ of $\mathsf{TH}_{\mathcal{H}}$ for a corrupted user v, and he can also stop the machine of any user by sending a command (stop) to a corresponding port of $\mathsf{TH}_{\mathcal{H}}$, which corresponds to exceeding the machine's runtime bound in the real world.

Necessary modifications of the abstract scheme for secure ordered channels. We want our system to fulfill our flow policy shown in Figure 5, so especially the non-interference requirement $NIReq_{A,a,\mathcal{G}}$ must hold. If we explicitly allow the adversary to schedule the communication between H_a and H_b he can obviously achieve two distinguishable behaviours by denial of service attacks as follows. On the one hand, he directly schedules every message sent from H_b to H_a in one behaviour, on the other hand he does not schedule any message sent from H_b to H_a. This problem cannot be solved by the filtering system Sys_1 if scheduling of the communication channel is done by the adversary.[5] In practice, this means that two persons will not be able to communicate without being interfered from outside if the channel they use can be cut off by the adversary. A possible solution for the problem is to define reliable, non-authenticated channels between a and b, so messages sent between two participants are not only output to the adversary but also output to the recipient and directly scheduled. Obviously, channels which are reliable *and* authenticated could be used as well for sending of messages, but in this case, we would no longer need the underlying cryptography (e.g., authentication). Therefore, we only consider these authenticated channels for key exchange as usual, but sending of messages is still performed over non-authenticated channels. The modifications carry over to the trusted host $\mathsf{TH}_{\mathcal{H}}$ as follows:

– If $\mathsf{TH}_{\mathcal{H}}$ receives an input $(\mathsf{snd\_init})$ from H_a, it implicitly initializes a communication with H_b and outputs $(\mathsf{snd\_init})$ to the adversary, $(\mathsf{rec\_init}, a)$ to H_b and schedules the second output.

[5] Sys_1 can only sort out messages from "wrong" senders, the messages mentioned are sent by the "valid" user H_b, so they have to be delivered because Sys_1 has no internal clock to check for denial of service attacks.

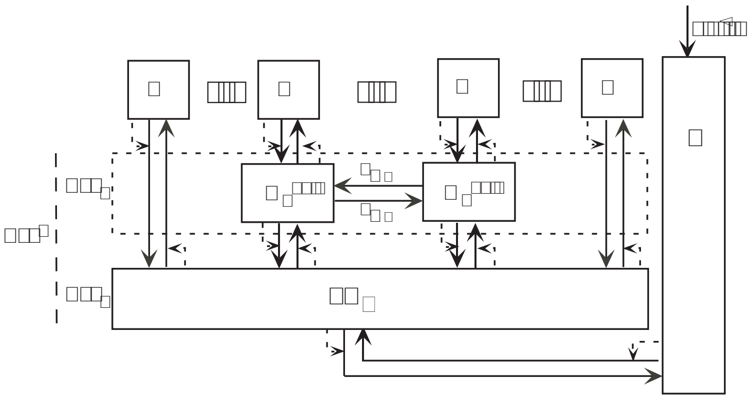


Fig. 7. Ideal system for non-interfered communication

– If $\mathsf{TH}_{\mathcal{H}}$ receives an input (send, m, b) from H_a, it outputs $(\mathsf{send\_blindly}, i, l, b)$ to the adversary, $(\mathsf{receive}, m, a)$ to H_b scheduling the second output.

These modifications are also done for switched variables a and b. We will omit a more detailed description of the machine $\mathsf{TH}_{\mathcal{H}}$ due to lack of space and refer the reader to the long version of this paper. After this brief review and modification of Sys'_0 we can turn our attention to the system Sys_1. The system Sys_1 is built by additional machines $\mathsf{M}_u^{\mathsf{n\text{-}in}}$ for $u \in \{a, b\}$. These machines will be inserted between the users and the trusted host $\mathsf{TH}_{\mathcal{H}}$, see Figure 7. Formally, we obtain the following scheme:

Scheme 1 **(Sys_1)** Let $n \in \mathbb{N}$ and polynomials $L, s, s' \in \mathbb{N}[x]$ be given. Here n denotes the number of intended participants, $L(k)$ bounds the message length and $s(k), s'(k)$ bound the number of messages each user can send and receive respectively for a security parameter k. Let $\mathcal{I} := \{1, \ldots, n\}$ denote the set of possible users again and $a, b \in \mathcal{I}$ the special users that should not be influenced from outside. Then

$$Sys_1 := \{(\hat{M}, S)\}$$

with $\hat{M} = \{\mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}\}$ and $S^c := \{\mathsf{out}_u?, \mathsf{in}_u!, \mathsf{in}_u{}^{\triangleleft}! \mid u \in \{a, b\}\} \cup \{\mathsf{in}'_u?, \mathsf{out}'_u!, \mathsf{out}'_u{}^{\triangleleft}! \mid u \in \{a, b\}\}$. Without loss of generality we just describe the ports and the behaviour of machine $\mathsf{M}_a^{\mathsf{n\text{-}in}}$. The machine $\mathsf{M}_b^{\mathsf{n\text{-}in}}$ is defined analogously by exchanging the variables a and b. The ports of machine $\mathsf{M}_a^{\mathsf{n\text{-}in}}$ are $\{\mathsf{in}_a?, \mathsf{out}_a!, \mathsf{out}_a{}^{\triangleleft}!\} \cup \{\mathsf{out}'_a?, \mathsf{in}'_a!, \mathsf{in}'_a{}^{\triangleleft}!\} \cup \{\mathsf{p}_{\mathsf{M}_b}?, \mathsf{p}_{\mathsf{M}_a}!, \mathsf{p}_{\mathsf{M}_a}{}^{\triangleleft}!\}$, cf. Figure 6.

Internally, $\mathsf{M}_a^{\mathsf{n\text{-}in}}$ maintains a counter $s_a \in \{0, \ldots, s(k)\}$ and an array $(s'_{a,u})_{u \in \mathcal{I}}$ over $\{0, \ldots, s'(k)\}$ bounding the number of messages H_a can send and receive, respectively, and a variable $stopped_a \in \{0, 1\}$ all initialized with 0 everywhere. The state-transition function of $\mathsf{M}_a^{\mathsf{n\text{-}in}}$ is defined by the following rules, written in a simple pseudo-code language.

Initialization.

- **Send initialization:** On input $(\mathsf{snd\_init})$ at $\mathsf{in}_a?$: If $s_a < s(k)$ it sets $s_a := s_a + 1$, otherwise it stops. If $stopped_a = 0$ it outputs $(\mathsf{snd\_init})$ at $\mathsf{in}'_a!$, 1 at ${\mathsf{in}'_a}^{\triangleleft}!$, otherwise it outputs $(\mathsf{snd\_init})$ at $\mathsf{p}_{\mathsf{M}_a}!$, 1 at ${\mathsf{p}_{\mathsf{M}_a}}^{\triangleleft}!$.
- **Receive initialization:** On input $(\mathsf{rec\_init}, u)$ at $\mathsf{out}'_a?$: It first checks whether $s'_{a,u} < s'(k)$ hold. In this case it sets $s'_{a,u} = s'_{a,u} + 1$, otherwise it stops. If $stopped_a = 0$ it checks $u = b$. If this also holds it outputs $(\mathsf{rec\_init}, b)$ at $\mathsf{out}_a!$ and 1 at ${\mathsf{out}_a}^{\triangleleft}!$. On input $(\mathsf{rec\_init}, b)$ at $\mathsf{p}_{\mathsf{M}_b}?$, it outputs $(\mathsf{rec\_init}, b)$ at $\mathsf{out}_a!$ and 1 at ${\mathsf{out}_a}^{\triangleleft}!$.

Sending and receiving messages.

- **Send:** On input (send, m, v) at $\mathsf{in}_a?$ with $m \in \Sigma^+$ and $\mathsf{len}(m) \leq L(k)$ it checks whether $s_a < s(k)$. If this holds it sets $s_a := s_a + 1$, otherwise it stops. If $stopped_a = 0$ holds, it outputs (send, m, v) at $\mathsf{in}'_a!$, 1 at ${\mathsf{in}'_a}^{\triangleleft}!$. Otherwise it first checks $v = b$. After a successful test it outputs $(\mathsf{receive}, a, m)$ at $\mathsf{p}_{\mathsf{M}_a}!$ and 1 at ${\mathsf{p}_{\mathsf{M}_a}}^{\triangleleft}!$.
- **Receive:** On input $(\mathsf{receive}, u, m)$ at $\mathsf{out}'_a?$ it first checks whether $s'_{a,u} < s'(k)$. If this holds it sets $s'_{a,u} := s'_{a,u} + 1$, otherwise it stops. If $u = b$ holds it outputs $(\mathsf{receive}, b, m)$ at $\mathsf{out}_a!$ and 1 at ${\mathsf{out}_a}^{\triangleleft}!$. On input $(\mathsf{receive}, b, m)$ at $\mathsf{p}_{\mathsf{M}_b}?$ it outputs $(\mathsf{receive}, b, m)$ at $\mathsf{out}_a!$ and 1 at ${\mathsf{out}_a}^{\triangleleft}!$.
- **Stop:** On input (stop) at $\mathsf{out}'_a?$ or $\mathsf{p}_{\mathsf{M}_b}?$: If $stopped_a = 0$, it sets $stopped_a = 1$ and outputs (stop) at $\mathsf{p}_{\mathsf{M}_a}!$ and 1 at ${\mathsf{p}_{\mathsf{M}_a}}^{\triangleleft}!$.

The special communication ports $\mathsf{p}_{\mathsf{M}_a}$ and $\mathsf{p}_{\mathsf{M}_b}$ are just included to prevent denial of service attacks. We already briefly stated in our review of Sys'_0 that a mighty attacker could simply overpower the machine of an honest user by sending too many messages, i.e., to exceed its runtime bound in the real world. In the ideal system this is modeled by letting the adversary stop arbitrary machines any time he likes. If we now consider an adversary that stops the machine of user a at the very start of the run and another one that never stops this machine, we would certainly obtain different views for this user. This problem cannot really be avoided if we do not provide additional channels for communication that guarantee availability. In practice this would correspond to a connection that contains trash all the time sent by the adversary, so the users (their machines in our case) would certainly look for a new way to communicate. Furthermore, this problem is much weaker in practice than in theory because it ought to be impossible (or at least very difficult) for an adversary to overpower a machine (the machine would surely be able to take countermeasures). If we did not consider these sorts of attacks the ports $\mathsf{p}_{\mathsf{M}_a}$ and $\mathsf{p}_{\mathsf{M}_b}$ could as well be omitted. Finally, a stopped machine $\mathsf{M}_a^{\mathsf{n\text{-}in}}$ would want the machine $\mathsf{M}_b^{\mathsf{n\text{-}in}}$ also to use the special communication ports, so it will stop the machine as soon as it has been stopped itself. Before we now build the combination of both systems to obtain our complete system Sys^*, we rename the ports $\mathsf{in}_u?$, $\mathsf{out}_u!$ and ${\mathsf{out}_u}^{\triangleleft}!$ of Sys'_0 into $\mathsf{in}'_u?$, $\mathsf{out}'_u!$ and ${\mathsf{out}'_u}^{\triangleleft}!$, respectively, for $u \in \{a, b\}$ such that Sys'_0 and Sys_1 are connected in the desired way. Furthermore, we restrict the structures of

Sys'_0 to all sets $\mathcal{H}$ with $\{a,b\} \subseteq \mathcal{H}$. Combination now means that we combine every structure of Sys'_0 with the (only) structure of Sys_1. The resulting system $Sys^* = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \{a,b\} \subseteq \mathcal{H} \subseteq \mathcal{I}\}$ is shown in Figure 7.

Remark 2. It is quite obvious how to modify the system Sys_1 to an arbitrary set of users (instead of $\{a,b\}$) that have to be guarded by the firewall. Moreover we can easily consider multiple disjoint sets of users so that a user can communicate with other users of its own set without being interfered from outside. This corresponds to multiple firewalls and can easily be achieved by modifying the filtering system Sys_1, so our specification carries over to arbitrary transitive flow policies.

5.3 The Real System

The real system $Sys^{\#}$ is derived by replacing the ideal system Sys'_0 with its concrete implementation Sys_0. For our purpose, it is sufficient to give an informal review of the system Sys_0. The system is a standard cryptographic system of the form $Sys_0 = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$, i.e., any subset of participants may be dishonest. It uses asymmetric encryption and digital signatures as cryptographic primitives. A user u can let his machine create signature and encryption keys that are sent to other users over authenticated channels $\mathsf{aut}_{u,v}$. Furthermore, messages sent from user u to user v will be signed and encrypted by M_u and sent to M_v over an insecure channel $\mathsf{net}_{u,v}$, representing the net in the real world. Similar to $\mathsf{TH}_{\mathcal{H}}$ each machine maintains internal counters which are used for discarded messages that are out of order. The adversary is able to schedule the communication between the users, and he can furthermore send arbitrary messages m to arbitrary users u for a dishonest sender v.

We now have to implement the modification of the ideal system in our concrete implementation. This can simply be achieved by changing the channel type of the (formerly insecure) channels between a and b to reliable, non-authenticated. This channel type is not comprised by the original model of [18], but it can be defined quite easily, cf. [1]. Moreover, by inspection of the proof of [2] and [18], it is easy to see that the "at least as secure as" relation still holds for these modified systems Sys_0 and Sys_1 with only slight changes in the proof. Therefore, and due to lack of space, we omit the proof here and refer the reader to the long version again.

5.4 Non-interference Proof

In the following we will show that our abstract system Sys^* (cf. Figure 7) fulfills its non-interference requirements given by the following flow policy. For two given elements $i_1, i_2 \in \mathcal{I} \cup \{\mathsf{A}\}$, we define $(S_{i_1}, S_{i_2}) \in \not\rightsquigarrow$ iff $i_1 \in (\mathcal{H} \setminus \{a,b\}) \cup \{\mathsf{A}\}$ and $i_2 \in \{a,b\}$. The flow policy is sketched in Figure 5.

Theorem 2. *(Non-Interference Properties of Sys^*) Let an arbitrary structure $(\{\mathsf{TH}_{\mathcal{H}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}\}, S_{\mathcal{H}}) \in Sys^*$ be given. For the sake of readability, we set $\hat{M}_{\mathcal{H}} := \{\mathsf{TH}_{\mathcal{H}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}\}$ in the following. Let a function ϕ_{Sys^*} be given* *that maps the structures $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})$ of Sys^* to the flow policy $\mathcal{G}_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})} = (\Delta_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})}, \mathcal{E}_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})})$ as defined above. The partition $\Delta_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})}$ of $S_{\mathcal{H}}$ is defined by $\Delta_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})} := \{S_i \mid i \in \mathcal{H}\} \cup \{\bar{S}\}$ with $S_i^c := \{\mathsf{out}_i?, \mathsf{in}_i!, \mathsf{in}_i{}^{\triangleleft}!\}$ for $i \in \mathcal{H}$ and $S_{\mathsf{A}} := \bar{S} = \mathsf{free}([\hat{M}_{\mathcal{H}}]) \setminus (\bigcup_{i\in\mathcal{H}} S_i)$. Then Sys^* fulfills ϕ_{Sys^*} perfectly.* □

Before we turn our attention to the proof of Theorem 2, we present the following lemma.

Lemma 2. *By definition of the system, the following invariants hold for all possible runs of the configuration.*

1. *The collection $\{\mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}\}$, i.e., the system Sys_1, is polynomial-time.*
2. *If H_a receives an input at $\mathsf{out}_a?$, it is of the form $(\mathsf{rec\_init}, b)$ or $(\mathsf{receive}, b, m)$ for an arbitrary $m \in \Sigma^+$. If H_a receives an input at $\mathsf{master}_a?$, it is sent by the master scheduler and is of the form 1.*
3. *No output of $\mathsf{X}^{\mathsf{n\text{-}in}}$ at $\mathsf{master}_a!$ depends on inputs from other machines. Each machine is clocked equally often using a rotating clocking scheme. Furthermore, each output at a port $\mathsf{p}^{\triangleleft}!$ for $\mathsf{p}^{\triangleleft}! \in S_a^c$ and the scheduled message does only depend on prior outputs of H_a at port $\mathsf{p}^{\mathsf{s}}!$ and $\mathsf{p}!$.*
4. *If H_a receives a term of the form $(\mathsf{rec\_init}, b)$ at $\mathsf{out}_a?$, it is a direct consequence of the input $(\mathsf{snd\_init})$ sent by H_b (i.e., the scheduling sequence must have been $\mathsf{H}_b, \mathsf{X}^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}, \mathsf{TH}_{\mathcal{H}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{H}_a$ or $\mathsf{H}_b, \mathsf{X}^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{H}_a$). This also implies that initializing a communication between H_a and H_b is not possible for the adversary, so there cannot be any replay attacks with initialization commands because they will be sorted out by $\mathsf{TH}_{\mathcal{H}}$.*
5. *If H_a receives a term of the form $(\mathsf{receive}, b, m)$ at $\mathsf{out}_a?$, it is a direct consequence (in the sense of Point 4) of the message (send, a, m) sent by H_b, so the scheduling sequence has been $\mathsf{H}_b, \mathsf{X}^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}, \mathsf{TH}_{\mathcal{H}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{H}_a$ or $\mathsf{H}_b, \mathsf{X}^{\mathsf{n\text{-}in}}, \mathsf{M}_b^{\mathsf{n\text{-}in}}, \mathsf{M}_a^{\mathsf{n\text{-}in}}, \mathsf{H}_a$. Thus, it is not possible for the adversary to pretend to be user H_b and furthermore the number of received messages of this form equals the number of messages sent by H_b to H_a. Therefore, the adversary can neither replay these messages nor throw them away.*

The invariants also hold if we exchange the variables a and b. □

The proof is omitted due to lack of space. It will be contained in the long version.

Proof (Theorem 2). We have to show that Sys^* fulfills the non-interference requirement ϕ_{Sys^*}. Let an arbitrary structure $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \in Sys^*$ be given so we have a flow policy $\mathcal{G}_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})} = (\Delta_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})}, \mathcal{E}_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})})$ for this structure. Let now two arbitrary blocks $S_{i_1}, S_{i_2} \in \Delta_{(\hat{M}_{\mathcal{H}},S_{\mathcal{H}})}$ with $i_1 \in (\mathcal{H} \setminus \{a,b\}) \cup \{\mathsf{A}\}$, $i_2 \in \{a,b\}$ be given, so $(S_{i_1}, S_{i_2}) \in \not\rightsquigarrow$ must hold. By definition of non-interference showing $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \models_{\mathsf{perf}} NIReq_{i_1,i_2,\mathcal{G}}$ is sufficient for proving $Sys^* \models_{\mathsf{perf}} \phi_{Sys^*}$.

Let a non-interference configuration $conf^{n\text{-}in}_{i_1,i_2} = (\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}, U^{n\text{-}in}, \mathsf{A})^{n\text{-}in}$ for this structure be given. Without loss of generality we can assume $i_2 = a$ because of the symmetry of the flow policy. Depending on the choice of the bit b we denote the two families of views of H_a by $view_{conf^{n\text{-}in}_{i_1,a},0}(\mathsf{H}_a)$ and $view_{conf^{n\text{-}in}_{i_1,a},1}(\mathsf{H}_a)$.

Assume for contradiction that the probability of a correct guess $b = b^*$ is greater than $\frac{1}{2}$, which implies $\mathit{view}_{\mathit{conf}^{n\text{-}in}_{i_1,a},0}(\{\mathsf{H}_a, \mathsf{H}_b\}) \neq \mathit{view}_{\mathit{conf}^{n\text{-}in}_{i_1,a},1}(\{\mathsf{H}_a, \mathsf{H}_b\})$. First of all, we can exclude denial of service attacks applying Part 3 of the above lemma, so there has to be a first input at $\{\mathsf{H}_a, \mathsf{H}_b\}$ with different probability in both cases because Part 3 ensures that scheduling of messages sent by a user only depends on its own prior behaviour. We will now use the previous lemma to show that this cannot happen.

By Part 2 of Lemma 2 this input can only be of the form $(\mathsf{rec\_init}, u)$, $(\mathsf{receive}, u, m)$ at $\mathsf{out}_u?$, or 1 at $\mathsf{master}_u?$ for $u \in \{a, b\}$. We will in the following write $\bar{u}$ for the other emphasized user (i.e., $\bar{u} \in \{a, b\} \setminus \{u\}$). Assume this input to be of the first form. Now Part 4 implies that this input is a direct consequence of an input $(\mathsf{snd\_init})$ sent by the other emphasized user $\mathsf{H}_{\bar{u}}$. Hence, there had to be an input of $\mathsf{H}_{\bar{u}}$ with different probability in both cases which contradicts our assumption of the *first* different input, so there cannot be any influence from outside Sys_1. Thus, we obtain identical probability distributions for possible inputs of H_a in both cases yielding the desired contradiction.

Now assume this input to be of the form $(\mathsf{receive}, u, m)$. By Part 5 the corresponding input $(\mathsf{send}, \bar{u}, m)$ must have been sent directly by H_u with exactly the same message m. Furthermore, the underlying system for secure ordered channels ensures that the message has been sent exactly that often as H_a receives this input, so there cannot be any influence from outside because of the same reason as in the first case.

Finally, assume this input to be at port $\mathsf{master}_u?$. This input does not depend on arbitrary behaviours of other machines by Part 3 so we obtain identical probability distributions again. Therefore, the views must in fact be identical in both cases so the probability of a correct guess $b = b^*$ is exactly $\frac{1}{2}$. Thus, we have $\mathit{Sys}^* \models_{\mathsf{perf}} \phi_{\mathit{Sys}^*}$. ■

After proving the non-interference property for the ideal specification, we now concentrate on the concrete implementation.

Theorem 3. *(Non-Interference Properties of $\mathit{Sys}^{\#}$) The real system $\mathit{Sys}^{\#}$ fulfills the non-interference property $\phi_{\mathit{Sys}^{\#}}$ computationally, with $\varphi_{\mathit{Sys}^{\#}}$ given as in theorem 1. In formulas, $\mathit{Sys}^{\#} \models_{\mathsf{poly}} \phi_{\mathit{Sys}^{\#}}$.* □

Proof. Putting it all together, we know that the original and also the modified real implementation of secure message transmission with ordered channels is computationally at least as secure as its (modified) specification. Using Part 1 of Lemma 2 we know that the system Sys_1 is polynomial-time, which is an essential precondition for applying the composition theorem in the computational case, so we have $\mathit{Sys}^{\#} \geq_{\mathsf{poly}} \mathit{Sys}^*$. Since perfect fulfillment of non-interference requirements implies computational fulfillment, we have $\mathit{Sys}^{\#} \models_{\mathsf{poly}} \phi_{\mathit{Sys}^{\#}}$ using theorem 1. ■

6 Conclusion

We have presented the first general definition of probabilistic non-interference in reactive systems which includes a computational case (Section 3). Our approach is mainly motivated by the concept of simulatability which is fundamental for modern cryptography, and it might help to build a bridge between prior research in the field of information flow and designing systems involving real cryptographic primitives. We have shown that our definition behaves well under simulatability (Section 4), which enables modular proofs and stepwise refinement without destroying the non-interference properties. This is not only important for the development process of cryptographic protocols but also because non-interference properties of ideal systems not containing any cryptographic details can often easily be validated by formal proof tools whereas real systems are usually much more difficult to validate. As an example, we have presented an abstract specification of a cryptographic firewall guarding two honest users from their environment (Section 5). Moreover, we have presented a concrete implementation that also fits our definition, which we have shown using our preservation theorem.

Acknowledgments

We thank *Heiko Mantel*, *Matthias Schunter*, and *Michael Waidner* for interesting discussions.

References

[1] M. Backes. Cryptographically sound analysis of security protocols. Ph.D thesis, Computer Science Department, Saarland University, 2002. Available at http://www-krypt.cs.uni-sb.de/∼mbackes/diss.ps. 19

[2] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In Proceedings of Formal Methods Europe 2002 (FME'02), Copenhagen, 2002. 14, 16, 19

[3] D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford MA, USA, March 1976. 1

[4] D. Clark, C. Hankin, S. Hunt, and R. Nagarajan. Possibilistic information flow is safe for probabilistic non-interference. Workshop on Issues in the Theory of Security (WITS'00), available at www.doc.ic.ac.uk/ clh/Papers/witscnh.ps.gz. 1

[5] D. E. Denning. A lattice model of secure information flow. Communications of the ACM 19/5 (1976) 236-243. 1

[6] J. A. Goguen and J. Meseguer. Security policies and security models. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington 1982, 11-20. 1

[7] J. A. Goguen and J. Meseguer. Unwinding and inference control. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Oakland 1984, 75-86. 1

[8] O. Goldreich. Foundations of cryptography: Basic tools. Cambridge University Press, 2001. 11

[9] J. W. Gray III. Probabilistic interference. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos 1990, 170-179. 1

[10] J. W. Gray III. Toward a mathematical foundation for information flow security. Journal of Computer Security, Vol.1, no.3,4, 1992, 255-295. 1

[11] C. A. R. Hoare. Communicating sequential processes. International Series in Computer Science, Prentice Hall, Hemel Hempstead 1985. 3

[12] P. Laud. Semantics and program analysis of computationally secure information flow. 10th European Symposium On Programming (ESOP 2001), LNCS 2028, Springer-Verlag, Berlin 2001, 77-91. 2

[13] N. Lynch. Distributed algorithms. Morgan Kaufmann Publishers, San Francisco 1996. 3

[14] H. Mantel. Unwinding possibilistic security properties. 6th European Symposium on Research in Computer Security (ESORICS'00), Toulouse 2000, 238-254. 1

[15] D. McCullough. Specifications for multi-level security and a hook-up property. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Oakland 1987, 161-166. 1

[16] J. McLean. Security models. in: John Marciniak (ed.): Encyclopedia of Software Engineering; Wiley Press, 1994. 1

[17] J. McLean. Security models and information flow. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos 1990, 180-187. 1

[18] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. IEEE Symposium on Security and Privacy, Oakland, May 2001, 184-201. 2, 3, 13, 14, 19

[19] D. Sutherland. A model of information. 9th National Computer Security Conference; National Bureau of Standards, National Computer Security Center, September 1986, 175-183. 1

[20] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos 1990, 144-161. 1

[21] A. C. Yao. Protocols for secure computations. 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160-164. 11

[22] A. Zakinthinos and E. S. Lee. A general theory of security properties. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington 1997, 94-102. 1

Bit-Slice Auction Circuit

Kaoru Kurosawa[1] and Wakaha Ogata[2]

[1] Ibaraki University
4-12-1 Nakanarusawa, Hitachi, Ibaraki, 316-8511, Japan
kurosawa@cis.ibaraki.ac.jp
[2] Tokyo Institute of Technology
2–12–1 O-okayama, Meguro-ku, Tokyo 152–8552, Japan
wakaha@ss.titech.ac.jp

Abstract. In this paper, we introduce a *bit-slice* approach for auctions and present a more efficient circuit than the *normal* approach for the highest-price auction. Our circuit can be combined with any auction protocol based on general circuit evaluation. Especially, if we combine with the mix and match technique, then we can obtain a highest-price auction protocol which is at least seven times faster. A second-price auction protocol is also easily constructed from our circuit.

Keywords: auction, multiparty protocol, bit-slice, circuit, mix and match

1 Introduction

1.1 Sealed-Bid Auction

Auctions are getting very popular in the Internet. They are now a major area in the web electric commerce.

A sealed-bid auction consists of two phases, the bidding phase and the opening phase. In the bidding phase, all the bidders submit their bidding prices to the auctioneer. In the opening phase (of the highest-price auction), the auctioneer announces the highest price and the identities of the winners.

In the second-price auction (also known as Vickrey auction), the highest bidder wins, and the clearing price, the price that the winner has to pay, is equal to the second highest bid. (It is closer to the real life auction than the highest-price auction.)

Throughout the paper, we assume that:

- There are n servers.
- There are m bidders, denoted by $A_1, A_2, \dots, A_m$.
- Each bidder A_i has a k-bit bid $B_i = (b_i^{(k-1)}, \dots, b_i^{(0)})_2$.

In general, $X = (x^{(k-1)}, \dots, x^{(0)})_2$ denotes a k-bit integer, where $x^{(k-1)}$ denotes the most significant bit and $x^{(0)}$ denotes the least significant bit.

1.2 Auction Protocols Based on Circuit Evaluation

In the trivial scheme which assumes a single trusted auctioneer, the auctioneer knows all the bidding prices. He may also tell a lie about the highest price and the winners.

Hence we need a cryptographically secure auction protocol which satisfies privacy and correctness. The correctness means that the announced highest price and the identities of the winners are guaranteed to be correct. The privacy means that no adversary can compute any other information.

In principle, it is known that any function can be computed securely by using general multiparty protocols [25, 16, 5, 9, 17]. (They are also known as general function evaluation protocols.) A problem is, however, that the cost of each bidder is very large in their general forms. Therefore, several schemes have been proposed to achieve efficient and secure auctions.

In the auction protocol of Naor, Pinkas and Sumner [22] which involves two servers, a proxy oblivious transfer protocol was introduced and it was combined with Yao's garbled circuit technique [25]. Jakobsson and Juels pointed out a flaw and it was fixed by Juels and Szydlo [21]. The fixed protocol is secure if the two servers do not collude. The cost of each server is $O(mkt)$, where t is a security parameter.

Jakobsson and Juels introduced a new general multiparty protocol called *mix and match* and showed its application to auctions [20]. The *mix and match* technique avoids the use of verifiable secret sharing schemes (VSS) which is intensively used in the other general multiparty protocols. In their auction protocol (JJ auction protocol), therefore, each bidder A_i has only to submit her encrypted bidding price without executing a VSS. The cost of each server is $O(mnk)$ exponentiations.

Cramer, Damgård and Nielsen introduced another general multiparty protocol which avoids VSS [10]. It can also be used for auctions. While the mix and match protocol is based on the DDH assumption alone, it is not known if this is possible for this protocol. On the other hand, the round complexity is $O(d)$ while it is $O(n + d)$ in the mix and match protocol, where d is the depth of the circuit of a given function. The message complexities are the same.

Baudron and Stern showed an auction protocol which assumes a semi-trusted auctioneer T [4]. In this protocol, T blindly and noninteractively evaluates a circuit whose output tells if A_i is a winner or not for each bidder A_i. A_i knows if he is a winner by decrypting the output ciphertext of this circuit. T learns no information if he does not collude with any bidder. The cost of T is $\Theta(mk^m)$. (This protocol is not based on *general* circuit evaluation. It uses some special predicates and makes use of its special properties.)

1.3 Our Contribution

In general, a multiparty protocol for computing a function $f(x_1, \ldots, x_n)$ is designed as follows. First we draw a Boolean circuit C_f which computes f. We next apply a general gate evaluation technique to each gate of C_f. Therefore, the smaller the circuit size is, the more efficient the protocol is.

The normal approach for circuit design for auctions is to compare the bidding prices one by one (in other words, to run a millionaire's problem protocol in order). To the authors' knowledge, the smallest size circuit in this approach for the highest-price auction requires $7mk$ logical gates and m Select_k gates, where a logical gate has 2 input and 1 output bits, and a Select_k gate has $2k+1$ input and k output bits. (We present such a circuit in Sec. 2.)

In this paper, we introduce a *bit-slice* approach for auctions and present a more efficient circuit than the *normal* approach for the highest-price auction. The proposed circuit requires only $2mk$ logical gates while the normal approach circuit requires $7mk$ logical gates and m Select_k gates as shown above.

Suppose that $B_{\max} = (b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)})_2$ is the highest bidding price, where $b_{\max}^{(k-1)}$ denotes the most significant bit and $b_{\max}^{(0)}$ denotes the least significant bit. Then the proposed approach first determines $b_{\max}^{(k-1)}$ by looking at the most significant bits of all the bids. It next determines $b_{\max}^{(k-2)}$ by looking at the second most significant bits of all the bids, and so on.

Our circuit can be combined with any auction protocol based on *general* circuit evaluation. Especially, if we combine our circuit with the mix and match technique, then we can further reduce the number of gates just to mk by using the homomorphic property of the encryption function. Hence we can obtain a protocol which is at least seven times faster than JJ auction protocol.

We also show that a second-price auction protocol (which is closer to the real-life auction than the highest-price auction) can be easily obtained from our bit-slice circuit for the highest-price auction.

1.4 Other Related Works

There are many auction protocols which do not use circuit evaluation. However, they have problems such as follows.

The first cryptographic auction scheme was proposed by Franklin and Reiter [14]. This scheme is not fully private, in the sense that it only ensures the confidentiality of bids until the end of the protocol.

In the scheme of Cachin [6] which involves two servers, a partial order of bids is leaked to one of the two servers. The cost of each bidder is $O(mkt)$ and the cost of each server is $O(m^2kt)$, where t is a security parameter.

In the scheme of Di Crescenzo [13] which involves a single server, a honest but curious server does not learn any information under the quadratic residuosity assumption. However, nothing is known about the security if the server is malicious. The cost of each bidder is $O(m^2k^2)$ and the cost of server is also $O(m^2k^2)$.

Some other works require $O(mn2^k)$ cost for each server [18, 23]. In the scheme of [3], the cost of a server is $O(m2^k)$ and the cost of each bidder is $O(2^k)$. Note that these costs are much larger than the cost of auction protocols based on circuit design such as [22, 20].

1.5 Organization of the Paper

In Sec. 2, we present the normal approach for circuit design for auctions. In Sec. 3, we propose a bit-slice circuit for the highest-price auction. In Sec. 4, we briefly describe the mix and match technique. In Sec. 5, we show a new highest-price auction protocol which is obtained by combining the bit slice circuit and the mix and match technique. In Sec. 6, we present our second-price auction protocol.

2 Normal Approach for Auction Circuit Design

The normal approach for circuit design for auctions is to compare the bidding prices one by one (in other words, to run a millionaire's problem protocol in order). In this section, we present such a circuit which seems to be the most efficient.

2.1 Primitive Gate

For two bits x and y, define $Bigger_1$ and EQ_1 by

$$Bigger_1(x,y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{otherwise} \end{cases}$$

$$EQ_1(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

We next define a gate $Select_\kappa$ which has $2\kappa + 1$ input bits and κ output bits as follows.

$$Select_\kappa(b, x^{(\kappa-1)}, \ldots, x^{(0)}, y^{(\kappa-1)}, \ldots, y^{(0)}) = \begin{cases} (x^{(\kappa-1)}, \ldots, x^{(0)}) & \text{if } b = 1 \\ (y^{(\kappa-1)}, \ldots, y^{(0)}) & \text{if } b = 0 \end{cases}$$

2.2 Boolean Circuit for the Millionaire's Problem

For $X = (x^{(k-1)}, \ldots, x^{(0)})_2$ and $Y = (y^{(k-1)}, \ldots, y^{(0)})_2$, define

$$Bigger_k(X,Y) = \begin{cases} 1 & \text{if } X > Y \\ 0 & \text{otherwise} \end{cases}$$

$$Max_k(X,Y) = \begin{cases} X & \text{if } X > Y \\ Y & \text{otherwise} \end{cases}$$

$$EQ_k(X,Y) = \begin{cases} 1 & \text{if } X = Y \\ 0 & \text{otherwise} \end{cases}$$

We first show two circuits for the millionaire's problem, $Bigger_k$ and Max_k, which seem to be the most efficient.

Let $a_k = 1$. For $i = k-1$ to 1, do

$$a_i = a_{i+1} \wedge EQ_1(x^{(i)}, y^{(i)}).$$

Then

$$\begin{aligned}
\mathit{Bigger}_k(X,Y) &= \mathit{Bigger}_1(x^{(k-1)}, y^{(k-1)}) \\
&\quad \vee(\mathit{Bigger}_1(x^{(k-2)}, y^{(k-2)}) \wedge a_{k-1}) \\
&\qquad \vdots \\
&\quad \vee(\mathit{Bigger}_1(x^{(0)}, y^{(0)}) \wedge a_1). \\
\mathit{Max}_k(X,Y) &= \mathit{Select}_k(\mathit{Bigger}_k(X,Y), X, Y).
\end{aligned}$$

2.3 Boolean Circuit for Auction

We next present two circuits for the highest-price auction, *Highest* and *Winner*.

Highest outputs the highest bidding price of m bids, $B_1, \ldots, B_m$. It is obtained by implementing the following algorithm by a circuit. Let $B_{\max} = 0$. For $i = 1, \ldots, m$, do

$$B_{\max} := \mathit{Max}_k(B_{\max}, B_i).$$

It is clear that the final $B_{\max}$ is the highest bidding price.

Winner is a circuit which outputs the winners. That is,

$$\mathit{Winner}(B_1, \ldots, B_m) = (w_1, \ldots, w_m),$$

where

$$w_i = \begin{cases} 1 & \text{if } B_i = B_{\max} \\ 0 & \text{otherwise} \end{cases}$$

Each w_i is obtained as follows.

$$w_i = EQ_k(B_i, B_{\max}).$$

The sizes of these circuits will be given in Sec. 3.3.

3 Bit-Slice Approach

In this section, we present a more efficient circuit than the *normal* approach by using a *bit-slice* approach for the highest-price auction. Suppose that $B_{\max} = (b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)})_2$ is the highest bidding price. Then the proposed circuit first determines $b_{\max}^{(k-1)}$ by looking at the most significant bits of all the bids. It next determines $b_{\max}^{(k-2)}$ by looking at the second most significant bits of all the bids, and so on.

For two m-dimensional binary vectors $\mathbf{X} = (x_1, \ldots, x_m)$ and $\mathbf{Y} = (y_1, \ldots, y_m)$, define

$$\mathbf{X} \wedge \mathbf{Y} = (x_1 \wedge y_1, \ldots, x_m \wedge y_m).$$

3.1 Proposed Circuit for Auction

Our idea is as follows. Let D_j be the highest price when considering the upper j bits of the bids. That is,

$$\begin{aligned} D_1 &= (b_{\max}^{(k-1)}, 0 \cdots, 0)_2, \\ D_2 &= (b_{\max}^{(k-1)}, b_{\max}^{(k-2)}, 0 \cdots, 0)_2, \end{aligned}$$

etc,

$$D_k = (b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)})_2 = B_{\max}.$$

In the first round, we find $b_{\max}^{(k-1)}$ and then eliminate all the bidders A_i such that $B_i < D_1$. In the second round, we find $b_{\max}^{(k-2)}$ and then eliminate all the bidders A_i such that $B_i < D_2$, and so on. At the end, the remained bidders are the winners. For that purpose, we update $\mathbf{W} = (w_1, \ldots, w_m)$ such that

$$w_i = \begin{cases} 1 & \text{if } B_i \geq D_j \\ 0 & \text{otherwise} \end{cases}$$

for $j = 1$ to k.

Our circuit is obtained by implementing the following algorithm. For given m bids, $B_1, \ldots, B_m$, define $\mathbf{V}_j$ as

$$\mathbf{V}_j = (b_1^{(j)}, \ldots, b_m^{(j)})$$

for $j = 0, \ldots, k-1$. That is, $\mathbf{V}_j$ is the vector consisting of the $j+1$th lowest bit of each bid.

Let $\mathbf{W} = (1, \ldots, 1)$. For $j = k-1$ to 0, do;

(Step 1) For $\mathbf{W} = (w_1, \ldots, w_m)$, let

$$\begin{aligned} \mathbf{S}_j &= \mathbf{W} \wedge \mathbf{V}_j \\ &= (w_1 \wedge b_1^{(j)}, \ldots, w_m \wedge b_m^{(j)}), & (1) \\ b_{\max}^{(j)} &= (w_1 \wedge b_1^{(j)}) \vee \cdots \vee (w_m \wedge b_m^{(j)}). & (2) \end{aligned}$$

(Step 2) If $b_{\max}^{(j)} = 1$, then let $\mathbf{W} = \mathbf{S}_j$.

Then the highest price is obtained as $B_{\max} = (b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)})_2$. Let the final $\mathbf{W}$ be $(w_1, \ldots, w_m)$. Then A_i is a winner if and only if $w_i = 1$.

We record this as the following theorem.

Theorem 1. *In the above algorithm,*

- $B_{\max}$ *is the highest bidding price.*
- *For the final* $\mathbf{W} = (w_1, \ldots, w_m)$*,* A_i *is a winner if and only if* $w_i = 1$*.*

The size of our circuit will be given in Sec. 3.3.

3.2 Example

Suppose that $m = 4$, $k = 5$ and each bid is

$$\begin{aligned} B_1 &= 20 = (1,0,1,0,0)_2, \\ B_2 &= 17 = (0,1,1,1,1)_2, \\ B_3 &= 18 = (1,0,0,1,0)_2, \\ B_4 &= 29 = (1,1,1,0,1)_2. \end{aligned}$$

Then $\mathbf{V}_4 = (1,0,1,1)$, $\mathbf{V}_3 = (0,1,0,1)$ and etc. Let $\mathbf{W} = (1,1,1,1)$. Now

1. $\mathbf{S}_4 = \mathbf{W} \wedge \mathbf{V}_4 = (1,0,1,1)$, $b_{\max}^{(4)} = 1$ and $\mathbf{W} := \mathbf{S}_4 = (1,0,1,1)$.
2. $\mathbf{S}_3 = \mathbf{W} \wedge \mathbf{V}_3 = (0,0,0,1)$, $b_{\max}^{(3)} = 1$ and $\mathbf{W} := \mathbf{S}_3 = (0,0,0,1)$.
3. $\mathbf{S}_2 = \mathbf{W} \wedge \mathbf{V}_2 = (0,0,0,1)$, $b_{\max}^{(2)} = 1$ and $\mathbf{W} := \mathbf{S}_2 = (0,0,0,1)$.
4. $\mathbf{S}_1 = \mathbf{W} \wedge \mathbf{V}_1 = (0,0,0,0)$, $b_{\max}^{(1)} = 0$.
5. $\mathbf{S}_0 = \mathbf{W} \wedge \mathbf{V}_0 = (0,0,0,1)$, $b_{\max}^{(0)} = 1$ and $\mathbf{W} := \mathbf{S}_0 = (0,0,0,1)$.

Therefore, we obtain that the highest bidding price is $(b_{\max}^{(4)}, \ldots, b_{\max}^{(0)})_2 = (1,1,1,0,1)_2 = 29$ and A_4 is the winner.

3.3 Comparison of Circuit Size

In this subsection, we compare the size of the normal circuit shown in Sec. 2 and that of our bit-slice circuit for the highest-price auction. See Table 1.

First the size of the normal circuit is given as follows. The circuit $Bigger_k$ requires k $Bigger_1$ gates, $2(k-1)$ AND gates, $k-1$ OR gates and $k-1$ EQ_1 gates. The circuit Max_k requires one $Select_k$ gate and one $Bigger_k$ circuit. The circuit *Highest* is obtained by implementing m Max_k circuits. In addition, the circuit *Winner* requires m EQ_k gates, where an EQ_k gate is implemented by k EQ_1 gates and $(k-1)$ AND gates.

Therefore, the normal circuits for the highest-price auction, *Highest* and *Winner*, require mk $Bigger_1$ gates, $3m(k-1)$ AND gates, $m(k-1)$ OR gates, $m(2k-1)$ EQ_1 gates and m $Select_k$ gates in total.

Next our bit slice circuit is given by implementing Eq.(1) and Eq.(2) k times. Therefore, it requires mk AND gates and $(m-1)k$ OR gates.

Hence roughly speaking, the proposed circuit requires only $2mk$ logical gates while the normal circuit requires $7mk$ logical gates and m $Select_k$ gates.

Table 1. Comparison of circuit sizes

	AND	OR	$Bigger_1$	EQ_1	$Select_k$
Normal circuit	$3m(k-1)$	$m(k-1)$	mk	$m(2k-1)$	m
Bit-slice circuit	mk	$(m-1)k$	0	0	0

4 MIX and Match Protocol

4.1 Overview

In this section, we briefly describe the mix and match technique introduced by Jakobsson and Juels [20]. It is a general multiparty protocol which does not use VSS. Instead, it uses a homomorphic encryption scheme (for example, ElGamal) and a MIX net [8, 1, 2].

This model involves n players, denoted by $P_1, P_2, \ldots, P_n$ and assumes that there exists a public board. We consider an adversary who may corrupt up to t players, where $n \geq 2t + 1$.

The players agree in advance on a representation of the target function f as a circuit C_f. Suppose that C_f consists of N gates, $G_1, \ldots, G_N$. Let the input of P_i be a k-bit integer B_i. The aim of the protocol is for players to compute $f(B_1, \ldots, B_n)$ without revealing any additional information. It goes as follows.

Input stage: Each P_i computes ciphertexts of the bits of B_i and broadcasts them. She proves that each ciphertext represents 0 or 1 in zero-knowledge by using the technique of [11].

Mix and Match stage: The players blindly evaluates each gate G_j in order.

Output stage: After evaluating the last gate G_N, the players obtain o_N, a ciphertext encrypting $f(B_1, \ldots, B_n)$. They jointly decrypt this ciphertext value to reveal the output of the function f.

This protocol meets the security requirements formalized by Canetti [7] for secure multiparty protocols [20, page 171]. The cost of each player is $O(nN)$ exponentiations and the overall message complexity is also $O(nN)$ (see [20, page 172]).

The details of the protocol are described in the following subsections.

4.2 Requirements for the Encryption Function

Let E be a public-key probabilistic encryption function. We denote by $E(m)$ the set of encryptions for a plaintext m and by $e \in E(m)$ a particular encryption of m. We say that e is a standard encryption if it is encrypted with no randomness.

E must satisfy the following properties.

– **homomorphic property**
  There exists a polynomial time computable operations, $^{-1}$ and $\otimes$, as follows for a large prime q.
  1. If $e \in E(m)$, then $e^{-1} \in E(-m \bmod q)$.
  2. If $e_1 \in E(m_1)$ and $e_2 \in E(m_2)$, then $e_1 \otimes e_2 \in E(m_1 + m_2 \bmod q)$.

  For a positive integer a, define

$$a \cdot e = \underbrace{e \otimes e \otimes \cdots \otimes e}_{a}.$$

- **random re-encryptability**
  Given $e \in E(m)$, there is a probabilistic re-encryption algorithm that outputs $e' \in E(m)$, where e' is uniformly distributed over $E(m)$.
- **threshold decryption**
  For a given ciphertext $e \in E(m)$, any t out of n players can decrypt e along with a zero-knowledge proof of the correctness. However, any $t-1$ out of n players cannot decrypt e.

Such $E(\cdot)$ can be obtained by slightly modifying ElGamal encryption scheme over a group G of order $|G| = q$, where q is a large prime. G can be constructed as a subgroup of Z_p^*, where p is a prime such that $q \mid p-1$. It can also be obtained from elliptic curves.

Let g be a generator of G, i.e. $G = \langle g \rangle$. The secret key x is randomly chosen from Z_q and the public key is $y = g^x$. An encryption of m is given by

$$(g^r, g^m y^r) \in E(m),$$

where $r \in Z_q$ is a random element. For ciphertexts, define $^{-1}$ and $\otimes$ as

$$(u, v)^{-1} = (u^{-1}, v^{-1}).$$

$$(u_1, v_1) \otimes (u_2, v_2) = (u_1 u_2, v_1 v_2).$$

Then it is easy to see that the homomorphic property is satisfied. A re-encryption of $(u, v) \in E(m)$ is given by $(u', v') = (g^{r'} u, y^{r'} v)$ for a random element $r' \in Z_q$.

For threshold decryption, each player obtains a private share x_i of x in Shamir's $(t+1, n)$-threshold secret-sharing scheme [24]. Each g^{x_i} is published. For details, see [12, 15]. Each player needs to broadcast $O(1)$ messages and compute $O(n)$ exponentiations in threshold decryption.

4.3 MIX Protocol

A MIX protocol takes a list of ciphertexts $(\xi_1, \ldots, \xi_L)$ and outputs a permuted and re-encrypted list of the ciphertexts $(\xi'_1, \ldots, \xi'_L)$ without revealing the relationship between $(\xi_1, \ldots, \xi_L)$ and $(\xi'_1, \ldots, \xi'_L)$, where ξ_i or ξ'_i can be a single ciphertext e, or a list of l ciphertexts, $(e_1, \ldots, e_l)$, for some $l > 1$. We further require that anybody (even an outsider) can verify the validity of $(\xi'_1, \ldots, \xi'_L)$ (public verifiability).

For small L, a MIX protocol is efficiently implemented as shown in [1, 2, 19]. In this protocol, each player needs to compute $O(nlL \log L)$ exponentiations and broadcast $O(nlL \log L)$ messages.

4.4 Plaintext Equality Test

Given two ciphertexts $e_1 \in E(m_1)$ and $e_2 \in E(m_2)$, this protocol checks if $m_1 = m_2$. Let $e_0 = e_1 \otimes e_2^{-1}$.

Table 2. Logical Table of AND

x_1	x_2	$x_1 \wedge x_2$
$a_1 \in E(0)$	$b_1 \in E(0)$	$c_1 \in E(0)$
$a_2 \in E(0)$	$b_2 \in E(1)$	$c_2 \in E(0)$
$a_3 \in E(1)$	$b_3 \in E(0)$	$c_3 \in E(0)$
$a_4 \in E(1)$	$b_4 \in E(1)$	$c_4 \in E(1)$

(Step 1) For each player P_i (where $i = 1, \ldots, n$):
P_i chooses a random element $a_i \in Z_q$ and computes $z_i = a_i \cdot e_0$. He broadcasts z_i and proves the validity of z_i in zero-knowledge.

(Step 2) Let $z = z_1 \otimes z_2 \otimes \cdots \otimes z_n$. The players jointly decrypt z by threshold verifiable decryption and obtain the plaintext m.

Then it holds that

$$m = \begin{cases} 0 & \text{if } \ m_1 = m_2, \\ \mathit{random} & \text{otherwise} \end{cases} \tag{3}$$

This protocol is minimal knowledge under the DDH assumption (see [20, page 169]). The cost of each player is $O(n)$ exponentiations.

4.5 Mix and Match Stage

For each logical gate $G(x_1, x_2)$ of a given circuit, n players jointly computes $E(G(x_1, x_2))$ from $e_1 \in E(x_1)$ and $e_2 \in E(x_2)$ keeping x_1 and x_2 secret. For simplicity, we show a mix and match stage for AND.

1. n players first consider the standard encryption of each entry of Table 2.
2. By applying a MIX protocol to the four rows of Table 2, n players jointly compute blinded and permuted rows of Table 2. Let the ith row be (a'_i, b'_i, c'_i) for $i = 1, \ldots, 4$.
3. n players next jointly find the row i such that the plaintext of e_1 is equal to that of a'_i and the plaintext of e_2 is equal to that of b'_i by using the plaintext equality test protocol.
4. For this i, it holds that $c'_i \in E(x_1 \wedge x_2)$.

5 Bit-Slice Circuit and Mix-Match Protocol

5.1 Overview

We can obtain a highest-price auction protocol by combining the proposed circuit of Sec. 3.1 with any general multiparty protocol. Then a more efficient protocol is obtained because the bit-slice circuit is more efficient than the normal circuit as shown in Table 1.

In this section, we show a combination with the mix and match technique, as an example. In this case, we can further improve the efficiency. That is, we can remove all the OR computations of Eq.(2) by using the homomorphic property of the encryption function as follows. Let

$$h_j = (x_1 \wedge b_1^{(j)}) + \cdots + (x_m \wedge b_m^{(j)}).$$

Then it is easy to see that $h_j = 0$ if and only if $b_{\max}^{(j)} = 0$. Therefore, n servers have only to execute a plaintext equality test protocol for checking if $h_j = 0$ to decide if $b_{\max}^{(j)} = 0$. Note that each server can compute summation locally by using the homomorphic property of the encryption scheme. Hence we can replace $(m-1)k$ OR computations with only k plaintext equality tests.

5.2 Bidding Phase

Each bidder A_i computes a ciphertext of her bidding price B_i as

$$ENC_i = (e_{i,k-1}, \ldots, e_{i,0}),$$

where $e_{i,j} \in E(b_i^{(j)})$, and submits ENC_i. She also proves in zero-knowledge that $b_i^{(j)} = 0$ or 1 by using the technique of [11]. (This submission may be also digitally signed by A_i.)

5.3 Opening Phase

Suppose that $e_1 \in E(b_i)$ and $e_2 \in E(b_2)$, where b_1 and b_2 are binary. Let $Mul(e_1, e_2)$ denote a protocol which outputs

$$e \in E(b_1 \wedge b_2)$$

by applying a mix and match protocol for AND.

Let $\widetilde{\mathbf{W}} = (\tilde{w}_1, \ldots, \tilde{w}_m)$, where each $\tilde{w}_j \in E(1)$ is the standard encryption.

(Step 1) For $j = k-1$ to 0, do:

(Step 1-a) For $\widetilde{\mathbf{W}} = (\tilde{w}_1, \ldots, \tilde{w}_m)$, n servers jointly compute

$$\widetilde{\mathbf{S}}_j = (Mul(\tilde{w}_1, e_{1,j}), \ldots, Mul(\tilde{w}_m, e_{m,j})).$$

(Step 1-b) Each server locally computes

$$h_j = Mul(\tilde{w}_1, e_{1,j}) \otimes \cdots \otimes Mul(\tilde{w}_m, e_{m,j}).$$

(Step 1-c) n servers jointly check if $h_j \in E(0)$ by using a plaintext equality test. Let

$$b_{\max}^{(j)} = \begin{cases} 0 & \text{if } h_j \in E(0) \\ 1 & \text{otherwise} \end{cases}$$

(Step 1-d) If $b_{\max}^{(j)} = 1$, then let $\widetilde{\mathbf{W}} = \widetilde{\mathbf{S}}_j$.

(Step 2) For the final $\widetilde{\mathbf{W}} = (\tilde{w}_1, \ldots, \tilde{w}_m)$, n servers jointly decrypt each $\tilde{w}_i$ by threshold verifiable decryption and obtain the plaintext w_i.

The highest price is obtained as $B_{\max} = (b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)})_2$. A_i is a winner if and only if $w_i = 1$.

Table 3. Comparison of the highest-price auction protocols

	AND	OR	$Bigger_1$	EQ_1	$Select_k$
JJ [20]	$3m(k-1)$	$m(k-1)$	mk	$m(2k-1)$	m
Proposed	mk	0	0	0	0

5.4 Comparison

If we combine the normal circuit with the mix and match technique, then JJ auction protocol is obtained [20]. Table 3 shows the comparison between JJ auction protocol and proposed protocol.

In our protocol, we can replace $(m-1)k$ OR computations of Table 1 with only k plaintext equality tests as shown in Sec. 5.1. Therefore, our protocol requires mk AND computations and k plaintext equality tests.

We omit the cost of k plaintext equality tests in this table because it is much smaller than the cost for each logical gate. For example, an AND computation requires 8 plaintext equality tests and a MIX protocol of four items if we implement the protocol of Sec. 4.5. Hence mk AND computations requires $8mk$ plaintext equality tests and mk MIX protocols of four items. This is much larger than the cost of k plaintext equality tests.

Now roughly speaking, our protocol requires mk logical gates while JJ auction protocol requires $7mk$ logical gates and m $Select_k$ circuit. Therefore, our protocol is at least seven times faster than JJ auction protocol.

5.5 Discussion

We can slightly improve the opening phase by letting the initial value of $\widetilde{\mathbf{W}}$ be

$$\widetilde{\mathbf{W}} = (e_{1,k-1}, \cdots, e_{m,k-1}).$$

The efficiency is further improved if each bid is only $k' < k$ bits long. (Of course, this fact is not known in advance.) The smaller k' is, the faster the modified version is. For example, if $B_i = (0, \ldots, 0, b_i^{(0)})$ for all i, then no mix and match for AND is required. If $B_i = (0, \ldots, 0, b_i^{(1)}, b_i^{(0)})$ for all i, then the mix and match for AND is executed only once, and so on.

Such speed-up is impossible in JJ auction protocol.

6 Second-Price Auction

In the second-price auction (also known as Vickrey auction), the highest bidder wins, and the clearing price, the price that the winner has to pay, is equal to the second highest bid. Therefore, Vickrey auctions are much closer to the real life auction than the highest-price auctions. A cryptographically secure second-price auction scheme should reveal only the identities of the winners (the highest bidders) and the second-highest bid.

In this section, we show that a second-price auction protocol is easily obtained from our bit-slice circuit for the highest-price auction.

6.1 Normal Approach Circuit

If we use the normal approach, we can obtain a second-price auction circuit by keeping track of the two highest bids found so far, B_{first} and B_{second}, as follows.

Let $B_{\text{first}} = B_{\text{second}} = 0$. For $i = 1, \ldots, m$, do

$$\begin{aligned} X &:= \mathit{Select}_k(\mathit{Bigger}_k(B_{\text{second}}, B_i), B_{\text{second}}, B_i). \\ B_{\text{second}} &:= \mathit{Select}_k(\mathit{Bigger}_k(B_{\text{first}}, X), X, B_{\text{first}}). \\ B_{\text{first}} &:= \mathit{Select}_k(\mathit{Bigger}_k(B_{\text{first}}, X), B_{\text{first}}, X). \end{aligned}$$

It is easy to see that the final B_{first} is the highest bidding price and B_{second} is the second-highest price. The identities of the winners are obtained similarly to Sec. 2.3.

6.2 Bit-Slice-Type Second-Price Auction

We define two types of highest-price auction schemes, a winner-only scheme and a price-only scheme. A winner-only scheme reveals only the identities of the winners, but not the highest price. A price-only scheme reveals only the highest bid, but not the identities of the winners.

Now suppose that there is a winner-only scheme Q_1 and a price-only scheme Q_2. Then we can obtain a second-price auction scheme as follows:

Step 1. Run Q_1.
Step 2. Delete the winners of Q_1 from the set of bidders.
Step 3. Run Q_2 for the rest of the bidders.

Indeed, the above scheme reveals only the identities of the winners of Q_1, and the highest bidding price of Q_2 which is the second highest price among the bidders.

Such protocols Q_1 and Q_2 are easily obtained from our bit-slice circuit for the highest-price auction as follows.

- Apply any multiparty protocol to the circuit of Sec. 3.1 and decrypt only $b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)}$ (but not $\mathbf{W}$) in the output stage. Then we obtain a price-only scheme.
- In the circuit of Sec. 3.1, replace Step 2 with

$$\mathbf{W} = \mathit{Select}_m(b_{\max}^{(j)}, \mathbf{S}_j, \mathbf{W}).$$

  Then apply any multiparty protocol to the above circuit and decrypt only $\mathbf{W}$ (but not $b_{\max}^{(k-1)}, \ldots, b_{\max}^{(0)}$) in the output stage. We now obtain a winner-only scheme.

Table 4. Comparison of the second-price auction protocols

	AND	OR	$Bigger_1$	EQ_1	$Select_k$	$Select_m$
Normal	$5m(k-1)$	$2m(k-1)$	$2mk$	$m(2k-1)$	$3m$	0
Bit slice	$(2m-1)k$	$(m-1)k$	0	0	0	k

6.3 Comparison

Suppose that we use the mix and match technique as a general multiparty protocol. In this case, the price-only scheme is obtained by deleting Step 2 of Sec. 5.3.

Then roughly speaking, our second-price auction protocol requires $3mk$ logical gates and k $Select_m$ gates. On the other hand, the second-price auction protocol obtained by combining the normal circuit and the mix and match technique requires $11mk$ logical gates and $3m$ $Select_k$ gates.

We show a comparison in Table 4.

Acknowledgement

The authors thank Martin Hirt for providing many useful comments.

References

[1] M. Abe, "Mix-Networks on permutation networks," Proc. of Asiacrypt '99, LNCS Vol. 1716, pp. 258–273 (1999). 31, 32

[2] M. Abe and F. Hoshino, "Remarks on Mix-Network Based on Permutation Networks," Proc. of PKC 2001, pp. 317–324 (2001). 31, 32

[3] M. Abe and K. Suzuki, "M1-st Price Auction Using Homomorphic Encryption," Proc. of PKC2002, pp. 115–124 (2002). 26

[4] O. Baudron and J. Stern, "Non-Interactive Private Auctions," Proc. of Financial Cryptography 2001 (2001). 25

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation," Proceedings of the twentieth annual ACM Symp. Theory of Computing, STOC, pp. 1–10, May 2–4 (1988). 25

[6] C. Cachin, "Efficient private bidding and auctions with an oblivious third party," ACM CSS '99, pp. 120–127, ACM (1999). 26

[7] R. Canetti, "Security and composition of multiparty cryptographic protocols," Journal of Cryptology, Vol. 13, No. 1, pp. 143–202 (2000). 31

[8] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," Communications of the ACM, Vol. 24, pp. 84–88 (1981). 31

[9] D. Chaum, C. Crépeau, and I. Damgård. "Multiparty unconditionally secure protocols," Proc. of the twentieth annual ACM Symp. Theory of Computing, STOC, pp. 11–19, May 2–4 (1988). 25

[10] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty Computation from Threshold Homomorphic Encryption," Proc. of Eurocrypt'01, LNCS Vol. 2045, pp. 280–300 (2001). 25

[11] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," Proc. of CRYPTO '94. LNCS Vol. 839, pp. 174–187 (1994). 31, 34

[12] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," In Proc. of Crypto '89, pp. 307–315. 32

[13] G.Di Crescenzo, "Private selective payment protocols," In Proc. of Financial Cryptography '00, LNCS Vol. 1962, pp. 72–89 (2000). 26

[14] M. Franklin and M. Reiter, "The Design and Implementation of a Secure Auction Service," IEEE Trans. on Software Engineering, Vol. 22, No. 5 (1996). 26

[15] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," In Proc of Eurocrypt '96, LNCS Vol. 1070, pp. 354–371 (1996). 32

[16] O. Goldreich, S. Micali, and A. Wigderson. "How to play any mental game," In Proceedings of the nineteenth annual ACM Symp. Theory of Computing, STOC, pp. 218–229, May 25–27 (1987). 25

[17] M. Hirt, U. Maurer, and B. Przydatek, "Efficient secure multiparty computation," Proc. of Asiacrypt'2000, LNCS Vol. 1976, pp. 143–161 (2000). 25

[18] M. Harkavy, J. D. Tygar, and H. Kikuchi, "Electronic auction with private bids," In Third USENIX Workshop on Electronic Commerce Proceedings, pp. 61–74 (1998). 26

[19] M. Jakobsson and A. Juels, "Millimix: Mixing in small batches," DIMACS Technical report 99-33 (June 1999). 32

[20] M. Jakobsson and A. Juels, "Mix and Match: Secure Function Evaluation via Ciphertexts," In Proc. of Asiacrypt 2000, LNCS Vol. 1976, pp. 162–177 (2000). 25, 26, 31, 33, 35

[21] A. Juels and M. Szydlo, "An Two-Server Auction Protocol," In Proc. of Financial Cryptography 2002, (2002). 25

[22] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," In 1st ACM Conference on Electronic Commerce, pp. 129–140, ACM (1999). 25, 26

[23] K. Sako, "An auction protocol which hides bids of losers," Proc. of PKC'2000, LNCS Vol. 1751, pp. 422–432 (2000). 26

[24] A. Shamir, "How to Share a Secret," Communications of the ACM, Vol. 22, pp. 612-613 (1979). 32

[25] A. Yao, "Protocols for secure computations (extended abstract)," Proc. of FOCS'82, pp. 160–164, IEEE Computer Society (1982). 25

Confidentiality Policies and Their Enforcement for Controlled Query Evaluation

Joachim Biskup[1] and Piero Bonatti[2]

[1] Fachbereich Informatik, Universität Dortmund
D-44221 Dortmund, Germany
biskup@ls6.informatik.uni-dortmund.de

[2] Dipartimento di Tecnologie dell'Informazione, Università di Milano
I-26013 Crema, Italy
bonatti@dti.unimi.it

Abstract. An important goal of security in information systems is *confidentiality*. A confidentiality policy specifies which users should be forbidden to acquire what kind of information, and a controlled query evaluation should enforce such a policy even if users are able to reason about a priori knowledge and the answers to previous queries. We put the following aspects into a unifying and comprehensive framework: formal models of confidentiality *policies* based on potential secrets or secrecies, user *awareness* of the policy instance, and *enforcement* methods applying either lying or refusal, or a combination of lying and refusal. Two new evaluation methods are introduced. Different approaches are systematically compared and evaluated.

Keywords: Inference control; Controlled query evaluation; Confidentiality; Policy; Potential secret; Secrecy; Refusal; Lying; Combined refusal and lying.

1 Introduction

An important goal of security in information systems is *confidentiality*, i.e., the ability to hide specific information from certain users according to some confidentiality policy. Roughly speaking, such a confidentiality policy specifies which users should be forbidden to acquire what kind of information. Typically, a confidentiality policy is expressed by the owner or a designated administrator of the information to be hidden. Using a logic-oriented model of information systems, including the relational model and Datalog, a controlled query evaluation has to enforce such a policy, even if users are able to *infer* more information than what is explicitly returned as an answer to their queries.

In this paper, we deal with the theoretical foundations of controlled query evaluation. Our work is based on a simple but powerful logic-oriented data model which considers an instance of an information system as a structure (interpretation) in the sense of logic, a query as a sentence, and the ordinary answer as the truth value of the query w.r.t. the instance.

This theoretical model is nonetheless relevant for practical and static approaches to information flow control and inference control [6, 5], including discretionary and mandatory access control, and data customization such as statistical perturbation and polyinstantiation.

Previous work on controlled query evaluation [1, 2, 3, 4, 9] has identified three important aspects:

1. Formal models of confidentiality *policies*: potential secrets versus secrecies.
   Under the model of *secrecies*, a confidentiality policy requires that, given a set of sentences that constitute the policy instance, a user should not be able to infer the truth value of those sentences in the current instance of the information system. Whereas under the model of *potential secrets*, a confidentiality policy requires that, for a given set of sentences, if any of these sentences is true in the current instance of the information system then the user should not be able to infer that fact. However, users are allowed to believe the opposite truth value. Thus under the former model, query answers should not suggest any truth value at all for the given set of sentences, while under the latter model, a predetermined alternative of the truth values can be suggested.
   Clearly, the goal of confidentiality has to be complemented by availability. In this paper, like in previous work, we deal with availability only by observing the following informal heuristic. A controlled query evaluation should be as cooperative to the user as possible, i.e., it should return a distorted answer (i.e., a refusal or a lie) only if necessary for achieving confidentiality.
2. User *awareness* of the policy instance: unknown versus known.
   A user who *knows* the policy instance might use such knowledge to infer hidden information, say, by the following kind of reasoning: "I received a distorted answer. The system did so only because otherwise the correct answer together with previously observed data would logically imply a specific sentence Ψ contained in the policy instance. So, observing the distortion, I conclude that Ψ is valid indeed." Even if the actual policy instance is *unknown* to him, the user might base a similar reasoning solely on his awareness of the model of the policy. Surely, any enforcement method should make such arguments impossible.
3. *Enforcement* method: lying versus refusal, and combined lying and refusal.
   (Uniform) *refusal* returns a `mum`, i.e., it refuses to return the truth value of the query, if the correct answer lets the user infer (either directly or indirectly) one of the sentences of the policy instance. Under similar conditions, (uniform) *lying* returns a lie, i.e., the negation of the correct answer. *Combined* methods are allowed to protect the sentences of the policy instance by both kinds of answer distortion.

Sicherman/de Jonge/van de Riet [9] introduce the topic of controlled query evaluation and discuss refusal for unknown and known secrecies. Bonatti/Kraus/Subrahmanian [4] pioneer lying for known potential secrets. Biskup [1] defines a common framework for studying and comparing enforcement methods dealing

Table 1. Aspects for controlled query evaluation and contributions of previous work

		confidentiality policy			
		potential secrets		secrecies	
enforcement	lying		[4] [2, section 4]	"not applicable"	[1, section 5]
	refusal		[2, section 3]	[9] [1, Section 4.2]	[9] [1, section 4.1]
	combined		[3, section 3]	[3, section 4]	
		unknown	known	known	unknown
		user awareness			

with lying and refusal for unknown and known secrecies. A main conclusion is that "for unknown secrecies refusal is better than lying". Biskup/Bonatti [2] adapt that framework in order to treat lying and refusal for known potential secrets. They show that in this context lying and refusal are incomparable in general, but functionally equivalent under some natural restriction (i.e., the policy instance should be closed under disjunction). Finally, Biskup/Bonatti [3] suggest a combination of refusal and lying for known policies, including both potential secrets and secrecies. The combined methods avoid the drawbacks of the uniform methods (the mandatory protection of disjunctions for lying, and, for refusal, the independence from the actual instance of the information system).

Table 1 gives a structured overview of all these works, highlighting the combinations of the three major aspects that have not yet been investigated. In this paper, we try to fill in the table's gaps, and provide a unifying view of all these approaches, including a general analysis of their mutual relationships. More precisely, the structure and the contributions of the paper are as follows.

– In Section 2 we introduce a uniform reformulation of the existing approaches, including a unified notion of confidentiality, parametric w.r.t the three classification dimensions.
Furthermore, this section points out the mutual relationships between the existing approaches for known policies and their counterparts for unknown policies, and the relationships between the methods based on secrecies and their counterparts for potential secrets. In the latter case, we observe that each method based on a policy instance presented as a set of secrecies

$$secr := \{\{\Psi_1, \neg\Psi_1\}, \ldots, \{\Psi_k, \neg\Psi_k\}\} \tag{1}$$

is equivalent to the corresponding method based on potential secrets, applied to the policy instance

$$pot\_sec(secr) := \{\Psi_1, \neg\Psi_1, \ldots, \Psi_k, \neg\Psi_k\}\,. \tag{2}$$

– In Section 3, we fill in the two table slots corresponding to lies and refusal for unknown potential secrets. The new methods are derived from their coun-

terparts for unknown secrecies by analogy with the reduction of secrecies to potential secrets illustrated in (2).
Then, in the same section, we try to derive a combined method for unknown potential secrets from the corresponding method for known potential secrets, by analogy with the relationships observed in Section 2. Surprisingly, all the natural attempts along this direction fail to preserve confidentiality.
– In Section 4 we prove some general results that explain the relationships empirically observed in Section 2. As a byproduct, we identify some general properties of the controlled query evaluation methods under which the reduction of secrecies to potential secrets based on (2) preserves confidentiality. These results have a potential practical impact on the design of uniform access control mechanisms that support simultaneously both of the policy models (and combinations thereof).

The paper is closed by a section with a discussion of the results and some conclusions. Proofs will be omitted due to space limitations, with the exception of the proof of Theorem 4.

2 A Unified Formal Model for Confidentiality Policies

2.1 Ordinary Query Evaluation

An *information system* maintains two kinds of data: A *schema* DS captures the universe of discourse for the intended application and is formally defined as the set of all allowed instances. An *instance* db is a *structure* which *interprets* the symbols of some logic, i.e. of the universe of discourse (see e.g. [8, 7]). We only consider the most elementary kind of query, namely a sentence in the language of the logic. Given a structure db (stored as an instance) and a sentence Φ (issued as a query), Φ is either *true* (*valid*) or *false* in db, or in other words, the structure is either a *model* of the sentence or not. When a user issues a query Φ against the schema DS, the (ordinary) *query evaluation* $eval(\Phi)$ determines the pertinent case for the current instance db. Thus we formally define

$$eval(\Phi) : DS \rightarrow \{true, false\} \text{ with } eval(\Phi)(db) := db \texttt{ model\_of } \Phi\,, \tag{3}$$

where the boolean operator `model_of` is assumed to be appropriately specified for the logic under consideration.[1] We also use an equivalent formalization where either the queried sentence or its negation is returned:

$$eval^*(\Phi) : DS \rightarrow \{\Phi, \neg\Phi\} \text{ with}$$
$$eval^*(\Phi)(db) := \texttt{if } db \texttt{ model\_of } \Phi \texttt{ then } \Phi \texttt{ else } \neg\Phi\,. \tag{4}$$

[1] This abstract formulation makes our approach compatible with a variety of database models. To get a more concrete understanding of our framework, the reader may instantiate `model_of` with standard first-order satisfaction.

The symbols of the logic comprise both the negation symbol (as implicitely assumed above) and disjunction. We assume that both connectives have their classical semantics . Our definition trivially implies that for all instances *db* and for all queries Φ we have db `model_of` $eval^*(\Phi)(db)$.

We also define the semantic relationship $\models$ for logical implication in a standard way: $\Phi \models \Psi$ iff for every structure *db* such that *db* `model_of` Φ we also have *db* `model_of` Ψ. The complementary relationship is denoted with $\not\models$.

2.2 Controlled Query Evaluation

Controlled query evaluation consists of two steps. First, the correct answer is judged by some *censor* and then, depending on the output of the censor, some *modificator* is applied. In order to assist the censor, the system maintains a *user log*, denoted by *log*, which represents the explicit part of the *user's assumed knowledge*. Formally, *log* is declared to be a set of sentences. The log is meant to contain all the sentences that the user is assumed to hold true in the instance, in particular publicly known *semantic constraints*. Additionally, the log records the sentences returned as *answers to previous queries*.

Formally we will describe an approach to *controlled query evaluation* by a family of (possibly) partial functions $control\_eval(Q, log_0)$, each of which has two parameters: a (possibly infinite) query sequence $Q = \langle \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots \rangle$, and an initial user log log_0. The inputs to any such function are "admissible" pairs (*db*, *policy*) where *db* is an instance of the information system, and *policy* is an instance of a suitably formalized confidentiality policy. The admissibility of an argument pair (*db*, *policy*) is determined by some formal *precondition* associated with the function. Throughout the paper we suppose that for each function, the policies in its admissible pairs all belong to exactly one of the policy models. The function returns an answer sequence to the user, and updates the user log as a side effect. For any specific function, we indicate the underlying choices w.r.t. the three aspects—model of *p*olicy, user *a*wareness, *e*nforcement method—by a superscript p, a, e with $p \in \{\texttt{sec}, \texttt{ps}\}$, $a \in \{\texttt{unknown}, \texttt{known}\}$, and $e \in \{L(ying), R(efusal), C(ombined)\}$. In symbols,

$$control\_eval^{p,a,e}(Q, log_0)(db, policy) = \langle (ans_1, log_1), (ans_2, log_2), \ldots, (ans_i, log_i), \ldots \rangle ,$$

where the side effect on the user log is described by

$$log_i := \texttt{if } ans_i = \texttt{mum then } log_{i-1} \texttt{ else } log_{i-1} \cup \{ans_i\} ,$$

and a $censor^{p,a,e}$ is formally described by a truth function (or two truth functions for combined methods) with arguments of the form $(\Psi, log, db, policy)$. The censor returns *true* iff the modification e is required.

2.3 Confidentiality Requirements

The syntactical appearance of an *instance* of a confidentiality policy depends on the model: Either a policy instance is given by a finite set *secr* of complementary pairs of sentences, $secr = \{\{\Psi_1, \neg\Psi_1\}, \ldots, \{\Psi_k, \neg\Psi_k\}\}$, where each pair is called a *secrecy*. Or a policy instance is given by a finite set *pot_sec* of sentences, $pot\_sec = \{\Psi_1, \ldots, \Psi_k\}$, where each sentence is called a *potential secret*.

The original motivations for the two models of confidentiality policies are quite different (see [1, 2] for a detailed exposition).

– A secrecy $\{\Psi, \neg\Psi\}$ specifies the following: A user, seeing only the apriori knowledge log_0 and the returned answer sequence, should not be able to distinguish whether Ψ or $\neg\Psi$ is true in the actual instance of the information system, or speaking otherwise, both cases should be possible for him. More formally, the controlled query evaluation must be "nowhere injective with respect to the secrecy $\{\Psi, \neg\Psi\}$".
– A potential secret Ψ specifies the following: Such a user should not be able to exclude that $\neg\Psi$ is true in the actual instance of the information system, or speaking otherwise, this case should be possible for him. More formally, the controlled query evaluation must have a "surjective restriction on the false potential secret, i.e. on $\neg\Psi$".

The intended semantics is formalized as follows.

Definition 1. *Let* $control\_eval^{p,a,e}(Q, log_0)$ *describe a specific controlled query evaluation with precond as associated precondition for "admissible" arguments, and* $policy_1$ *be a policy instance.*

1. $control\_eval^{p,a,e}(Q, log_0)$ *is defined to* preserve confidentiality[2] *with respect to* $policy_1$ *iff*
for all finite prefixes Q' *of* Q,
for all instances db_1 *of the information system such that* $(db_1, policy_1)$ *satisfies precond,*
and for all $\Theta \in policy_1$,
there exists db_2 *and* $policy_2$ *such that* $(db_2, policy_2)$ *also satisfies precond and such that the following properties hold:*

   (a) [same answers]

   $control\_eval^{p,a,e}(Q', log_0)(db_1, policy_1) = control\_eval^{p,a,e}(Q', log_0)(db_2, policy_2)$;

   (b) [different secrets/false potential secrets]

   if $p = \texttt{sec}$, i.e., $\Theta = \{\Psi, \neg\Psi\}$ is a secrecy: $\{eval^*(\Psi)(db_1), eval^*(\Psi)(db_2)\} = \{\Psi, \neg\Psi\}$,

   if $p = \texttt{ps}$, i.e., $\Theta = \Psi$ is a potential secret: $eval^*(\Psi)(db_2) = \neg\Psi$;

   (c) [awareness] if $a = \texttt{known}$: $policy_1 = policy_2$.
2. *More generally,* $control\_eval(Q, log_0)^{p,a,e}$ *is defined to* preserve confidentiality *iff it preserves confidentiality with respect to all "admissible" policy instances.*

[2] Or to be *secure*, as a shorthand used in previous work

The above definition generalizes and encompasses all the notions of confidentiality introduced in previous works on controlled query evaluation. Some assumptions are implicit in the definition: (i) the user may know the algorithm of the controlled evaluation function, (ii) the user is rational, i.e., he derives nothing besides what is implied by his knowledge and the behavior of the database.

2.4 Existing Methods for Known Policies

Table 2 sketches the censors of the enforcement methods for known policies reported previously (i.e., columns 2 and 3 of Table 1) together with the pertinent preconditions in the sense of the new Definition 1. The original definitions are reformulated using the function $pot\_sec(.)$ (cf. (2)).

Proposition 1 (confidentiality). *The (suitably formalized versions of) previously reported enforcement methods for* known policies *preserve confidentiality in the sense of Definition 1.*

A careful comparison of the different methods summarized in Table 2 shows an interesting relationship between the treatment of potential secrets and secrecies. Each method for secrecies is equivalent to the corresponding method for potential secrets applied to the policy $pot\_sec(secr)$ (where $secr$ is the original policy). This is not immediately obvious for refusal, but it can be proved easily.

Proposition 2 (correspondence). *Each (suitably formalized versions of a) previously reported enforcement method for* known secrecies *using a policy instance secr is equivalent to the corresponding method for* known potential secrets *using the policy instance $pot\_sec(secr)$.*

2.5 Existing Methods for Unknown Policies

For unknown policies, only methods based on secrecies have been introduced so far. The existing methods for unknown secrecies (i.e., column 4 of Table 1) are summarized in the rightmost column of Table 3. Extending Proposition 1, we note that they also satisfy the generalized notion of confidentiality (Definition 1). The left column of Table 3 provides equivalent reformulations of the corresponding methods for known secrecies.

A crucial difference between a method for *known* policies and the corresponding method for *unknown* policies is the following: in the former case the pertinent censor takes care about *all* sentences in pot_sec or $pot\_sec(secr)$, respectively, whereas in the latter case only sentences that are *true in the current instance* of the information system are considered. This property makes the censors *instance dependent*, whereas the censors for known policies depend only on the query, the log and the policy.

Table 2. Correspondences between enforcement methods for *known* policies: for each method we indicate the pertinent censor as given in previous work and the precondition as requested by Definition 1. For combined methods, we make the simplifying assumption that $policy \neq \emptyset$ in order to avoid explicit consistency checks for the log

potential secrets $pot\_sec = \{\Psi_1, \ldots, \Psi_k\}$	**secrecies** $secr = \{\{\Psi_1, \neg\Psi_1\}, \ldots, \{\Psi_k, \neg\Psi_k\}\}$ $pot\_sec(secr) = \{\Psi_1, \neg\Psi_1, \ldots, \Psi_k, \neg\Psi_k\}$
lying [4], [2, section 4]: $censor^{\mathtt{ps},\mathtt{known},L}$: $log \cup \{eval^*(\Phi)(db)\} \models pot\_sec\_disj$, where $pot\_sec\_disj := \bigvee_{\Psi \in pot\_sec} \Psi$; $precondition^{\mathtt{ps},\mathtt{known},L}$: $log_0 \not\models pot\_sec\_disj$	**lying** [1, section 5]: $censor^{\mathtt{sec},\mathtt{known},L}$: "not applicable" $precondition^{\mathtt{sec},\mathtt{known},L}$: $log_0 \not\models \bigvee_{\Psi \in pot\_sec(secr)} \Psi$
refusal [2, section 3]: $censor^{\mathtt{ps},\mathtt{known},R}$: (exists $\Psi \in pot\_sec$) $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi$ or $log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi \,]$; $precondition^{\mathtt{ps},\mathtt{known},R}$: db `model_of` log_0 and (for all $\Psi \in pot\_sec$) $[\, log_0 \not\models \Psi \,]$	**refusal** [9], [1, section 4.2]: $censor^{\mathtt{sec},\mathtt{known},R}$: (exists $\Psi \in pot\_sec(secr)$) $[\, db$ `model_of` Ψ and $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi$ or $log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi$ or $log \cup \{\neg eval^*(\Phi)(db)\} \models \neg\Psi \,]\,]$; $precondition^{\mathtt{sec},\mathtt{known},R}$: db `model_of` log_0 and (for all $\Psi \in pot\_sec(secr)$) $[\, log_0 \not\models \Psi \,]$
combined [3, section 3]: $refusalcensor^{\mathtt{ps},\mathtt{known},C}$: (exists $\Psi_1 \in pot\_sec$) $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi_1 \,]$ and (exists $\Psi_2 \in pot\_sec$) $[\, log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi_2 \,]$; $lyingcensor^{\mathtt{ps},\mathtt{known},C}$: (exists $\Psi_1 \in pot\_sec$) $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi_1 \,]$ and (for all $\Psi_2 \in pot\_sec$) $[\, log \cup \{\neg eval^*(\Phi)(db)\} \not\models \Psi_2 \,]$; $precondition^{\mathtt{ps},\mathtt{known},C}$: (for all $\Psi \in pot\_sec$) $[\, log_0 \not\models \Psi \,]$	**combined** [3, section 4]: $refusalcensor^{\mathtt{sec},\mathtt{known},C}$: (exists $\Psi_1 \in pot\_sec(secr)$) $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi_1 \,]$ and (exists $\Psi_2 \in pot\_sec(secr)$) $[\, log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi_2 \,]$; $lyingcensor^{\mathtt{sec},\mathtt{known},C}$: (exists $\Psi_1 \in pot\_sec(secr)$) $[\, log \cup \{eval^*(\Phi)(db)\} \models \Psi_1 \,]$ and (for all $\Psi_2 \in pot\_sec(secr)$) $[\, log \cup \{\neg eval^*(\Phi)(db)\} \not\models \Psi_2 \,]$; $precondition^{\mathtt{sec},\mathtt{known},C}$: (for all $\Psi \in pot\_sec(secr)$) $[\, log_0 \not\models \Psi \,]$

Table 3. Correspondences between enforcement methods depending on the user awareness of the policy instance: For each method we indicate the suitably reformulated pertinent censor and the precondition as requested by Definition 1

known	unknown
lying under secrecies [1, section 5]: "not applicable" $censor^{\mathtt{sec},\mathtt{known},L}$: $log \cup \{eval^*(\Phi)(db)\} \models \bigvee_{\Psi \in pot\_sec(secr)} \Psi$ $precondition^{\mathtt{sec},\mathtt{known},L}$: $log_0 \not\models \bigvee_{\Psi \in pot\_sec(secr)} \Psi$	**lying under secrecies** [1, section 5]: $censor^{\mathtt{sec},\mathtt{unknown},L}$: $log \cup \{eval^*(\Phi)(db)\} \models$ $\bigvee_{\Psi \in pot\_sec(secr) \text{ and } db \ \mathtt{model\_of}\ \Psi} \Psi$ $precondition^{\mathtt{sec},\mathtt{unknown},L}$: $log_0 \not\models \bigvee_{\Psi \in pot\_sec(secr) \text{ and } db \ \mathtt{model\_of}\ \Psi} \Psi$
refusal under secrecies [9], [1, sect. 4.2]: $censor^{\mathtt{sec},\mathtt{known},R}$: (exists $\Psi \in pot\_sec(secr)$) [$log \cup \{eval^*(\Phi)(db)\} \models \Psi$ or $log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi$] ; $precondition^{\mathtt{sec},\mathtt{known},R}$: db $\mathtt{model\_of}$ log_0 and (for all $\Psi \in pot\_sec(secr)$)[$log_0 \not\models \Psi$]	**refusal under secrecies** [9], [1, sect. 4.1]: $censor^{\mathtt{sec},\mathtt{unknown},R}$: (exists $\Psi \in pot\_sec(secr)$) [db $\mathtt{model\_of}$ Ψ and $log \cup \{eval^*(\Phi)(db)\} \models \Psi$] ; $precondition^{\mathtt{sec},\mathtt{unknown},R}$: db $\mathtt{model\_of}$ log_0 and (for all $\Psi \in pot\_sec(secr)$) [if db $\mathtt{model\_of}$ Ψ then $log_0 \not\models \Psi$]

3 Filling in the Gaps

In this section, we shall try to fill in the gaps of Table 1 by analogy with the relationships between the existing methods observed in Section 2.4 and Section 2.5. In the next subsection, we shall derive secure controlled query evaluations for unknown potential secrets from their counterparts for unknown secrecies (Table 3), by analogy with the relationships observed in Table 2. In the second subsection, we shall try to adapt to unknown policies the combined methods for known policies, by analogy with the observations on Table 3. Unfortunately, it will turn out that all the obvious attempts in this direction fail to preserve confidentiality.

3.1 Lies and Refusal for Unknown Potential Secrets

In the case of *known* potential secrets, the controlled query evaluation based on *lies* uses a censor that protects the disjunction of *all* potential secrets. In the case of *unknown* potential secrets, we can take a less restrictive disjunction. Now it is sufficient to take care of only those potential secrets that are valid in the current instance of the information system. Note that in this case the censor actually depends on the current instance, in contrast to the case for known policies.

Formally we define the *uniform lying* method as follows. The $censor^{\mathtt{ps},\mathtt{unknown},L}$ is given by

$$log \cup \{eval^*(\Phi)(db)\} \models true\_pot\_sec\_disj \tag{5}$$

with $true\_pot\_sec\_disj := \bigvee_{\Psi \in pot\_sec \text{ and } db \ \mathtt{model\_of} \ \Psi} \Psi$, and the precondition $precondition^{\mathtt{ps},\mathtt{unknown},L}$ is given by

$$log_0 \not\models true\_pot\_sec\_disj\,. \tag{6}$$

Theorem 1 (confidentiality). *Uniform lying for unknown potential secrets preserves confidentiality in the sense of Definition 1.*

Next, we introduce a *uniform refusal* method for unknown potential secrets by following the same ideas adopted above. Formally, the method is defined as follows. The $censor^{\mathtt{ps},\mathtt{unknown},R}$ is given by

$$(\text{exists}\, \Psi \in pot\_sec)[\, db \ \mathtt{model\_of} \ \Psi \text{ and } log \cup \{eval^*(\Phi)(db)\} \models \Psi\,]\,, \tag{7}$$

and the $precondition^{\mathtt{ps},\mathtt{unknown},R}$ by

$$db \ \mathtt{model\_of} \ log_0 \text{ and } (\text{for all}\, \Psi \in pot\_sec)[\text{ if } db \ \mathtt{model\_of} \ \Psi \text{ then } log_0 \not\models \Psi\,]\,. \tag{8}$$

Theorem 2 (confidentiality). *Uniform refusal for unknown potential secrets preserves confidentiality in the sense of Definition 1.*

3.2 Combined Methods for Unknown Policies

We try to adapt the combined method for known potential secrets to unknown such policies, by restricting the censor tests to true secrets only. Due to this restriction, we have to explicitly ensure log consistency. The returned answers are:

```
$ans_i :=$
if $[\, log_{i-1} \cup \{eval^*(\Phi_i)(db)\}$ is inconsistent or
$(\text{exists}\, \Psi_1 \in pot\_sec)[\, db \ \mathtt{model\_of} \ \Psi_1 \text{ and } log_{i-1} \cup \{eval^*(\Phi_i)(db)\} \models \Psi_1]\,]$
then
    if $[\, log_{i-1} \cup \{\neg eval^*(\Phi_i)(db)\}$ is inconsistent or
    $(\text{exists}\, \Psi_2 \in pot\_sec)[\, db \ \mathtt{model\_of} \ \Psi_2 \text{ and } log_{i-1} \cup \{\neg eval^*(\Phi_i)(db)\} \models \Psi_2]\,]$
    then mum
    else $\neg eval^*(\Phi_i)(db)$
else $eval^*(\Phi_i)(db)$
```

Thus the censor for refusals, $refusalcensor^{\mathtt{ps},\mathtt{unknown},C}$, looks like

$$\begin{array}{l} [\, log \cup \{eval^*(\Phi)(db)\} \text{ is inconsistent or} \\ (\text{exists}\, \Psi_1 \in pot\_sec)[\, db \ \mathtt{model\_of} \ \Psi_1 \text{ and } log \cup \{eval^*(\Phi)(db)\} \models \Psi_1\,]\,] \\ \text{and} \\ [\, log \cup \{\neg eval^*(\Phi)(db)\} \text{ is inconsistent or} \\ (\text{exists}\, \Psi_2 \in pot\_sec)[\, db \ \mathtt{model\_of} \ \Psi_2 \text{ and } log \cup \{\neg eval^*(\Phi)(db)\} \models \Psi_2\,]\,]\,; \end{array} \tag{9}$$

and the censor for lies, $lyingcensor^{\mathtt{ps},\mathtt{unknown},C}$, looks like

$$\begin{array}{l}
[\,log \cup \{\,eval^*(\Phi)(db)\}\text{ is inconsistent or}\\
(\text{exists}\,\Psi_1 \in pot\_sec)[\,db\ \mathtt{model\_of}\ \Psi_1 \text{ and } log \cup \{\,eval^*(\Phi)(db)\} \models \Psi_1\,]\,]\\
\ \text{and}\\
[log \cup \{\neg eval^*(\Phi)(db)\}\text{ is consistent and}\\
(\text{for all } \Psi_2 \in pot\_sec)[\text{ if } db\ \mathtt{model\_of}\ \Psi_2 \text{ then } log \cup \{\neg eval^*(\Phi)(db)\} \not\models \Psi_2\,]\,]\,.
\end{array} \tag{10}$$

Furthermore, the $precondition^{\mathtt{ps},\mathtt{unknown},C}(.,log_0)(db, pot\_sec)$ is specified by

$$\begin{array}{l}
log_0 \text{ is consistent and}\\
(\text{for all } \Psi \in pot\_sec)\,[\text{ if } db\ \mathtt{model\_of}\ \Psi \text{ then } log_0 \not\models \Psi\,]\,.
\end{array} \tag{11}$$

Unfortunately, the controlled evaluation method for combined lying and refusal for unknown potential secrets do not preserve confidentiality in the sense of Definition 1.

Example 1. Let $Q = \langle p \vee q \rangle$, $log_0 = \emptyset$, $pot\_sec = \{p \vee q, \neg q\}$ and $db = \{p\}$. Then $ans_1 = \mathtt{mum}$. Now consider any pair (db', pot_sec') satisfying the precondition (11) and returning the same answers as (db, pot_sec). By definition, since $ans_1 = \mathtt{mum}$, $p \vee q$ entails a potential secret Ψ such that $db'\ \mathtt{model\_of}\ \Psi$. Since the vocabulary is $\{p, q\}$ and three out of the four possible interpretations satisfy $p \vee q$, there are two possibilities: either $\Psi \equiv p \vee q$ or Ψ is a tautology. The latter case is not possible because it violates the precondition. It follows that for all the allowed pairs (db', pot_sec') that return $ans_1 = \mathtt{mum}$, $db'\ \mathtt{model\_of}\ p \vee q$, and hence conditions (a) and (b) of Definition 1 cannot be simultaneously satisfied. This proves that the above combined method does not preserve confidentiality.

There are at least two obvious alternative definitions for ans_i, where only Ψ_1 or Ψ_2 (respectively) must be satisfied by db. It can be shown that these two alternatives do not preserve confidentiality, either.

In the light of these negative results, we can assess different methods for *unknown* policies w.r.t. the given state of the art. First, for unknown policies—be they secrecies (see [1]) or potential secrets (see Section 3.1)—the censor for uniform refusal is weaker than the censor for uniform lying. Consequently, we can state a "longest honeymoon lemma" in the sense of [3]. Informally speaking, the lemma says that the method based on uniform refusal does never modify a correct answer *before* the corresponding method based on uniform lies. Of course, this result holds for input pairs that are admissible for both methods. In general, the two preconditions are incomparable.

Second, the censors for uniform refusal should not be further weakened: If we allowed $log \models \Psi$, for some Ψ occurring in the policy such that $db\ \mathtt{model\_of}\ \Psi$, then we could not prevent users from getting a view (given by log) that entails an actual secret. Surely we do not want such a situation. Hence uniform refusal cannot be improved unless some refusals are replaced by lies. However, as shown above, the current attempts at designing combined methods for unknown potential secrets fail to preserve confidentiality.

We conclude that in the framework of unknown policies, the methods based on uniform refusals are currently the best choice from the point of view of the longest honeymoon lemma, and that there are difficulties in improving them.

4 A Systematic View of Controlled Query Evaluation

In this section we develop a more general analysis of the relationships between different policy models and different awareness assumptions. The results of this section justify the recurrent patterns observed in the previous sections.

4.1 Known Policies vs. Unknown Policies

We start with a simple and intuitive result, stating that unknown policies are easier to handle than known policies.

Proposition 3. *If control_eval preserves confidentiality w.r.t. known policies, then control_eval preserves confidentiality w.r.t. unknown policies. The converse is not true.*

The converse of the first part of Proposition 3 holds if the censor of *control_eval* is *instance independent*, that is, for all db and db',

$$censor(\Psi, log, db, policy) = censor(\Psi, log, db', policy)\,.$$

Theorem 3. *If control_eval has an instance independent censor and preserves confidentiality w.r.t. unknown policies, then control_eval preserves confidentiality w.r.t. known policies.*

In other words, if the censor is instance independent, then *control_eval* preserves confidentiality w.r.t. known policies iff it preserves confidentiality w.r.t. unknown policies. Intuitively, this means that in order to take really advantage of the extra degrees of freedom permitted by unknown policies (as suggested by Proposition 3), the censor *must be instance dependent.*

On one hand, this result justifies the difference between the existing censors for known policies and those for unknown policies. On the other hand, this result tells us that even if the instance dependent attempts at defining a secure combined method for unknown policies failed (Section 3.2), nonetheless any significant future approach should keep on searching for an instance dependent censor.

4.2 Potential Secrets vs. Secrecies

In this section we study some general conditions under which the transformation of secrecies into potential secrets (function *pot_sec*(.)) yields a secure query evaluation method. First, the transformation is formalized.

Definition 2. *Let $control\_eval^{\mathtt{ps},a,e}$ be any controlled query evaluation based on potential secrets. The* naive reduction *of secrecies to $control\_eval^{\mathtt{ps},a,e}$ is the function $naive\_red(control\_eval^{\mathtt{ps},a,e}) :=$*

$$\lambda Q, log_0, db, secr.\ control\_eval^{\mathtt{ps},a,e}(Q, log_0)(db, pot\_sec(secr))\,.$$

Let the precondition of the naive reduction be satisfied by $(Q, log_0, db, secr)$ iff the precondition of $control\_eval^{\mathtt{ps},a,e}$ is satisfied by $(Q, log_0, db, pot\_sec(secr))$.

The main theorem needs the following definition.

Definition 3. *Let $censor^{\mathtt{ps},a,e}$ and $precond^{\mathtt{ps},a,e}$ denote the censor and the precondition (respectively) of $control\_eval^{\mathtt{ps},a,e}$. We say that $control\_eval^{\mathtt{ps},a,e}$ is* insensitive to false potential secrets *iff for all Q, log_0, db, pot_sec, and for all sets of sentences S whose members are all false in db,*

$$censor^{\mathtt{ps},a,e}(Q, log_0, db, pot\_sec) = censor^{\mathtt{ps},a,e}(Q, log_0, db, pot\_sec \cup S)$$
$$precond^{\mathtt{ps},a,e}(Q, log_0, db, pot\_sec) = precond^{\mathtt{ps},a,e}(Q, log_0, db, pot\_sec \cup S)\,.$$

Note that all the methods for unknown policies introduced so far are insensitive to false potential secrets (this property causes instance dependence, discussed in the previous subsection). Now we are ready to state the main theorem of this subsection.

Theorem 4. *$naive\_red(control\_eval^{\mathtt{ps},a,e})$ preserves confidentiality (w.r.t. `sec`, a and e, cf. Definition 1) whenever $control\_eval^{\mathtt{ps},a,e}$ satisfies any of the following conditions:*

1. *$a =$ `known` and $control\_eval^{\mathtt{ps},a,e}$ preserves confidentiality (w.r.t. `ps`, a and e).*
2. *$e = L$ (lies), the logs are always consistent and entail no potential secrets, all answers are true in the absence of secrets, and all pairs with an empty policy are admissible.*
3. *$a =$ `unknown`, $control\_eval^{\mathtt{ps},a,e}$ preserves confidentiality and is insensitive to false potential secrets.*

Proof. Part 1 Let $precond^{\mathtt{ps}}$ denote the precondition of $control\_eval^{\mathtt{ps},a,e}$. Consider arbitrary Q, log_0, db_1 and $secr_1$, satisfying the corresponding precondition of $naive\_red(control\_eval^{\mathtt{ps},a,e})$. Let Q' be any finite prefix of Q, and consider $\Theta = \{\Psi, \neg\Psi\} \in secr_1$.

Then, by the definitions, $(db_1, pot\_sec(secr_1))$ satisfies $precond^{\mathtt{ps}}$ and

$$\Psi^* = eval^*(\Psi)(db_1) \in pot\_sec(secr_1). \tag{12}$$

Since, by assumption, confidentiality is preserved for known potential secrets, there exists db_2 such that the following holds:

– $(db_2, pot\_sec(secr_1))$ satisfies $precond^{\mathtt{ps}}$, and thus also $(db_2, secr_1)$ satisfies $precond^{\mathtt{sec}}$.

- $(db_1, pot\_sec(secr_1))$ and $(db_2, pot\_sec(secr_1))$ produce the same answers, and hence so do $(db_1, secr_1)$ and $(db_2, secr_1)$.
- $eval^*(\Psi^*)(db_2) = \neg\Psi^*$, and thus also

$$eval^*(\Psi)(db_2) = \neg\Psi^*. \tag{13}$$

From (12) and (13) we conclude that

$$\{eval^*(\Psi)(db_1), eval^*(\Psi)(db_2)\} = \{\Psi^*, \neg\Psi^*\} = \{\Psi, \neg\Psi\}.$$

Hence confidentiality is preserved for known secrecies. This concludes Part 1.

Part 2 Given Q, log_0, db_1 and $secr_1$, let $\Psi \in pot\_sec(secr_1)$ and Q' be any finite prefix of Q. By assumption, there exists always an instance db_2 that satisfies the last log of $control\_eval^{\mathtt{ps},a,e}(Q', log_0)(db_1, pot\_sec(secr_1))$ and falsifies Ψ. It can be verified by a straightforward induction that this evaluation sequence equals

$$control\_eval^{\mathtt{ps},a,e}(Q', log_0)(db_2, pot\_sec(\emptyset))$$

(because all answers must be true by assumption and db_2 is a model of the original answers by construction). Moreover, the pair $(db_2, pot\_sec(\emptyset))$ is admissible by assumption. It follows that the naive reduction preserves confidentiality. This completes Part 2.

Part 3 (Sketch) It suffices to show that for all finite sequences of queries Q', for all log_0, for all admissible pairs $(db_1, pot\_sec(secr_1))$ for $control\_eval^{\mathtt{ps},a,e}$, and for all $\Psi \in pot\_sec(secr_1)$, there exists an admissible pair $(db_2, pot\_sec(secr_2))$ satisfying conditions (a) and (b) of Definition 1, with $policy_1 = pot\_sec(secr_1))$, $policy_2 = pot\_sec(secr_2))$, and $p = \mathtt{ps}$.

By assumption, $control\_eval^{\mathtt{ps},a,e}$ preserves confidentiality, therefore there exists an admissible pair (db_2, pot_sec_2) satisfying conditions (a) and (b) of Definition 1. Since $control\_eval^{\mathtt{ps},a,e}$ is insensitive to false potential secrets, we can assume, without loss of generality, that all the sentences in pot_sec_2 are satisfied by db_2.

Now let $secr_2 = \{\{\Psi, \neg\Psi\} \mid \Psi \in pot\_sec_2\}$. By construction, the sentences in $pot\_sec(secr_2) \setminus pot\_sec_2$ are all false in db_2, and hence (by assumption) $control\_eval(Q', log_0)(db_2, pot\_sec_2) = control\_eval(Q', log_0)(db_2, pot\_sec(secr_2))$ and $(db_2, pot\_sec(secr_2))$ is admissible. Then $(db_2, pot\_sec(secr_2))$ is an admissible pair that satisfies (a) and (b) of Definition 1, as required. $\square$

It is interesting to note that all the existing methods fall into the three cases of the above theorem.

From Theorem 3 and Theorem 4.(1) we immediately get the following result.

Corollary 1. *If the censor of $control\_eval^{\mathtt{ps},a,e}$ is instance independent and $control\_eval$ preserves confidentiality w.r.t.* ps*, a and e, then the naive reduction $naive\_red(control\_eval^{\mathtt{ps},a,e})$ preserves confidentiality w.r.t. both* sec*,*known*,e, and* sec*,*unknown*,e.*

Remark 1. The above results subsume neither Theorem 1 nor Theorem 2. In fact, in this section we show how methods for potential secrets can be adapted to secrecies (not viceversa), while the gaps of Table 1 concerned only potential secrets.

The naive reduction does not always preserve confidentiality, even if the underlying $\mathit{control\_eval}^{\mathtt{ps},a,e}$ does.

Example 2. Suppose that the vocabulary is $\{p, q\}$, and $\mathit{control\_eval}^{\mathtt{ps},a,e}$ returns always $\mathtt{mum}$ for all possible combinations of Q, log_0, db and $\mathit{pot\_sec}$, with two exceptions, where $\mathit{log}_0 = \emptyset$, and either

– $\mathit{pot\_sec} = \{p \vee \neg q, \neg(p \vee \neg q)\}$, $\mathit{db} = \{q\}$, or
– $\mathit{pot\_sec} = \{\neg(p \vee q)\}$, $\mathit{db} = \emptyset$.

In these two cases, let $\mathit{ans}_i := \mathit{eval}^*(\Psi_i)(\mathit{db})$ if (i) $\mathit{log}_{i-1} \cup \mathit{ans}_i$ does not entail any potential secret, and (ii) both $\{q\}$ and $\emptyset$ are models of $\mathit{log}_{i-1} \cup \mathit{ans}_i$. If these conditions are not simultaneously satisfied, let $\mathit{ans}_i := \mathtt{mum}$.

Finally, the only admissible instances (in all cases) are those that satisfy log_0.

It can be verified that this method preserves confidentiality (hint: the two special cases return the same answers for all query sequences; the other cases are obviously indistinguishable). Now, given $Q = \langle \neg p, q \rangle$, the answers returned for the admissible pair $(\{q\}, \{p \vee \neg q, \neg(p \vee \neg q)\})$ are $\langle \neg p, \mathtt{mum} \rangle$. The only other instance that returns these answers is $\emptyset$ under the policy $\{\neg(p \vee q)\}$, which is *not* closed under negation, i.e., it does not correspond to any set of secrecies. Then the naive reduction does not preserve confidentiality w.r.t. $p = \mathtt{sec}$. □

Note that the above example makes use of a controlled evaluation where (i) the logs are always satisfied by the current instance, and (ii) the answers depend on the false potential secret $p \vee \neg q$. Then the critical assumptions in Theorem 4.(2) (uniform lies) and Theorem 4.(3) (independence from false potential secrets) cannot be simply dropped, because this example would then become a counterexample.

5 Summary and Conclusions

The many works on controlled query evaluation have been given a uniform view in Table 2 and Table 3, and the different notions of confidentiality have been unified by Definition 1. Moreover, we have extended the picture of Table 1 by introducing two new controlled evaluation methods based on lies and refusals for unknown potential secrets (Section 3.1).

Surprisingly, all the attempts at designing secure combined methods for unknown policies failed. The attempts were based on *instance dependent* censors, by analogy with the existing methods for unknown policies. The general results of Section 4.1 tell us that this is the right direction if the extra degree of freedom allowed by unknown policies is to be exploited effectively.

We have introduced a so-called *naive reduction* of secrecies to potential secrets. The naive reduction does not always preserve confidentiality, but we proved in Theorem 4 that confidentiality is guaranteed under fairly natural assumptions. All the known controlled evaluation methods fall into the three points of this theorem, so Table 1 could be entirely reconstructed from the methods based on potential secrets (including the new methods). This fact provides a formal justification of the relationships between different methods empirically observed in Section 2. Thus we achieve a deeper and systematic view of the many forms of controlled query evaluation proposed so far. Moreover, our results show how administrators may freely choose the preferred policy model (secrecies or potential secrets) if the adopted evaluation method is based on potential secrets and fits into one of the cases of Theorem 4. Finer-grained confidentiality requirements can be obtained by mixing secrecies and potential secrets. For each pair of complementary sentences the administrator may decide whether both sentences should be protected, or only one of them should be protected, or none of them needs protection.

We are planning to relate our theoretical studies to the existing approaches to confidentiality, (e.g., polyinstantiation and statistical database perturbation), in order to investigate the practical consequences of our theoretical results. Further future work includes availability policies. Thereby, we hope to get a deeper understanding about the minimality of censors (see for instance [4] or [2], Prop. 4) and the various roles of the user log.

Acknowledgements

We are grateful to the anonymous referees for their careful and cooperative comments and suggestions.

References

[1] Biskup, J.: For unknown secrecies refusal is better than lying, Data and Knowledge Engineering, 33 (2000), pp. 1-23. 40, 41, 44, 46, 47, 49

[2] Biskup, J., Bonatti, P. A. : Lying versus refusal for known potential secrets, Data and Knowledge Engineering, 38 (2001), pp. 199-222. 40, 41, 44, 46, 54

[3] Biskup, J., Bonatti, P. A. : Controlled query evaluation for known policies by combining lying and refusal, Proceedings 2nd Int. Symp. on th Foundations of Information and Knowledge Systems, FoIKS 02, Lecture Notes in Computer Science 2284, Springer, Berlin etc., 2002, pp. 49-66. 40, 41, 46, 49

[4] Bonatti, P. A., Kraus, S., Subrahmanian, V. S.: Foundations of secure deductive databases, IEEE Transactions on Knowledge and Data Engineering 7,3 (1995), pp. 406-422. 40, 41, 46, 54

[5] S. Castano, M. Fugini, G. Martella, P. Samarati: Database Security, Addison-Wesley, 1994. 40

[6] D. E. Denning: Cryptography and Data Security, Addison-Wesley, 1982. 40

[7] Lloyd, J. W.: Foundations of Logic Programming, Springer, 1987. 42

[8] Shoenfield, J. R.: Mathematical Logic, Addison-Wesley, Reading etc., 1967. 42

[9] Sicherman, G. L., de Jonge, W., van de Riet, R. P.: Answering queries without revealing secrets, ACM Transactions on Database Systems 8,1 (1983), pp. 41-59. 40, 41, 46, 47

Cardinality-Based Inference Control in Sum-Only Data Cubes*

Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia

Center for Secure Information Systems, George Mason University
Fairfax, VA 22030-4444, USA
{lwang3,dwijesek,jajodia}@gmu.edu

Abstract. This paper addresses the inference problems in data warehouses and decision support systems such as on-line analytical processing (OLAP) systems. Even though OLAP systems restrict user accesses to predefined aggregations, inappropriate disclosure of sensitive attribute values may still occur. Based on a definition of non-compromiseability to mean that any member of a set of variables satisfying a given set of their aggregations can have more than one value, we derive sufficient conditions for non-compromiseability in sum-only data cubes. Under this definition, (1) the non-compromiseability of multi-dimensional aggregations can be reduced to that of one dimensional aggregations, (2) full or dense core cuboids are non-compromiseable, and (3) there is a tight lower bound for the cardinality of a core cuboid to remain non-compromiseable. Based on these results, taken together with a three-tier model for controlling inferences, we provide a divide-and-conquer algorithm that uniformly divides data sets into chunks and builds a data cube on each such chunk. The union of these data cubes are then used to provide users with inference-free OLAP queries.

1 Introduction

Decision support systems such as On-line Analytical Processing (OLAP) are becoming increasingly important in industry. These systems are designed to answer queries involving large amounts of data and their statistical averages in near real time. It is well known that access control alone is insufficient in eliminating all forms of disclosures, as information not released directly may be inferred indirectly from answers to legitimate queries. This is known as *inference problem*. An OLAP query typically consists of multiple aggregations, and hence vulnerable to unwanted inferences. Providing inference-free answers to sum-only data cube style OLAP queries while not adversely impacting performance or restricting availability in an OLAP system is the subject matter of this paper.

The inference problem has been investigated since 70's and many inference control methods have been proposed for statistical databases. However, most of

* This work was partially supported by the National Science Foundation under grant CCR-0113515.

these methods become computationally infeasible if directly applied to OLAP systems. OLAP applications demand short response times although queries usually aggregate large amounts of data [20]. Because most existing inference control algorithms have run times proportional to the size of queries or aggregated data sets, their impact upon performance renders them impractical for OLAP systems.

While arbitrary queries are common in statistical databases, OLAP queries usually comprise of well-structured operations such as group-by, cross-tab and sub-totals. These operations can conveniently be integrated with various data cube operations, such as slicing-dicing, roll up and drill down [19]. We will show that the limited formats and predictable structures of OLAP queries as well as the multi-dimensional hierarchical data model of OLAP datasets can be exploited to simplify inference control.

Table 1 shows a small two-dimensional data set consisting of monthly employee salaries. Individual salaries should be hidden from users, and hence have been replaced with the symbol ?. The symbol N/a denotes null value for inapplicable combinations of months and employees that are known to users.[1] Assume subtotals are allowed to be released to users through OLAP queries. Inference problem occurs if any of the values represented by symbol ? can be derived from the subtotals. No value in the first two quarters can be inferred, because infinitely many different values may satisfy each ? symbol with the subtotals satisfied. For the third quarter, Mary's salary in September can be inferred from the subtotals of September salaries because she is the only employee with a valid salary for September. For the fourth quarter, by subtracting Bob's and Jim's fourth quarter salaries (\$4300 and \$3000 respectively) from the subtotals in October and November (\$7100 and \$4100 respectively) Alice's salary for October can be computed to be \$3900.

Based on a definition of non-compromiseability to mean that any member of a set of variables satisfying a given set of their aggregations can have more than one value [2], we derive sufficient conditions for non-compromiseability in sum-only data cubes: (1) the non-compromiseability of multi-dimensional aggregations can be reduced to that of one dimensional aggregations, (2) full or dense core cuboids are non-compromiseable, and (3) there is a tight lower bound on the cardinality of a core cuboid for it to remain non-compromiseable. Based on these results, and a three-tier model for controlling inferences, we provide a divide-and-conquer algorithm that uniformly divides data sets into chunks and builds a data cube on each such chunk. The union of these data cubes are then used to provide users with inference-free OLAP queries.

The rest of the paper is organized as follows. Section 2 reviews existing inference control methods proposed in statistical databases and OLAP systems. Section 3 formalizes sum-only data cube and proves sufficient conditions for its

[1] In general, data values are known through various kinds of *external knowledge* (knowledge obtained through channels other than queries.)

[2] In the settings of this paper, each variable can have either one value or infinitely many values.

non-compromiseability. On the basis of a three-tier model these conditions are integrated into an inference control algorithm in Section 4. Section 5 concludes the paper.

2 Related Work

Inference control has been extensively studied in statistical databases as surveyed in [14, 1, 15]. Inference control methods proposed in statistical databases are usually classified into two main categories; *restriction based* techniques and *perturbation based* techniques. Restriction based techniques [18] include restricting the size of a *query set* (i.e., the entities that satisfy a single query), overlap control [16] in query sets, auditing all queries in order to determine when inferences are possible [11, 8, 22, 24], suppressing sensitive data in released tables containing statistical data [12], partitioning data into mutually exclusive sets [9, 10], and restricting each query to range over at most one partition. Perturbation based technique includes adding noise to source data [29], outputs [5, 25], database structure [27], or size of query sets (by sampling data to answer queries) [13]. Some variations of the inference problem have been studied lately, such as the inferences caused by arithmetic constraints [7, 6], inferring approximate values instead of exact values [24] and inferring intervals enclosing exact values [23].

The inference control methods proposed for statistical databases generally sacrifice efficiency for the control of inferences caused by arbitrary queries, which

Table 1. An example data cube

Quarter	Month	Alice	Bob	Jim	Mary	Sub Total
1	January	?	?	?	?	5500
	February	?	?	?	?	5500
	March	?	?	?	?	5500
	Sub Total	3000	3000	4500	6000	
2	April	?	?	?	?	6100
	May	?	N/a	?	?	6100
	June	?	?	?	?	4100
	Sub Total	4500	3300	4500	4000	
3	July	?	?	?	?	6100
	August	?	?	?	?	6100
	September	N/a	N/a	N/a	?	2000
	Sub Total	3500	2200	2500	6000	
4	October	?	?	?	N/a	7100
	November	N/a	?	?	N/a	4100
	December	?	N/a	N/a	?	4100
*	Bonus	?	N/a	N/a	?	6000
	Sub Total	7000	4300	3000	7000	

is essential for general purpose databases. However, this sacrifice is not desirable for OLAP systems, because in OLAP systems near real time response takes priority over the generality of answerable queries. Hence most of these methods are computationally infeasible in OLAP systems. As an example, Audit Expert [11] models inference problem with a linear system $Ax = b$ and detects the occurrence of inference by transforming the m by n matrix A (corresponding to m queries on n attribute values) to its reduced row echelon form. The transformation has a well-known complexity of $O(m^2 n)$, which is prohibitive in the context of data warehouses and OLAP systems since m and n can be as large as a million.

Our work shares similar motivation with that of [16], i.e., to efficiently control inference with the cardinality of data and queries, which can be easily obtained, stored and maintained. Dobkin et al. gave sufficient conditions for the non-compromiseability of arbitrary sum only queries [16]. The conditions are based on the smallest number of queries that suffices to compromise the individual data. Our work addresses multi-dimensional data cube queries. The fact that data cube queries are a special case of arbitrary queries implies better results.

To the best of our knowledge, inference control for OLAP systems and data warehouses are limited to [3, 2, 17, 26]. They all attempt to perturb sensitive values while preserving the data distribution model, such that classification (or association rules) results obtained before and after the perturbation do not change. These approaches are application-specific, that is, the sole purpose of data analysis is limited to classification (or association rule mining). We do not have this restriction. Moreover, we do not use perturbation in this paper.

3 Cardinality-Based Non-compromiseability Criteria for Data Cubes

This section defines our model for sum-only data cubes and formalizes compromiseability. We then derive cardinality-based sufficient conditions for non-compromiseability based on the model and definitions.

3.1 Problem Formulation

In our model, a k-dimensional *data cube* consists of one *core cuboid* and several *aggregation cuboids*. In addition, we use an *aggregation matrix* to abstract the relationship between them. Each *dimension* is modeled as a closed integer interval. The Cartesian product of the k dimensions is referred to as *full core cuboid*. Each integer vector in the full core cuboid is a *tuple*. A *core cuboid* is a subset of the full core cuboid, which contains at least one tuple for every value chosen from every dimension. This condition allows us to uniquely identify the size of each dimension for a given core cuboid. Definition 1 formalizes these concepts.

Definition 1 (Core Cuboids and Slices).

Given a set of k integers D_i satisfying $D_i > 1$ for all $1 \leq i \leq k$. A k-dimensional core cuboid is any subset S of $\Pi_{i=1}^{k}[1, D_i]$ satisfying the property that for any $x_i \in [1, D_i]$ there exist $(k-1)$ integers $x_j \in [1, D_j]$ for all

$1 \leq j \leq k$ and $j \neq i$, such that $(x_1, \ldots x_{i-1}, x_i, x_{i+1}, \ldots x_k) \in S$. C_c denotes a core cuboid. Each vector $t \in C_c$ is referred to as a tuple. Further, the i^{th} element of vector $t \in C_c$, denoted by $t[i]$, is referred to as the i^{th} dimension of t. We say that $\Pi_{i=1}^{k}[1, D_i]$ is the full core cuboid denoted by C_f. We say a tuple t is missing from the core cuboid C_c if $t \in C_f \setminus C_c$. The subset of C_c defined by $\{t \mid t \in C_c, t[i] = j\}$ for each $j \in [1, D_i]$ is said to be the j^{th} slice of C_c on the i^{th} dimension, denoted by $P_i(C_c, j)$. If $P_i(C_c, j) = \{t \mid t \in C_f, t[i] = j, j \in [1, D_i]\}$, we say that $P_i(C_c, j)$ is a full slice.

For example, the fourth quarter salaries in the sample data cube of Table 1 is depicted in Table 2. It has two dimensions: month (dimension 1) and employee name (dimension 2). Both dimensions have four different values that are mapped to the integer interval $[1, 4]$. Therefore, the full core cuboid C_f is $[1, 4] \times [1, 4]$. The core cuboid C_c contains nine tuples and seven missing tuples are shown as N/a.

To define aggregations of a data cube, we follow [19] to augment each dimension with a special value *ALL*, for which we use the symbol *. Each *aggregation vector* is similar to a tuple except that it is formed with the augmented dimensions. An aggregation vector selects a set of tuples in core cuboids with its * values, which form its *aggregation set*. All aggregation vectors having * value in the same dimensions form an *aggregation cuboid*. The concepts of aggregation vector, aggregation cuboid and aggregation set are formalized in Definition 2.

Definition 2 (j-* Aggregation Vectors, Cuboids and Data Cubes). *A j-* aggregation vector t is a k dimensional vector satisfying $t \in \Pi_{i=1}^{k}([1, D_i] \cup \{*\})$ and $\mid \{i : t[i] = * \quad for \quad 1 \leq i \leq k\} \mid = j$. If $t[i] = *$, then we say that the i^{th} element is a *-elements, and others are called non *-elements. A j-* aggregation cuboid is a set of aggregation vectors C such that for any $t, t' \in C$, $\{i : t[i] = *\} = \{i : t'[i] = *\}$ and $\mid \{i : t[i] = *\} \mid = j$. The aggregation set of an j-* aggregation vector t is defined as $\{t' : t' \in C_c$ such that $t'[i] = t[i], \forall i\, t[i] \neq *\}$. We use the notation $Qset(t)$ for the aggregation set of t. The aggregation set of a set of aggregation vectors S is defined as the union of $Qset(t)$ for all $t \in S$. We use notation $Qset(S)$ for the aggregation set of S. A data cube is defined as a pair $< C_c, S_{all} >$, where C_c is a core cuboid, and S_{all} is the set of all j-* aggregation cuboids, for all $1 \leq j \leq k$.*

For example, subtotals of the fourth quarter data in the data cube of Table 1 is depicted in Table 2. Each subtotal is represented as an aggregation vector with * value. For example, $(1, *)$ represents the subtotal in October. The aggregation set of $(1, *)$ is {(1,1), (1,2), (1,3)}. The set of four aggregation vectors $\{(1, *), (2, *), (3, *), (4, *)\}$ form an aggregation cuboid because all of them have * value in the second dimension.

To abstract the relationship between a core cuboid and its aggregation cuboids in a given data cube, we define a binary matrix referred to as *aggregation matrix*. Each element of an aggregation matrix is associated with a tuple and an aggregation vector. An element 1 means the tuple is contained in the aggregation set of the aggregation vector, 0 otherwise. We assign the tuples in C_f

Table 2. Illustration of data cube

	1 (Al)	2 (Bob)	3 (Jim)	4 (Ma)	5 (SubT)
1 (Oct)	(1,1)	(1,2)	(1,3)	N/a	(1,*)
2 (Nov)	N/a	(2,2)	(2,3)	N/a	(2,*)
3 (Dec)	(3,1)	N/a	N/a	(3,4)	(3,*)
4 (Bonus)	(4,1)	N/a	N/a	(4,4)	(4,*)
5 (SubT)	(*,1)	(*,2)	(*,3)	(*,4)	(*,*)

and any C in dictionary order, the aggregation cuboids in S_{all} in ascending order on the number of *-elements and descending order on the index of the *-element. This assignment enables us to refer to the i^{th} tuple in C_f as $C_f[i]$ (similarly for C_c, S_{all} or their subsets). We use $M[i,j]$ for the $(i,j)^{th}$ element of matrix M. Aggregation matrix is formalized in Definition 3.

Definition 3 (Aggregation Matrix).
The aggregation matrix of the aggregation cuboid C on the core cuboid C_c is defined as the following $(m \times n)$ matrix $M_{C_c,C}$ (or simply M when C_c and C are clear from context).

$$M_{C_c,C}[i,j] = \begin{cases} 1, \text{ if } C_f[j] \in Qset(C[i]); \\ 0, \text{ otherwise.} \end{cases}$$

We define the aggregation matrix of S on C_c as the row block matrix with the i^{th} row block as the aggregation matrix of the i^{th} aggregation cuboid in S.

We use S_1 to represent the set of all 1- aggregation cuboids for a given C_c, and M_1 the aggregation matrix of S_1 on C_c (that is M_{C_c,S_1}), referred to as the 1-* aggregation matrix.*

Aggregation matrix and compromiseability are illustrated in Table 3. By representing individual salaries with variables x_i, we get a linear system $M_{C_c,S_1} \cdot \overrightarrow{X} = \overrightarrow{B}$. It has at least one solution, since $\overrightarrow{B}$ is caculated from the "real" salaries, which must satisfy the stated linear system. From linear algebra [21], each x_i can have either a unique value or infinitely many different values among all the solutions to $M_{C_c,S_1} \cdot \overrightarrow{X} = \overrightarrow{B}$. This depends on M_{C_c,S_1} but not on $\overrightarrow{B}$ (this is not valid if additional knowledge about $\overrightarrow{X}$ is known to users, for example, salaries are non-negative [23, 24, 22]). If an x_i has a unique value among all the solutions, then clearly the sensitive value represented by x_i was compromised. In this example, x_1 has the value of 3900 in any solution, so Alice's salary for October is compromised. In Definition 4, we formalize the definition of compromiseability. We distinguish between two cases, that is, the trivial case illustrated by the third quarter data of Table 1, and the complementary case illustrated by the fourth quarter data.

Table 3. Equations formulating the disclosure of the core cuboid given in Table 2

	1 (Alice)	2 (Bob)	3 (Jim)	4 (Mary)	5 (Sub Total)
1 (Oct)	x_1	x_2	x_3	N/a	7100
2 (Nov)	N/a	x_4	x_5	N/a	4100
3 (Dec)	x_6	N/a	N/a	x_7	4100
4 (Bonus)	x_8	N/a	N/a	x_9	6000
5 (Sub Total)	7000	4300	3000	7000	-

$$\begin{pmatrix} 1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \\ 0 \\ x_4 \\ x_5 \\ 0 \\ x_6 \\ 0 \\ 0 \\ x_7 \\ x_8 \\ 0 \\ 0 \\ x_9 \end{pmatrix} = \begin{pmatrix} 7100 \\ 4100 \\ 4100 \\ 6000 \\ 7000 \\ 4300 \\ 3000 \\ 7000 \end{pmatrix}$$

Definition 4 (Compromiseability).
Given a data cube $< C_c, S_{all} >$ and a set of aggregation cuboids $S \subseteq S_{all}$, B is arbitrarily chosen such that $M_{C_c,S}.\overrightarrow{X} = \overrightarrow{B}$ has at least one solution. S is said to compromise C_c, if at least one component x_i of $\overrightarrow{X}$ has the same value among all the solutions to $M_{C_c,S}.\overrightarrow{X} = \overrightarrow{B}$.

1. *C_c is trivially compromised by S if there is an integer $i \in [1, m]$ such that the i^{th} row of $M_{C_c,S}$ is e_j. Here $1 \leq j \leq n$.*
2. *C_c is non-trivially compromised by S if C_c is not trivially compromised by S.*

It is well-known that C_c is compromised by S if and only if there exists at least one unit row vector e_i (where $e_i[i] = 1$ and $e_i[j] = 0$ for $j \neq i$) in any reduced row echelon form of $M_{C_c,S}$ [21]. This yields an alternative definition of compromiseability which we shall use in the rest of this paper.

3.2 Trivial Compromiseability

In this section, we derive cardinality-based criteria of non-compromiseability in the trivial case. We have two results. Firstly, full core cuboids cannot be trivially compromised. The second is an upper bound on the cardinality of the core cuboid such that it is trivially compromised by the set of all 1-* aggregation cuboids. They are stated and proved in Theorem 1.

Theorem 1. *1. A full core cuboid C_f cannot be trivially compromised by any set of aggregation cuboids S.*
2. C_c is trivially compromised by S_1 if $|C_c| < 2^{k-1} \cdot max(D_1, D_2, \ldots, D_k)$ for $k \geq 2$

Proof: Given in [30] (ommitted here due to space limitation).

Theorem 1 provides cardinality-based criteria for the two extreme cases, i.e., the core cuboid is either full or sparse. However, cardinality-based criteria are ineffective for cases in between. As an example, consider the third quarter data in Table 1, which is trivially compromised. Without changing the cardinality, evenly distributing the three "N/a" in three months makes the core cuboid free of trivial compromise. This invalidates any cardinality based criteria because trivial compromiseability varies for core cuboids with exactly the same cardinality.

3.3 Non-trivial Compromiseability

In this section, we derive cardinality-based criteria for determining compromiseability in the non-trivial case. We have two results. The first is that full core cuboids cannot be non-trivially compromised. The second is a lower bound on the cardinality of the core cuboid such that it remains safe from non-trivial compromise. First we have Lemma 1.

Lemma 1. *1. C_c can not be non-trivially compromised by any single cuboid.*
2. If C_c cannot be compromised by S_1, then it cannot be compromised by S_{all}.
3. For any integers k and $D_1, D_2, \ldots, D_k$ that satisfy $D_i \geq 4$ for $1 \leq i \leq k$, there is a k-dimensional data cube $< C_c, S_{all} >$, with integer boundaries D_i, such that C_c is non-trivially compromised by S_1.

Proof: Given in [30] (ommitted here due to space limitation).

Because of the second claim of Lemma 1, it is sufficient to safeguard the core cuboid from the collection of 1-* aggregation cuboids. The last condition in Lemma 1 shows that it is impossible to obtain a criteria for preventing non-trivial compromiseability by only looking at the dimension cardinalities.

Theorem 2 (Non-trivial Compromiseability).

1. *C_f cannot be non-trivially compromised by S_1.*
2. *For any integers k and $D_i \geq 4$, there exists a k-dimensional data cube $< C_c, S_{all} >$ satisfying $|C_f - C_c| = 2D_l + 2D_m - 9$ such that C_c is non-trivially compromised by S_1, where D_l and D_m are the least two among D_i.*
3. *If $|C_f - C_c| < 2D_l + 2D_m - 9$, then C_c cannot be non-trivially compromised.*

Proof: See the Appendix.

The first claim in Theorem 2 guarantees the non-trivial compromiseability of full core cuboid. Second and third claims give a tight lower bound on the cardinality of a core cuboid for it to remain free of non-trivial compromises. The second claim also implies that no cardinality based criteria can be derived for core cuboids with a cardinality below the computed lower bound.

Corollary 1 (Non-trivial Compromiseability).
If for any $i \in [1, k]$, there exists $j \in [1, D_i]$ such that $|P_i(C_f, j) - P_i(C_c, j)| = 0$, C_c cannot be non-trivially compromised.

Proof: Follows from the proof of Theorem 2. □

By Corollary 1, full slice on every dimension guarantees non-compromiseability in the non-trivial case.

4 A Cardinality-Based Inference Control Algorithm for Data Cubes

This section describes an algorithm to control inferences in data cube style OLAP queries using the results on compromiseability developed in Section 3. Our algorithm is based on a three-tier model consisting of core data, pre-computed aggregations and answerable queries.

4.1 Three-Tier Inference Control Model

Our three-tier model consists of three basic components and two abstract relations in between as given below and illustrated in Figure 1. In addition we enforce three properties on the model.

1. **Three Tiers:**
   (a) A set of data items D.
   (b) A set of aggregations A.
   (c) A set of queries Q.
2. **Relations Between Tiers:**
   (a) $R_{AD} \subseteq A \times D$.
   (b) $R_{QA} \subseteq Q \times A$.
3. **Properties:**
   (a) $|A| << |Q|$.
   (b) There are partitions P_D on D and P_A on A, such that for any $(a, d) \in R_{AD}$ and $(a', d') \in R_{AD}$, d and d' are in the same chunk of P_D if and only if a and a' are in the same chunk of P_A.
   (c) D is not compromised by A.

Three-tier inference control model simplifies inference control problem in several ways. Firstly, because all queries in Q are derived from aggregations in A, it suffices to ensure the non-compromiseability A instead of Q. This reduces the complexity of inference control due to the first characteristic of A. Secondly, the second characteristic of A allows us to adopt a divide-and-conquer approach to further reduce the complexity of inference control. Thirdly, inference control is embedded in the off-line design of A and R_{AD}, so the overhead of on-line inference control is eliminated or reduced. Although the restriction of Q to be derived from A reduces the total number of answerable queries, A can be designed in such a way that it contains most semantics required by the application. Hence the restricted queries are mostly arbitrary and meaningless with respect to application requirements.

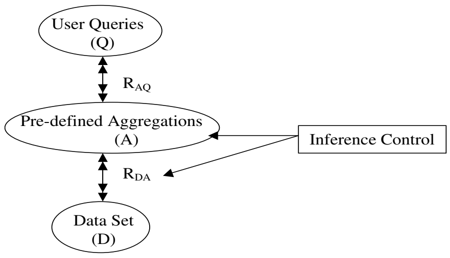


Fig. 1. Three-tier model for controlling inferences

```
Algorithm Ctrl_Inf_Cube
Input: Data Cube $< C_c, S_{all} >$, integers $D_i^j \in [1, D_i]$ for $1 \leq i \leq k$ and $0 \leq j \leq m_i$, where
        $m_i$ is fixed for each $i$ and $D_i^j$ satisfy $D_i^0 = 1$, $D_i^{m_i} = D_i$, and $D_i^{j-1} < D_i^j$ for all
        $1 \leq j \leq m_i$.
Output: a set of aggregations $S_A$ that do not compromise $C_c$
Method:
    1. Let $S_A = \phi$;
    2. For each $k$ dimensional vector $v \in \prod_{i=1}^{k}[1, m_i]$
           Let $C_{tmp} = \{t : t \in C_c, \forall i \in [1,k]\ \ t[i] \in [D_i^{v[i]-1}, D_i^{v[i]}]\}$;
           Let $C_p = \{t : \exists\, t' \in C_{tmp}, \forall i \in [1,k]\ \ t[i] = t'[i] - D_i^{v[i]-1} + 1\}$;
           Let $S_A = S_A \cup$ Ctrul_Inf_Chunk( $C_p, [1, D_i^{v[i]} - D_i^{v[i]-1} + 1]$);
    3. Return $S_A$.
```

```
Subroutine Ctrl_Inf_Chunk
Input: $k$ dimensional core cuboid $C'_c$ and the $k$ dimension domains $[1, D'_i], i = 1, 2, \ldots, k$
Output: $S'_{all}$ if it does not compromise $C'_c$ according to the cardinality-based criteria,
$\phi$ otherwise, where $S'_{all}$ is the set of all aggregation cuboids defined on $C'_c$
Method:
    1. If $|C'_c| = 0$ or $|C'_c| = |C'_f|$, where $C'_f$ is the full core cuboid of $C'_c$
           Return $S'_{all}$;
    2. If $C'_c$ is trivially compromised by $S'_{all}$
           Return $\phi$;
    3. Let $D_l$, $D_m$ be the two smallest among $D'_i$;
    4. If $| C'_f - C'_c | < 2D_l + 2D_m - 9$
           Return $S'_{all}$;
    5. If for all $i \in [1,k]$ there exists $j \in [1, D'_i]$ such that $| P_i(C'_f, j) - P_i(C'_c, j) | = 0$
           Return $S'_{all}$;
    6. Return $\phi$.
```

Fig. 2. Inference control algorithm for data cubes

4.2 Inference Control Algorithm

The inference control algorithm shown in Figure 2 applies the results given in Section 3 using our three-tier model. The algorithm first partitions the core cuboid into disjoint chunks, each of which is then passed to the subroutine *Ctrl_Inf_Chunk*. The subroutine checks the non-compromiseability of the *sub-*

data cube defined on this chunk of data, using the cardinality based criteria. If it is compromised the subroutine returns an empty set, indicating no aggregation is allowed on the data. Otherwise, the subroutine returns the set of all aggregation cuboids of the sub-data cube. The final outcome is the union of all partial results returned by the subroutine (this set of aggregations can then be used to answer data cube style OLAP queries without inference problem).

Correctness The correctness of the algorithm, that is, the non-compromiseability of the final result is straight-forward. The subroutine Ctrl_Inf_Chunk guarantees the non-compromiseability of each sub-data cube respectively. In addition, the sub-data cubes are disjoint, making the non-compromiseability of each of them independent of others.

Runtime Analysis: The main routine of the algorithm partitions C_c by evaluating the k dimensions of each tuple. Let $n = |C_c|$, so the runtime of the main routine is $O(nk)$=$O(n)$ (suppose k is fixed with respect to n). The subroutine *Ctrl_Inf_Chunk* is called for each of the $N = \prod_{i=1}^{k} m_i$ chunks (m_i are defined in the algorithm). It evaluates the cardinality of each input chunk C'_c, which has the same complexity as establishing its 1-* aggregation matrix M'_1.

Let $n' = \prod_{i=1}^{k} D'_i$ be the number of columns in M'_1 (D'_i are defined in the algorithm), then $m' = n' \sum_{i=1}^{k} \frac{1}{D'_i}$ is the number of rows. Let D_i^{max} be the maximum value among D'_i. Out of the $(m'n')$ elements, $O(m' \cdot D_i^{max})$ elements must be considered to compute M'_1. Suppose $(\sum_{i=1}^{k} \frac{1}{D'_i})D_i^{max} = O(k)$. Then the runtime of the subroutine is $O(k \cdot \prod_{i=1}^{k} D'_i)$. It is called N times so the total runtime is $O(k \cdot \prod_{i=1}^{k} m_i \cdot \prod_{i=1}^{k} D'_i) = O(k \cdot \prod_{i=1}^{k} m_i \cdot \prod_{i=1}^{k} \frac{D_i}{m_i})$, which is $O(k \cdot \prod_{i=1}^{k} D_i) = O(n)$. We note that by definition, determining non-compromiseability has a complexity of $O(n^3)$ and the maximum non-compromiseable subset of aggregations cannot be found in polynomial time [11].

Enhancing the Algorithm: The algorithm demonstrates a simple application of the cardinality based criteria in Section 3, which can be improved in many aspects. The dimension hierarchies inherent to most OLAP datasets can be exploited to increase the semantics included in the final output of the algorithm. For example, assume time dimension has a hierarchy comprised of day, week, month and year. Instead of partitioning the dataset arbitrarily, each week can be used for a chunk. Queries about weeks, months and years can then be answered with only the aggregations in algorithm output.

Notice that the key to cardinality-based non-compromiseability is that each chunk in the partition of a core cube must be either empty or dense (full). The row shuffling [4] technique proposed by Barbara et al. increases the subspace density of data sets by shuffling tuples along those categorical, unordered dimensions. Row shuffling can be integrated into the inference control algorithm as a pre-processing step prior to partitioning.

Data Cube Operations: We briefly discuss how our algorithm may address common data cube operations such as slicing, dicing, rolling up, drilling down and range query. Slicing, dicing and range query require aggregations to be defined on a subspace formed by intervals in dimension domains. Our algorithm also partitions the data set into small chunks. Therefore, in order to enhance our algorithm to address these operations, the subspace required by these data cube operations should be formed as the union of multiple chunks. Rolling up and drilling down require aggregations to be defined at different granularities than those in the original data cube. Rolling up does not directly create an inference threat because with coarser granulated queries include less information about individual data. Our ongoing work is addressing these details.

Although update operations are uncommon in decision support systems, data stored in data warehouses need to be updated over time. Our algorithm is suitable for update operations in two aspects. Firstly, changing values has no effect on cardinality, which determines the non-compromiseability in our algorithm. Secondly, because we have *localized* protection by partitioning data set into small disjoint chunks, the effect of an insertion or deletion is restricted to only the chunks containing updated tuples.

5 Conclusions

Based on a definition of non-compromiseability to mean that each unknown sensitive variable has more than one choices of value to fit a given set of their aggregations, we have derived sufficient conditions for non-compromiseability in sum-only data cubes. We have proved that full core cuboids can not be compromised, and that there is a tight lower bound on the cardinality of a non-compromiseable core cuboid. To apply our results to the inference control of data cube style OLAP queries, we have shown a *divide and conquer* algorithm based on a three-tier model. Future work includes enhancing our results and algorithm to include data cube operations and consider other variations of OLAP queries.

References

[1] N. R. Adam and J. C. Wortmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989. 57

[2] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001. 58

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 439–450, 2000. 58

[4] D. Barbará and X. Wu. Using approximations to scale exploratory data analysis in datacubes. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 382–386, 1999. 65

[5] L. L. Beck. A security mechanism for statistical databases. *ACM Trans. on Database Systems*, 5(3):316–338, 1980. 57

[6] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Trans. Knowledge and Data Engineering*, 12(6):900–919, 2000. 57

[7] A. Brodsky, C. Farkas, D. Wijesekera, and X. S. Wang. Constraints, inference channels and secure databases. In *the 6th International Conference on Principles and Practice of Constraint Programming*, pages 98–113, 2000. 57

[8] F. Y. Chin, P. Kossowski, and S. C. Loh. Efficient inference control for range sum queries. *Theoretical Computer Science*, 32:77–86, 1984. 57

[9] F. Y. Chin and G. Özsoyoglu. Security in partitioned dynamic statistical databases. In *Proc. of IEEE COMPSAC*, pages 594–601, 1979. 57

[10] F. Y. Chin and G. Özsoyoglu. Statistical database design. *ACM Trans. on Database Systems*, 6(1):113–139, 1981. 57

[11] F. Y. Chin and G. Özsoyoglu. Auditing and inference control in statistical databases. *IEEE Trans. on Software Engineering*, 8(6):574–582, 1982. 57, 58, 65

[12] L. H. Cox. Suppression methodology and statistical disclosure control. *Journal of American Statistic Association*, 75(370):377–385, 1980. 57

[13] D. E. Denning. Secure statistical databases with random sample queries. *ACM Trans. on Database Systems*, 5(3):291–315, 1980. 57

[14] D. E. Denning and P. J. Denning. Data security. *ACM computing surveys*, 11(3):227–249, 1979. 57

[15] D. E. Denning and J. Schlörer. Inference controls for statistical databases. *IEEE Computer*, 16(7):69–82, 1983. 57

[16] D. Dobkin, A. K. Jones, and R. J. Lipton. Secure databases: protection against user influence. *ACM Trans. on Database Systems*, 4(1):97–106, 1979. 57, 58

[17] A. Evfimievski, R. Srikant, , R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proceedings of the 8th Conference on Knowledge Discovery and Data Mining (KDD'02)*, 2002. 58

[18] L. P. Fellegi. On the qestion of statistical confidentiality. *Journal of American Statistic Association*, 67(337):7–18, 1972. 57

[19] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, crosstab and sub-totals. In *Proceedings of the 12th International Conference on Data Engineering*, pages 152–159, 1996. 56, 59

[20] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 205–227, 1996. 56

[21] K. Hoffman. *Linear Algebra*. Prentice-Hall, 1961. 60, 61

[22] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Auditing boolean attributes. In *Proc. of the 9th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 86–91, 2000. 57, 60

[23] Y. Li, L. Wang, X. S. Wang, and S. Jajodia. Auditing interval-based inference. In *Proceedings of the 14th Conference on Advanced Information Systems Engineering (CAiSE'02)*, 2001. 57, 60

[24] F. M. Malvestuto and M. Moscarini. Computational issues connected with the protection of sensetive statistics by auditing sum-queries. In *Proc. of IEEE Scientific and Statistical Database Management*, pages 134–144, 1998. 57, 60

[25] J. M. Mateo-Sanz and J. Domingo-Ferrer. A method for data-oriented multivariate microaggregation. In *Proceedings of the Conference on Statistical Data Protection'98*, pages 89–99, 1998. 57
[26] S. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th Conference on Very Large Data Base (VLDB'02)*, 2002. 58
[27] J. Schlörer. Security of statistical databases: multidimensional transformation. *ACM Trans. on Database Systems*, 6(1):95–112, 1981. 57
[28] R. P. Tewarson. *Sparse Matrices*. Academic Press, 1973. 69
[29] J. F. Traub, Y. Yemini, and H. Woźniakowski. The statistical security of a statistical database. *ACM Trans. on Database Systems*, 9(4):672–679, 1984. 57
[30] L. Wang, D. Wijesekera, and J. Sushil. Cardinality-based inference control in sum-only data cubes (extended version). Technical Report, 2002. Available at http://ise.gmu.edu/techrep/. 62

Appendix

Proof(Theorem 2):

1. Without loss of generality, we show that $t_0 = (1, 1, \dots, 1)$ cannot be nontrivially compromised by S_1. Let $C'_f = \{t : \forall i \in [1,k], t[i] = 1 \vee t[i] = 2\}$. Since $(D_1, D_2, \dots, D_k) > 1$, we have that $C'_f \subseteq C_f$ and $|C'_f| = 2^k$. It suffices to prove that t_0 cannot be compromised by S_1 in the data cube $< C'_f, S_{all} >$. Let M'_1 be the 1-* aggregation matrix of S_1 on C'_f. According to Definition 3, there are 2^k non-zero column vectors in M'_1, corresponding to the 2^k tuples in C'_f. In the rest of the proof we formally show that each of the 2^k non-zero column vectors can be represented by the linear combination of the left $2^k - 1$ column vectors. Then, it follows from linear algebra that t_0 cannot be compromised by S_1 in data cube $< C'_f, S_{all} >$ (and hence in $< C_f, S_{all} >$. In order to prove our informally stated claim, we define the *sign assignment vector* as an n dimensional column vector t_{sign} where n is $|C_f|$, as follows:
   - $t_{sign}[1] = 1$
   - $t_{sign}[2^i + j] = -t_{sign}[j]$ for all $0 \le i \le k-1$ and $1 \le j \le 2^i$
   - $t_{sign}[j] = 0$ for all $j > 2^k$

   Claim: $M'_1 \cdot t_{sign} = 0$, where 0 is the n dimensional zero column vector.
   Justification:

   Let $t = S_1[i]$, $t[l] = *$ for $l \in [1,k]$.
   Let v be $M'_1[i, -]$.
   Suppose $t[j] = 1$ or $t[j] = 2$ for all $j \ne l$.
   Then $|Qset(t)| = 2$, and as a consequence we get $Qset(t) = \{t_1, t_2\}$
   where $t_1, t_2 \in C_f$, $t_1[l] = 1$,$t_1[l] = 2$
   and $t_1[j] = t_2[j] = t[j]$ for all $j \ne l$
   Hence, there are two integers $j_1, j_2 \in [1, n]$ satisfying
   $v[j_1] = v[j_2] = 1$ and $v[j] = 0$ for any $j \ne j_1, j_2$.
   By Definition 3, $M'_1[-, j_1]$ (the j_1^{th} column of M'_1) and $M'_1[-, j_2]$

correspond to t_1 and t_2 respectively.
Because C'_f is formed in dictionary order, we get $j_2 = j_1 + 2^{l-1}$.
Hence, we have $v \cdot t_{sign} = 0$.
Otherwise, $|Qset(t)| = 0$; and hence $Qset(t) = \phi$.
Hence, $v = 0$, and hence, $0 \cdot t_{sign} = 0$.
This justifies our claim. Hence, as stated earlier, the justification concludes the main proof.

2. Without loss of generality we assume D_1, D_2 are the least two among D_i's. For an arbitrary but fixed value of D_2, we show by induction on D_1 that C_c as constructed in the proof of Lemma 1 satisfies $|C_f - C_c| = 2D_1 + 2D_2 - 9$.
Inductive Hypothesis: C_c as constructed in the proof of Lemma 1 satisfies:
   - $|C_f - C_c| = 2j + 2D_2 - 9$ for any $j \geq 4$.
   - $|\ P_1(C_f, j) - P_1(C_c, j)\ | = 2$ for any $j \in [1, D_1]$.

   Base Case: In the base case of the proof of Lemma 1, the core cuboid C_c satisfies $|C_f - C_c| = 2D_1 + 2D_2 - 9$. Notice that the core cuboid, $D_1 = 4$, and $|\ P_1(C_f, j) - P_1(C_c, j)\ | = 2$. This validates the base case of our inductive hypothesis.
Inductive Case: Suppose for $D_1 = j$ we have $|C_f - C_c| = 2j + 2D_2 - 9$ and $|\ P_1(C_f, j) - P_1(C_c, j)\ | = 2$. Let C'_f be the full core cuboid corresponding to C'_c for $D_1 = j+1$. By the definition of C in the proof of Lemma 1, we have $|C| = |P_1(C_c, j)|$ and as a consequence $|C'_f - C'_c| = |C_f - C_c| + 2 = 2(j+1) + 2D_2 - 9$. Since $P_1(C'_c, j+1) = C$. Hence, $|\ P_1(C'_f, j) - P_1(C'_c, j)\ | = 2$. This validates the inductive case of our inductive argument and consequently concludes our proof of the tightness of the cardinality lower bound for avoiding nontrivial compromiseability.
3. **Lower Bound:** We show that if C_c is nontrivially compromised then we have $|C_f - C_c| \geq 2D_1 + 2D_2 - 9$. First we make following assumptions.
   (a) The tuple $t = (1, 1, \ldots, 1) \in C_c$ is nontrivially compromised by S_1
   (b) No tuple in C_c is trivially compromised
   (c) There exists $S \subseteq S_1$ such that S non-trivially compromises t, but for any $C \in S$, $S \setminus C$ does not non-trivially compromise t
   (d) For any $t' \in C_f \setminus C_c$, t cannot be nontrivially compromised by S_1 in data cube $< C_c \cup \{t'\}, S_{all} >$.

   Assumption (b) holds by Definition 4. Assumption (c) is reasonable considering the case S contains only a single aggregation cuboid. Assumption (d) is reasonable considering the case $|C_f \setminus C_c| = 1$.
Claim: Suppose Assumption (a),(b),(c), and (d) hold. Furthermore assume that there is a $C \in S$ where $t \in C$ satisfies $t[i] = *$. Then $|P_i(C_f, 1) - P_i(C_c, 1)| \geq 1$, and $|P_i(C_f, j) - P_i(C_c, j)| \geq 2$ holds for any $j \in [2, D_i]$.
Justification: The proof is by contradiction. Without loss of generality, we only justify the claim for $i = k$ and $j = 2$. That is, given a $C \in S$ satisfying $t[k] = *$ for any $t \in C$, we prove that $|P_k(C_f, 2) - P_k(C_c, 2)| \geq 2$.
First we transform the aggregation matrix of S on C_c by row permutation into a singly bordered block diagonal form (SBBDF) [28], denoted by $M_{m \times n}$. The i^{th} diagonal block of M corresponds to $P_k(C_c, i)$ and $\{t :\ t \in S \backslash C, t[k] =$

$i\}$, and the border of M denotes the aggregation cuboid C. We call the columns of M corresponding to the i^{th} diagonal block as the i^{th} *slice of* M. Due to Assumption (a), there exists a row vector a satisfying $a \cdot M = e_1$. Let r_i be $M[i, -]$ then we get $e_1 = \sum_{i=1}^{m} a[i] \cdot r_i$. Suppose each diagonal block of M has size $m' \times n'$. Use r_i^j, for $1 \leq j \leq D_k$ to represent the row vector composed of the elements of r_i that falls into the j^{th} slice of M. Notice that there are n' elements in r_i^j. We also use e_1' and $0'$ to represent the n' dimensional unit row vector and n' dimensional zero row vector, respectively. Then the following are true:
i. $e_1' = \sum_{i=1}^{m'} a[i] r_i^1 + \sum_{i=m-m'+1}^{m} a[i] r_i^1$
ii. $0' = \sum_{i=m'+1}^{2m'} a[i] r_i^2 + \sum_{i=m-m'+1}^{m} a[i] r_i^2$
First we suppose $|P_k(C_f, 2) - P_k(C_c, 2)| = 0$, that is, the second slice of M contains no zero column. We then derive contradictions to our assumptions. Since $|P_k(C_f, 2) - P_k(C_c, 2)| = 0$ the first slice of M contains no more non-zero columns than the second slice of M does. Intuitively if the latter is transformed into a zero vector then applying the same transformation on the former leads to a zero vector, too. This is formally represented as:
iii. $0' = \sum_{i=1}^{m'} a[m' + i] r_i^1 + \sum_{i=m-m'+1}^{m} a[i] r_i^1$.
Subtracting (iii) from (i) gives $e_1' = \sum_{i=1}^{m'} (a[i] - a[m' + i]) r_i^1$. That implies C_c is nontrivially compromised by $S \setminus \{C_k\}$, contradicting Assumption (c). Thus $P_k(C_f, 2) - P_k(C_c, 2)| \neq 0$.
Next we assume $|P_k(C_f, 2) - P_k(C_c, 2)| = 1$ and derive a contradiction to our assumptions.
First the row vector r_i^3 satisfies the following condition:
iv. $0' = \sum_{i=2m'+1}^{3m'} a[i] r_i^3 + \sum_{i=m-m'+1}^{m} a[i] r_i^3$.
Let $t' \in P_k(C_f, 2) \setminus P_k(C_c, 2)$. Notice that (i), (ii) still hold. Suppose t' corresponds to $M[-, y] = 0$. Now assume we add t' to $P_k(C_c, 2)$, consequently we have $M[-, y] \neq 0$. Due to Assumption (d), we have that the left side of (ii) becomes e_1', that is, $a \cdot M[-, y] = 1$. There is also an extra 1-element $M[x, y]$ in the border of M.
Now let t'' be the tuple corresponding to $M[-, y+n']$ in the third slice of M. Suppose $t'' \in P_k(C_c, 3)$ and consequently $M[-, y + n'] \neq 0$. We have that $M[-, y + n'] = M[-, y]$ and consequently $a \cdot M[-, y + n'] = 1$.
By removing t' from $P_k(C_c, 2)$ we return to the original state that all our assumption hold. Now we show by contradiction that $t'' \in P_k(C_c, 3)$ cannot hold any longer. Intuitively, since t' is the only missing tuple in the second slice of M, the third slice of M contains no more non-zero vectors than the second slice of M does, except t''. Because $a \cdot M[-, y + n'] = 1$, elements of a transform the second slice of M to a zero vector, as shown by (ii), also transform the third slice of M to a unit vector. This is formally represented in (v):
v. $e'' = \sum_{i=2m'+1}^{3m'} a[i - m'] r_i^3 + \sum_{i=m-m'+1}^{m} a[i] r_i^3$
Subtracting (iv) from (v) we get that $e'' = \sum_{i=2m'+1}^{3m'} (a[i - m'] - a[i]) r_i^3$; implying C_c is compromised by $S \setminus \{C_i\}$. Hence, Assumption (c) is false. Consequently, $t'' \notin C_c$.

Similar proof exists for the i^{th} slice of C_c, where $i = 4, 5, \ldots, D_k$. However, $M[x, -] \neq 0$ because if so, we can let a_x be zero and then decrease the number of missing tuples in C_c, contradicting Assumption (d). Hence $M[x, -]$ is a unit vector with the 1-element in the first slice of M. However, this further contradicts Assumption (b), that no trivial compromise is possible. Hence we have that $|P_k(C_f, 2) - P_k(C_c, 2)| = 1$ is false.
Now consider $|P_k(C_f, 1) - P_k(C_c, 1)|$. Suppose all the assumptions hold, and $|P_k(C_f, 1) - P_k(C_c, 1)| = 0$. Let $t_1, t_2 \in P_k(C_f, 2) \setminus P_k(C_c, 2)$. Now define $C'_c = C_c \setminus \{t\} \cup \{t_1\}$ and M' be the aggregation matrix of S on C'_c. From $a \cdot M = e_1$, and Assumption (d) we get $a \cdot M' = e_i$, where $M[-, i]$ corresponds to t_1. This implies the nontrivially compromise of t_1 in $< C'_c, S_{all} >$, with $|P_k(C_f, 1) - P_k(C'_c, 1)| = 1$, which contradicts what we have already proved. Hence, we get $|P_k(C_f, 1) - P_k(C_c, 1)| \geq 1$. This concludes the justification of our claim.
The claim implies that the number of missing tuples in C_c increases monotonically with the following:

- The number of aggregation cuboids in S.
- D_i, provided there is $C \in S$ satisfying $t[i] = *$ for any $t \in C$.

Hence $|C_f - C_c|$ reaches its lower bound when $S = \{C_1, C_2\}$, which is equal to $2D_1 + 2D_2 - 9$, as shown in the first part of the current proof - concluding the proof of Theorem 2.

□

Outbound Authentication for Programmable Secure Coprocessors

Sean W. Smith[*]

Department of Computer Science, Dartmouth College
Hanover, New Hampshire USA
`http://www.cs.dartmouth.edu/~sws/`

Abstract. A programmable secure coprocessor platform can help solve many security problems in distributed computing. However, these solutions usually require that coprocessor applications be able to participate as full-fledged parties in distributed cryptographic protocols. Thus, to fully enable these solutions, a generic platform must not only provide programmability, maintenance, and configuration in the hostile field—it must also provide outbound authentication for the entities that result. A particular application on a particular untampered device must be able to prove who it is to a party on the other side of the Internet.

This paper offers our experiences in solving this problem for a high-end secure coprocessor product. This work required synthesis of a number of techniques, so that parties with different and dynamic views of trust can draw consistent and complete conclusions about remote coprocessor applications. These issues may be relevant to the industry's growing interest in rights management for general desktop machines.

1 Introduction

How does one secure computation that takes place remotely—particularly when someone with direct access to that remote machine may benefit from compromising that computation? This issue lies at the heart of many current e-commerce, rights management, and PKI issues.

To address this problem, research (e.g., [17, 24, 25]) has long explored the potential of *secure coprocessors*: high-assurance hardware devices that can be trusted to carry out computation unmolested by an adversary with direct physical access. For example, such an adversary could subvert rights management on a complex dataset by receiving the dataset and then not following the policy; secure coprocessors enable solutions by receiving the dataset encapsulated with the policy, and only revealing data items in accordance with the policy. [13] For

[*] A preliminary version of this paper appeared as Dartmouth College Technical Report TR2001-401. This paper reports work the author did while a research staff member at IBM Watson. Subsequent work was supported in part by the Mellon Foundation, AT&T/Internet2, and the U.S. Department of Justice (contract 2000-DT-CX-K001). The views and conclusions do not necessarily reflect the sponsors.

another example, an adversary could subvert *decentralized e-cash* simply by increasing a register. However, secure coprocessors enable solutions: the register lives inside a trusted box, which modifies the value only as part of a transaction with another trusted box. [24] Many other applications—including private information retrieval [3, 19], e-commerce co-servers [14], and mobile agents [26]—can also benefit from the high-assurance neutral environment that secure coprocessors could provide.

As the literature (e.g.,[5, 8]) discusses, achieving this potential requires several factors, including establishing and maintaining physical security, enabling the device to authenticate code-loads and other commands that come from the outside world, and building applications whose design does not negate the security advantages of the underlying platform.

However, using secure coprocessors to secure distributed computation *also* requires *outbound authentication (OA)*: the ability of coprocessor applications be able to authenticate themselves to remote parties. (Code-downloading loses much of its effect if one cannot easily authenticate the entity that results!) Merely configuring the coprocessor platform as the appropriate entity—a rights box, a wallet, an auction marketplace—does not suffice in general. A signed statement *about* the configuration also does not suffice. For maximal effectiveness, the platform should enable the *entity itself* to have authenticated key pairs and engage in protocols with any party on the Internet: so that only that particular trusted auction marketplace, following the trusted rules, is able to receive the encrypted strategy from a remote client; so that only that particular trusted rights box, following the trusted rules, is able to receive the object and the rights policy it should enforce.

The Research Project. The software architecture for a programmable secure coprocessor platform must address the complexities of shipping, upgrades, maintenance, and hostile code, for a generic platform that can be configured and maintained in the hostile field. [18] Our team spent several years working on developing a such a device; other reports [7, 8, 20] present our experiences in bringing such a device into existence as a COTS product, the IBM 4758.

Although our initial security architecture [20] sketched a design for outbound authentication, we did not fully implement it—nor fully grasp the nature of the problem—until the Model 2 device released in 2000. As is common in product development, we had to concurrently undertake tasks one might prefer to tackle sequentially: identify fundamental problems; reason about solutions; design, code and test; and ensure that it satisfied legacy application concerns.

The Basic Problem. A relying party needs to conclude that a particular key pair really belongs to a particular software entity within a particular untampered coprocessor. Design and production constraints led to a non-trivial set of software entities in a coprocessor at any one time, and in any one coprocessor over time. Relying parties trust some of these entities and not others; furthermore, we needed to accommodate a *multiplicity* of trust sets (different parties have different views), as well as the *dynamic* nature of any one party's trust set over time.

This background sets the stage for the basic problem: how should the device generate, certify, change, store, and delete private keys, so that relying parties can draw those conclusions consistent with their trust set, and only those conclusions?

This Paper. This paper is a post-facto report expanding on this research and development experience, which may have relevance both to other secure coprocessor technology (e.g., [23]) as well as to the growing industry interest in remotely authenticating what's executing on a desktop (e.g., [9, 10, 11, 21]).

Section 2 discusses the evolution of the problem in the context of the underlying technology. Section 3 presents the theoretical foundations. Section 4 presents the design. Section 5 suggests some directions for future study.

2 Evolution of Problem

2.1 Underlying Technology

We start with a brief overview of the software structure of the 4758. The device is *tamper-responding*: with high assurance, on-board circuits detect tamper attempts and destroy the contents of volatile RAM and non-volatile battery-backed RAM (BBRAM) before an adversary can see them. The device also is a general purpose computing device; internal software is divided into *layers*, with layer boundaries corresponding to divisions in function, storage region, and external control. The current family of devices has four layers: Layer 0 in ROM, and Layer 1 through Layer 3 in rewritable FLASH.

The layer sequence also corresponds to the sequence of execution phases after device boot: initially Layer 0 runs, then invokes Layer 1, and so on. (Because our device is an enclosed controlled system, we can avoid the difficulties of secure boot that arise in exposed desktop systems; we know execution starts in Layer 0 ROM in a known state, and higher-level firmware is changed only when the device itself permits it.) In the current family, Layer 2 is intended to be an internal operating system, leading to the constraint that it must execute at maximum CPU privilege; it invokes the single application (Layer 3) but continues to run, depending on its use of the CPU privilege levels to protect itself.

We intended the device to be a generic platform for secure coprocessor applications. The research team insisted on the goal that third parties (different from IBM, and from each other) be able to develop and install code for the OS layer and the application layer. Business forces pressured us to have only one shippable version of the device, and to ensure that an untampered device with no hardware damage can always be revived. We converged on a design where Layer 1 contains the security configuration software which establishes owners and public keys for the higher layers, and validates code installation and update commands for those layers from those owners. This design decision stemmed from our vision that application code and OS code may come from different entities who may not necessarily trust each other's updates; centralization of loading made it easier to enforce the appropriate security policies.

Layer 1 is updatable, in case we want to upgrade algorithms, fix bugs, or change the public key of the party authorized to update the layer. However, we mirror this layer—updates are made to the inactive copy, which is then atomically made active for the next boot—so that failures during update will not leave us with a non-functioning code-loading layer.

Authentication Approach. Another business constraint we had was that the only guaranteed contact we would have with a card was at manufacture time. In particular, we could assume no audits, nor database of card-specific data (secret or otherwise), nor provide any on-line services to cards once they left. This constraint naturally suggested the use of *public-key cryptography* for authentication, both inbound and outbound.

Because of the last-touch-at-manufacturing constraint, we (the manufacturer) can last do something cryptographic at the factory.

2.2 User and Developer Scenarios

Discussions about potential relying parties led to additional requirements.

Developers were not necessarily going to trust each other. For example, although an application developer must trust the contents of the lower layers when his application is actually installed, he should be free to require that his secrets be destroyed should a lower layer be updated in a way he did not trust. As a consequence, we allowed each code-load to include a policy specifying the conditions under which that layer's secrets should be preserved across changes to lower layers. Any other scenario destroys secrets.

However, even full-preservation developers reserved the right to, post-facto, decide that certain versions of code—even their own—were untrusted, and be able to verify whether an untrusted version had been installed during the lifetime of their secrets.

In theory, the OS layer should resist penetration by a malicious application; in practice, operating systems have a bad history here, so we only allow one application above it (and intend the OS layer solely to assist the application developer). Furthermore, we need to allow that some relying parties will believe that the OS in general (or specific version) may indeed be penetrable by malicious applications.

Small developers may be unable to assure the public of the integrity and correctness of their applications (e.g., through code inspection, formal modeling, etc). Where possible, we should maximize the credibility our architecture can endow on such applications.

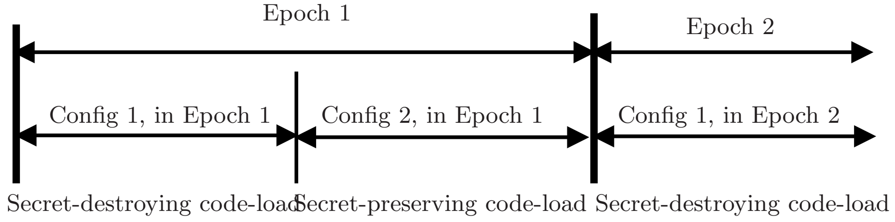

Epoch 1
Epoch 2
Config 1, in Epoch 1
Config 2, in Epoch 1
Config 1, in Epoch 2
Secret-destroying code-load
Secret-preserving code-load
Secret-destroying code-load

Fig. 1. An *epoch* starts with code-load action that clears a layer's secrets; with an epoch, each secret-preserving code-load starts a new *configuration*

2.3 On-Card Entities

One of the first things we need to deal with is the notion of what an on-card entity *is*. Let's start with a simple case: suppose the coprocessor had exactly one place to hold software and that it zeroized all state with each code-load. In this scenario, the notion of entity is pretty clear: a particular code-load C_1 executing inside an untampered device D_1. The same code C_1 inside another device D_2 would constitute a different entity; as would a re-installation of C_1 inside D_1.

However, this simple case raises a challenges. If a reload replaces C_1 with C_2, and reloads clear all tamper-protected memory, how does the resulting entity—C_2 on D_1—authenticate itself to a party on the other side of the net? The card itself would have no secrets left, since the only data storage hidden from physical attack was cleared. Consequently, any authentication secrets would have to come with C_2, and we would start down a path of shared secrets and personalized code-loads. Should an application entity "include" the OS underneath it? Should it include the configuration control layers that ran earlier in this boot sequence, but are no longer around?

Since we built the 4758 to support real applications, we gravitated toward a practical definition: an entity is an installation of the application software in a trusted place, identified by all underlying software and hardware.

Secret Retention. As noted, developers demanded that we sometimes permit secret retention across reload. With a secret-preserving load; the entity may stay the same, but the code may change. The conflicting concepts that developers had about what exactly happens to their on-card entity when code update occurs lead us to think more closely about entity lifetimes. We introduce some language to formalize that. Figure 1 sketches these concepts.

Definition 1 (Configuration, Epoch). *A Layer N* configuration *is the maximal period in which that Layer is runnable, with an unchanging software environment. A Layer N* epoch *is the maximal period in which the Layer can run and accumulate state. If E is an on-card entity in Layer N,*

– *E is an* epoch-entity *if its lifetime extends for a Layer n epoch.*

– *E is a* configuration-entity *if its lifetime extends for a Layer n configuration.*

An Layer n epoch-entity consists of a sequence of Layer n configuration-entities. This sequence may be unbounded—since any particular epoch might persist indefinitely, across arbitrarily many configuration changes.

2.4 Authentication Scenarios

This design left us with on-card software entities made up of several components with differing owners, lifetimes, and state. A natural way to do outbound authentication to give the card a certified key pair, whose private key lives in tamper-protected memory. However, the complexity of the entity structure creates numerous problems.

Application Code. Suppose entity C is the code C_1 residing in the application Layer 3 in a particular device. C may change: two possible changes include a simple code update taking the current code C_1 to C_2, or a complete re-install of a different application from a different owner, taking C_1 to C_3.

If a relying party P trusts C_1, C_2, and C_3 to be free of flaws, vulnerabilities, and malice, then the natural approach might work. However, if P distrusts some of this code, then problems arise.

– If P does not trust C_1, then how can P distinguish between an entity with the C_2 patch, and an entity with a corrupt C_1 pretending to have the C_2 patch?
– If P does does not trust C_2, then then how can P distinguish between the an entity with the honest C_1, and an entity with the corrupt C_2 pretending to be the honest C_1? (The mere existence of a signed update command compromises all the cards.)
– If P does not trust C_3, then how can P distinguish between the honest C_1 and a malicious C_3 that pretends to be C_1?

Code-Loading Code. Even more serious problems arise if a corrupted version of the configuration software in Layer 1 exists. If an evil version existed that allowed arbitrary behavior, then (without further countermeasures) a party P cannot distinguish between *any* on-card entity E_1, and an E_2 consisting of a rogue Layer 1 carrying out some elaborate impersonation.

OS Code. Problems can also arise because the OS code changes. Debugging an application requires an operating system with debug hooks; in final development stages, a reasonable scenario is to be able to "update" back-and-forth between a version of the OS with debug hooks and a version without.

With no additional countermeasures, a party P cannot distinguish between the application running securely with the real OS, the application with debug hooks underneath it, and the application with the real OS but with a policy that permits hot-update to the debug version. The private key would be the same in all cases.

Internal Certification. The above scenarios suggest that perhaps a single key pair (for all entities in a card for the lifetime of the card) may not suffice. However, extending to schemes where one on-card entity generates and certifies key pairs for other on-card entities also creates challenges.

For example, suppose Layer 1 generates and certifies key pairs for the Layer 2 entity. If a reload replaces corrupt OS B_1 with an honest B_2, then party P should be able to distinguish between the certified key pair for B_2 and that for B_1. However, without further countermeasures, if supervisor-level code can see all data on the card, then B_1 can forge messages from B_2—since it could have seen the Layer 1 private key.

A similar penetrated-barrier issue arises if we expect an OS in Layer 2 to maintain a private key separate from an application Layer 3, or if we entertained alternative schemes where mutually suspicious applications executed concurrently. If a hostile application might in theory penetrate the OS protections, then an external party cannot distinguish between messages from the OS, messages from the honest application, and messages from rogue applications.

This line of thinking led us to the more general observation that, if the *certifier* outlives the *certified*, then the integrity of what the certified does with their key pair depends on the *future* behavior of the certifier. In the case of the coprocessor, this observation has subtle and dangerous implications—for example, one of the reasons we centralized configuration control in Layer 1 was to enable the application developer to distrust the OS developer and request that the application (and its secrets) be destroyed, if the underlying OS undergoes an update the application developer does not trust. What if the untrusted OS has access to a private key used in certifying the original application?

We revisit these issues in Section 4.3.

3 Theory

The construction of the card suggests that we use certified key pairs for outbound authentication. However, as we just sketched, the straightforward approach of just sending the card out with a certified key pair permits trouble.

In this section, we try to formalize the principles that emerged while considering this problem.

A card leaves the factory and undergoes some sequence of code loads and other configuration changes. A relying party interacts with an entity allegedly running inside this card. The card's OA scheme enables this application to wield a private key and to offer a collection of certificates purporting to authenticate its keyholder.

It would be simplest if the party could use a straightforward validation algorithm on this collection. As Maurer [15, 16] formalized, a relying party's validation algorithm needs to consider which entities that party trusts. Our experience showed that parties have a wide variety of trust views that change dynamically. Furthermore, we saw the existence of two spaces: the conclusions that a party *will* draw, given an entity's collection of certificates and the party's trust view; and the conclusions that a party *should* draw, given the history of those keyholders and the party's trust view.

We needed to design a scheme that permits these sets of conclusions to match, for parties with a wide variety of trust views.

3.1 What the Entity Says

Relying party P wants to authenticate interaction with a particular entity E. Many scenarios could exist here; for simplicity, our analysis reduces these to the scenario of E needing to prove to P that $\mathsf{own}(E, K)$: that E has exclusive use of the private element of key pair K.

We need to be able to talk about what happens to a particular coprocessor.

Definition 2 (History, Run, $\prec$). *Let a* history *be a finite sequence of computation for a particular device. Let a* run *be some unbounded sequence of computation for a particular device. We write $H \prec R$ when history H is a prefix of run R.*

In the context of OA for coprocessors that cannot be opened or otherwise examined, and that disappear once they leave the factory, it seemed reasonable to impose the restriction that on-card entities carry their certificates. For simplicity, we also imposed the restriction that they present the same fixed set no matter who asks.

Definition 3. *When entity E wishes to prove it owns K after history H, let $\mathsf{Chain}(E, K, H)$ denote the set of certificates that it presents.*

3.2 Validation

Will a relying party P believe that E owns K?

First, we need some notion of trust. A party P usually has some ideas of which on-card applications it might trust to behave "correctly" regarding keys and signed statements, and of which ones it is unsure.

Definition 4. *For a party P, let $\mathsf{TrustSet}(P)$ denote the set of entities whose statements about certificates P trusts. Let root be the factory CA: the trust root for the card. A* legitimate *trust set is one that contains root.*

In the context of OA for coprocessors, it was reasonable to impose the restriction that the external party decides validity based on an entity's chain and the party's own list of trusted entities. (The commercial restriction that we can never count on accessing cards after they leave made revocation infeasible.) We formalize that:

Definition 5 (Trust-set scheme). *A* trust-set *certification scheme is one where the relying party's $\mathsf{Validate}$ algorithm is deterministic on the variables $\mathsf{Chain}(E, K, H)$ and $\mathsf{TrustSet}(P)$.*

We thus needed to design a trust-set certification scheme that accommodates *any legitimate trust set*, since discussion with developers (and experiences doing security consulting) suggested that relying parties would have a wide divergence of opinions about which versions of which software they trust.

3.3 Dependency

The problem scenarios in Section 2.4 arose because one entity E_2 had an unexpected avenue to use the private key that belonged to another entity E_1. We need language to express these situations, where the integrity of E_1's key actions *depends* on the correct behavior of E_2.

Definition 6 (Dependency Function). *Let $\mathcal{E}$ be the set of entities. A* dependency function *is a function $\mathcal{D} : \mathcal{E} \longrightarrow 2^{\mathcal{E}}$ such that, for all E_1, E_2:*

- $E_1 \in \mathcal{D}(E_1)$
- *if* $E_2 \in \mathcal{D}(E_1)$ *then* $\mathcal{D}(E_2) \subset \mathcal{D}(E_1)$

When a dependency function depends on the run R, we write $\mathcal{D}_R$.

What dependency function shall we use for analysis? In our specialized hardware, code runs in a single-sandbox controlled environment which (if the physical security works as intended) is free from outside observation or interference. Hence, in our analysis, dependence follows read and write:

Definition 7. *For entities E_1 and E_2 in run R, we write $E_2 \xrightarrow{\text{data}}_R E_1$ when E_1 has read/write access to the secrets of E_2 ($E_2 \xrightarrow{\text{data}}_R E_2$ trivially) and $E_2 \xrightarrow{\text{code}}_R E_1$ when E_1 has write access to the code of E_2. Let $\longrightarrow_R$ be the transitive closure of the union of these two relations. For an entity E in a run R, define* $\mathrm{Dep}_R(E)$ *to be $\{F : E \longrightarrow_R F\}$.*

In terms of the coprocessor, if C_1 follows B_1 in the post-boot sequence, then we have $C_1 \xrightarrow{data}_R B_1$ (since B_1 could have manipulated data before passing control). If C_2 is a secret-preserving replacement of C_1, then $C_1 \xrightarrow{data}_R C_2$ (because C_2 still can touch the secrets C_1 left). If A can reburn the FLASH segment where B lives, then $B \xrightarrow{code}_R A$ (because A can insert malicious code into B, that would have access to B's private keys).

3.4 Consistency and Completeness

Should the relying party draw the conclusions it actually will? In our analysis, security dependence depends on the run; entity and trust do not. This leads to a potential conundrum. Suppose, in run R, $C \longrightarrow_R B$ and $C \in \mathsf{TrustSet}(P)$, but $B \notin \mathsf{TrustSet}(P)$. Then a relying party P cannot reasonably accept any signed statement from C, because B may have forged it.

To capture this notion, we define *consistency* for OA. The intention of consistency is that if the party concludes that an message came from an entity, then it really did come from that entity—modulo the relying party's trust view. That is, in any $H \prec R$ where P concludes $\mathsf{own}(E, K)$ from $\mathsf{Chain}(E, K, H)$, if the entities in $\mathsf{TrustSet}(P)$ behave themselves, then E really does own K. We formalize this notion:

Definition 8. *An OA scheme is* consistent *for a dependency function $\mathcal{D}$ when, for any entity E, a relying party P with any legitimate trust set, and history and run $H \prec R$:*

$$\mathit{Validate}(P, \mathit{Chain}(E, K, H)) \Longrightarrow \mathcal{D}_R(E) \subseteq \mathit{TrustSet}(P)$$

One might also ask if the relying party *will* draw the conclusions it actually *should.* We consider this question with the term *completeness*. If in any run where E produces $\mathsf{Chain}(E, K, H)$ and $\mathcal{D}_R(E)$ is trusted by P—so in P's view, no one who had a chance to subvert E would have—then P should conclude that E owns K.

Definition 9. *An OA scheme is* complete *for a dependency function $\mathcal{D}$ when, for any entity E claiming key K, relying party P with any legitimate trust set, and history and run $H \prec R$:*

$$\mathcal{D}_R(E) \subseteq \mathit{TrustSet}(P) \Longrightarrow \mathit{Validate}(P, \mathit{Chain}(E, K, H))$$

These definitions equip us to formalize a fundamental observation:

Theorem 1. *Suppose a trust-set OA scheme is both consistent and complete for a given dependency function $\mathcal{D}$. Suppose entity E claims K in histories $H_1 \prec R_1$ and $H_2 \prec R_2$. Then:*

$$\mathcal{D}_{R_1}(E) \neq \mathcal{D}_{R_2}(E) \Longrightarrow \mathit{Chain}(E, K, H_1) \neq \mathit{Chain}(E, K, H_2)$$

Proof. Suppose $\mathcal{D}_{R_1}(E) \neq \mathcal{D}_{R_2}(E)$ but $\mathsf{Chain}(E, K, H_1) = \mathsf{Chain}(E, K, H_2)$. We cannot have both $\mathcal{D}_{R_1}(E) \subseteq \mathcal{D}_{R_2}(E)$ and $\mathcal{D}_{R_2}(E) \subseteq \mathcal{D}_{R_1}(E)$, so, without loss of generality, let us assume $\mathcal{D}_{R_2}(E) \not\subseteq \mathcal{D}_{R_1}(E)$. There thus exists a set S with $\mathcal{D}_{R_1}(E) \subseteq S$ but $\mathcal{D}_{R_2}(E) \not\subseteq S$.

Since the scheme is consistent and complete, it must work for any legitimate trust set, including S. Let party P have $S = \mathsf{TrustSet}(P)$. Since this is a trust-set certification scheme and E produces the same chains in both histories, party P must either validate these chains in both scenarios, or reject them in both scenarios. If party P accepts in run R_2, then the scheme cannot be consistent for $\mathcal{D}$, since E depends on an entity that P did not trust. But if party P rejects in run R_1, then the scheme cannot be complete for $\mathcal{D}$, since party P trusts all entities on which E depends.

3.5 Design Implications

We consider the implications of Theorem 1 for specific ways of constructing chains and drawing conclusions, for specific notions of dependency. For example, we can express the standard approach—P makes its conclusion by recursively verifying signatures and applying a basic inference rule—in a Maurer-style calculus [15]. Suppose C is a set of *certificates*: statements of the form

K_1 says $\mathsf{own}(E_2, K_2)$. Suppose S be a set of entities *trusted to speak the truth about certificate ownership*: $\{E_1 : \mathsf{trust}(E_1, \mathsf{own}(E_2, K_2))\}$. A relying party may start by believing $C \cup \{\mathsf{own}(\mathsf{root}, K_{\mathsf{root}})\}$.

To describe what a party will conclude, we can define $\mathsf{View}_{\mathsf{will}}(C, S)$ to be the set of statements derivable from this set by applying the rule

$$\mathsf{own}(E_1, K_1), \quad E_1 \in S, \quad K_1 \text{ } \mathsf{says} \text{ } \mathsf{own}(E_2, K_2) \vdash \mathsf{own}(E_2, K_2)$$

The $\mathsf{Validate}$ algorithm for party P then reduces to the decision of whether $\mathsf{own}(E, K)$ is in this set.

We can also express what a party should conclude about an entity, in terms of the chain the entity presents, and the views that the party has regarding trust and dependency. If $\mathcal{D}$ is a dependency function, we can define $\mathsf{View}_{\mathsf{should}}(C, S, \mathcal{D})$ to be the set of statements derivable by applying the alternate rule:

$$\mathsf{own}(E_1, K_1), \quad \mathcal{D}(E_1) \subseteq S, \quad K_1 \text{ } \mathsf{says} \text{ } \mathsf{own}(E_2, K_2) \vdash \mathsf{own}(E_2, K_2)$$

In terms of this calculus, we obtain consistency be ensuring that for any chain and legitimate trust set, and $H \prec R$, the set $\mathsf{View}_{\mathsf{will}}(\mathsf{Chain}(E, K, H), S)$ is contained in the set $\mathsf{View}_{\mathsf{should}}(\mathsf{Chain}(E, K, H), S, \mathcal{D}_R)$. The relying party should only use a certificate to reach a conclusion when the entire dependency set of the signer is in $\mathsf{TrustSet}(P)$.

4 Design

For simplicity of verification, we would like $\mathsf{Chain}(E, K, H)$ to be a literal chain: a linear sequence of certificates going back to root. To ensure consistency and completeness, we need to make sure that, at each step in the chain, the partial set of certifiers equals the dependency set of that node (for the dependency function we see relying parties using). To achieve this goal, the elements we can manipulate include generation of this chain, as well as how dependency is established in the device.

4.1 Layer Separation

Because of the post-boot execution sequence, code that executes earlier can subvert code that executes later.[1] If B, C are Layer i, Layer $i+1$ respectively, then $C \longrightarrow_R B$ unavoidably.

However, the other direction should be avoidable, and we used hardware to avoid it. To provide high-assurance separation, we developed *ratchet locks:* an independent microcontroller tracks a counter, reset to zero at boot time. The microcontroller will advance the ratchet at the main coprocessor CPU's request, but never roll it back. Before B invokes the next layer, it requests an advance.

[1] With only one chance to get the hardware right, we did not feel comfortable with attempting to restore the system to a more trusted state, short of reboot.

To ensure $B \overset{data}{\not\longrightarrow}_R C$, we reserved a portion of BBRAM for B, and used the ratchet hardware to enforce access control. To ensure $B \overset{code}{\not\longrightarrow}_R C$, we write-protect the FLASH region where B is stored. The ratchet hardware restricts write privileges only to the designated prefix of this execution sequence.

To keep configuration entities from needlessly depending on the epoch entities, in our Model 2 device, we subdivided the higher BBRAM to get four regions, one each for epoch and configuration lifetimes, for Layer 2 and Layer 3. The initial clean-up code in Layer 1 (already in the dependency set) zeroizes the appropriate regions on the appropriate transition. (For transitions to a new Layer 1, the clean-up is enforced by the *old* Layer 1 and the permanent Layer 0—to avoid incurring dependency on the new code.)

4.2 The Code-Loading Code

As discussed elsewhere, we felt that centralizing code-loading and policy decisions in one place enabled cleaner solutions to the trust issues arising when different parties control different layers of code. But this centralization creates some issues for OA. Suppose the code-loading Layer 1 entity A_1 is reloaded with A_2. Business constraints dictated that A_1 do the reloading, because the ROM code had no public-key support. It's unavoidable that $A_2 \xrightarrow{code}_R A_1$ (because A_1 could have cheated, and not installed the correct code). However, to avoid $A_1 \xrightarrow{data}_R A_2$, we take these steps as an atomic part of the reload: A_1 generates a key pair for its successor A_2; A_1 uses its current key pair to sign a *transition certificate* attesting to this change of versions and key pairs; and A_1 destroys its current private key.

This technique—which we implemented and shipped with the Model 1 devices in 1997—differs from the standard concept of *forward security* [1] in that we change keys with each new version of software, and ensure that *the name of the new version is spoken by the old version.* As a consequence, a single malicious version cannot hide its presence in the trust chain; for a coalition of malicious versions, the trust chain will name at least one. (Section 5 considers forward-secure signatures further.)

4.3 The OA Manager

Since we do not know a priori what device applications will be doing, we felt that application key pairs needed to be created and used at the application's discretion. Within our software architecture, Layer 2 should do this work—since it's easier to provide these services at run-time instead of reboot, and the Layer 1 protected memory is locked away before Layer 2 and Layer 3 run.

This *OA Manager* component in Layer 2 will wield a key pair generated and certified by Layer 1, and will then generate and certify key pairs at the request of Layer 3. These certificates indicate that said key pair belongs to an application, and also include a field chosen by the application. (A fuller treatment of our

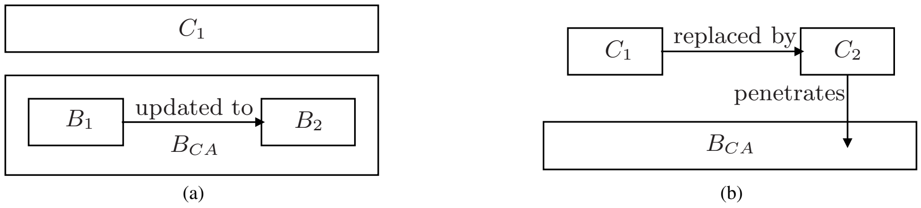


Fig. 2. Having the OS certify key pairs for the application creates interesting lifetime issues. In (a), if the OS is updated while preserving its key pair, then the application depends on both versions; in (b), if the OS outlives the application but is potentially penetrable, then the application may depend on future applications

trust calculus would thus distinguish between owning and trusting a key pair for certification purposes, and owning and trusting a key pair for the application-specified purpose—the last link.)

To keep the chain linear, we decided to have Layer 1 generate and destroy the OA Manager key pair (e.g., instead of adding a second horizontal path between successive versions of the OA Manager key pairs). The question then arises of when the OA Manager key pair should be created and destroyed.

We discuss some false starts. If the OA Manager outlived the Layer 2 configuration, then our certification scheme *cannot be both consistent and complete.* For a counterexample (see Figure 2, (a)) suppose that application C_1 is a configuration-entity on top of OS B_1; that OS B_1 changes code to OS B_2 but both are part of the same entity B_{CA}; and that party P trusts C_1, B_1 but not B_2. For the scheme to be complete, P should accept certificate chains from C_1—but that means accepting a chain from B_{CA}, and $B_{CA} \longrightarrow_R B_2 \notin \mathsf{TrustSet}(P)$.

This counterexample fails because the application entity has a CA whose dependency set is larger than the application's. Limiting the CA to the current Layer 2 configuration eliminates this issue, but still fails to address penetration risk. Parties who come to believe that a particular OS can be penetrated by an application can end up with the current B_{CA} depending on *future application loads.* (See Figure 2, (b).)

Our final design avoided these problems by having the Layer 2 OA Manager live *exactly* as long as the Layer 3 configuration. Using the protected BBRAM regions, we ensure that upon any change to the Layer 3 configuration, Layer 1 destroys the old OA Manager private key, generates a new key pair, and certifies it to belong to the new OA Manager for the new Layer 3 configuration. This approach ensures that the trust chain names the dependency set for Layer 3 configurations—even if dependency is extended to include penetration of the OS/application barrier.

Since we need to accommodate the notion of both *epoch* and *configuration* lifetimes (as well as to allow parties who choose the former to change their minds), we identify entities both by their current epoch, as well as the current configuration within that epoch. When it requests a key pair, the Layer 3 application can specify which lifetime it desires; the certificate includes this information. Private keys for a configuration lifetime are kept in the Layer 3 configuration region in BBRAM; private keys for an epoch lifetime are kept in the epoch region.

Note that a Layer 3 epoch certificate (say, for epoch E) still names the configuration (say, C_1) in which it began existence. If, in some later configuration C_k within that same epoch, the relying party decides that it wants to examine the individual configurations to determine the whether an untrusted version was present, it can do that by examining the trust chain for C_k and the sequence of OA Manager certificates from C_1 to C_k. An untrusted Layer 1 will be revealed in the Layer 1 part of the chain; otherwise, the sequence of OA Manager certificates will have correct information, revealing the presence of any untrusted Layer 2 or Layer 3 version.

4.4 Summary

As noted earlier, the trust chain for the current Layer 1 version starts with the certificate the factory root signed for the first version of Layer 1 in the card, followed by the sequence of transition certificates for each subsequent version of Layer 1 installed. The trust chain for the OA Manager appends the OA Manager certificate, signed by the version of Layer 1 active when that Layer 3 configuration began, and providing full identification for the current Layer 2 and Layer 3 configurations and epochs. The trust chain for a Layer 3 key pair appends the certificate from the OA Manager who created it.

Our design thus constitutes a trust-set scheme that is consistent and complete for the dependency function we felt was appropriate, for any legitimate trust set.

4.5 Implementation

Full support for OA shipped with all Model 2 family devices and the CP/Q++ embedded operating system. (The announced Linux port [12] still has the Layer 1 OA hooks; extending Linux to handle that is an area of future work.)

Implementation required some additional design decisions. To accommodate small developers (Section 2.2), we decided to have the OA Manager retain all Layer 3 private keys and wield them on the application's behalf; consequently, a party who trusts the penetration-resistance of a particular Layer 2 can thus trust that the key was at least used within that application on an untampered device. Another design decision resulted from the insistence of an experienced application architect that users and developers will not pay attention to details of certificate paths; to mitigate this risk, we do not provide a "verify this chain" service—applications must explicitly walk the chain—and we gave different families of cards different factory roots.

A few aspects of the implementation proved surprising. One aspect was the fact that the design required two APIs: one between Layer 1 and Layer 2, and another between Layer 2 and the application. Another aspect was finding places to store keys. We extended the limited area in BBRAM by storing a MAC key and a TDES encryption key in each protected region, and storing the ciphertext for new material wherever we could: during a code-change, that region's FLASH segment; during application run-time, in the Layer 2-provided PPD data storage service. Another interesting aspect was the multiplicity of keys and identities added when extending the Layer 1 transition engine to perform the appropriate generations and certifications. For example, if Layer 1 decides to accept a new Layer 1 load, we now also need to generate a new OA Manager key pair, and certify it *with the new Layer* 1 *key pair* as additional elements of this atomic change. Our code thus needed two passes before commitment: one to determine everyone's names should the change succeed, and another to then use these names in the construction of new certificates.

As has been noted elsewhere [8], we regret the design decisions to use our own certificate format, and the fact that the device has no form of secure time (e.g., Layer 3 can always change the clock). Naming the configuration and epoch entities was challenging, particularly since the initial architecture was designed in terms of parameters such as code version and owner, and a precise notion of "entity" only emerged later.

5 Conclusions

One might characterize the entire OA architecture process "tracing each dependency, and securing it." Our experience here, like other aspects of this work, balanced the goals of enabling secure coprocessing applications while also living within product deadlines. OA enables Alice to design and release an application; Bob to download it into his coprocessor; and Charlie to then authenticate remotely that he's working with this application in an untampered device.

Outbound authentication allows third-party developers to finally deploy coprocessor applications, such as Web servers [14] and rights management boxes [13], that can by authenticated by anyone in the Internet, and participate in PKI-based protocols.

We quickly enumerate some avenues for future research and reflection.

Alternative Software Structure Our OA design follows the 4758 architecture's sequence of increasingly less-trusted entities after boot. Some current research explores architectures that dispense with this limiting assumption, and also dispensing with the 4758 assumptions of one central loader/policy engine, and of a Layer 2 that exists only to serve a one-application Layer 3. It would be interesting to explore OA in these realms.

Similarly, the analysis and design presented in this paper assumes that an authority makes a statement about an entity at the time a key pair is created. Long-lived entities with the potential for run-time corruption suggest ongoing integrity-checking techniques. It would be interesting to examine OA in light of such techniques.

Alternate Platforms Since our work, the *Trusted Computing Platform Alliance* [21] has published ideas on how remote parties might gain assurance about the software configuration of remote desktop machines; Microsoft is considering similar ideas [9, 10, 11]. It would be interesting to explore the interaction of our OA work with this increasingly timely topic, as well as the longer history of work in securely booting desktops [2, 6, 22].

Alternate Cryptography We developed our transition certificate scheme for Layer 1 to ensure that not all corrupt entities could hide their presence in a chain. Entities aside, this scheme is essentially a basic forward-secure signature scheme (e.g., [4], Sec. 3.3). It would be interesting how the broader space of forward-secure signature schemes might be used in these settings.

Alternate Dependency Our dependency function—entity E_1 can subvert E_2 when it can read or write secrets, or write code, at any time—emerged for the special case of our device. A more careful incorporation of time would be interesting, as would an examination of the other avenues of manipulation in more complex settings (e.g., the opportunities for covert channels in common desktop operating systems, or if the coprocessor *cryptopages* to the host file system).

Formalizing Trust Sets One linchpin of our design was the divergence and dynamic nature of what relying parties tend to trust. (Consequently, our analysis assumed that parties may have "any legitimate trust set.") It would be interesting to measure and formally characterize the trust behavior that occurs (or should occur) with real-world relying parties and software entities.

Formalizing Penetration Recovery Much complicated reasoning arose from scenarios such as "what if, in six months, trusted software component X turns out be flawed?" Further exploration of the design and implementation of authentication schemes that explicitly handle such scenarios would be interesting.

Acknowledgments

The author gratefully acknowledges the comments and advice of the greater IBM 4758 team; particular thanks go to Mark Lindemann, who co-coded the Layer 2 OA Manager, and Jonathan Edwards, who tested the API and transformed design notes into manuals and customer training material. The author is also grateful for the comments and advice of the Internet2 PKI Labs on new research issues here. Finally, gratitude is due the anonymous referees, who have made this a stronger and clearer paper.

References

[1] R. Anderson. Invited lecture, *ACM CCS*, 1997. The definitive written record appears to be *Two Remarks on Public Key Cryptology,* http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/forwardsecure.pdf. 83

[2] W. A. Arbaugh, D. J. Farber, J. M. Smith. "A Secure and Reliable Bootstrap Architecture." *IEEE Computer Society Conference on Security and Privacy.* 1997. 87

[3] D. Asonov and J. Freytag. "Almost Optimal Private Information Retrieval." *Privacy Enhancing Technology 2002*. Springer-Verlag LNCS. 73

[4] M. Bellare and S. Miner. "A Forward-Secure Digital Signature Scheme." Extended abstract appears in *Crypto 99*, Springer-Verlag LNCS. Full version at `http://www-cse.ucsd.edu/~mihir/papers/fsig.html`. 87

[5] M. Bond, R. Anderson. "API-Level Attacks on Embedded Systems." *IEEE Computer.* 34:67-75. October 2001. 73

[6] L. Clark and L. J. Hoffmann. "BITS: A Smartcard Protected Operating System." *Communications of the ACM.* 37: 66-70. 1994. 87

[7] J. Dyer, R. Perez, S. W. Smith, M. Lindemann. "Application Support Architecture for a High-Performance, Programmable Secure Coprocessor." *22nd National Information Systems Security Conference.* October 1999. 73

[8] J. Dyer, M. Lindemann, R. Perez, R. Sailer, S. W. Smith, L. van Doorn, S. Weingart. "Building the IBM 4758 Secure Coprocessor." *IEEE Computer*, 34:57-66. October 2001. 73, 86

[9] P. England, J. DeTreville and B. Lampson. *Digital Rights Management Operating System.* United States Patent 6,330,670. December 2001. 74, 87

[10] P. England, J. DeTreville and B. Lampson. *Loading and Identifying a Digital Rights Management Operating System.* United States Patent 6,327,652. December 2001. 74, 87

[11] P. England, M. Peinado. "Authenticated Operation of Open Computing Devices." *7th Australasian Conference on Information Security and Privacy.* Springer-Verlag LNCS2384. June 2002. 74, 87

[12] "IBM Research Demonstrates Linux Running on Secure Cryptographic Coprocessor." Press release, August 28, 2001. 85

[13] A. Iliev, S. W. Smith. "Prototyping an Armored Data Vault: Rights Management for Big Brother's Computer." *Privacy Enhancing Technology 2002*. Springer-Verlag LNCS. 72, 86

[14] S. Jiang, S. W. Smith, K. Minami. "Securing Web Servers against Insider Attack." *ACSA/ACM Annual Computer Security Applications Conference.* December 2001. 73, 86

[15] R. Kohlas and U. Maurer. "Reasoning About Public-Key Certification: On Bindings Between Entities and Public Keys." *Journal on Selected Areas in Communications.* 18: 551-560. 2000. (A preliminary version appeared in *Financial Cryptography 99*, Springer-Verlag.). 78, 81

[16] U. Maurer. "Modelling a Public-Key Infrastructure." *ESORICS 1996*. Springer-Verlag LNCS. 78

[17] S. W. Smith. *Secure Coprocessing Applications and Research Issues.* Los Alamos Unclassified Release LA-UR-96-2805, Los Alamos National Laboratory. August 1996. 72

[18] S. W. Smith, E. R. Palmer, S. H. Weingart. "Using a High-Performance, Programmable Secure Coprocessor." *Proceedings, Second International Conference on Financial Cryptography.* Springer-Verlag LNCS, 1998. 73

[19] S. W. Smith, D. Safford. "Practical Server Privacy Using Secure Coprocessors." *IBM Systems Journal (special issue on End-to-End Security)* 40: 683-695. 2001. 73

[20] S. W. Smith, S. H. Weingart. "Building a High-Performance, Programmable Secure Coprocessor." *Computer Networks (Special Issue on Computer Network Security)*. 31: 831-860. April 1999. 73

[21] Trusted Computing Platform Alliance. *TCPA Design Philosophies and Concepts, Version 1.0.* January, 2001. 74, 87

[22] J. D. Tygar and B. S. Yee. "Dyad: A System for Using Physically Secure Coprocessors." *Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment.* April 1993. 87

[23] N. van Someren. "Access Control for Secure Execution." *RSA Conference 2001.* 74

[24] B. S. Yee. *Using Secure Coprocessors.* Ph.D. thesis. Computer Science Technical Report CMU-CS-94-149, Carnegie Mellon University. May 1994. 72, 73

[25] B. S. Yee and J. D. Tygar. "Secure Coprocessors in Electronic Commerce Applications." *1st USENIX Electronic Commerce Workshop.* 1996. 72

[26] B. S. Yee. *A Sanctuary For Mobile Agents.* Computer Science Technical Report CS97-537, UCSD, April 1997. (An earlier version of this paper appeared at the *DARPA Workshop on Foundations for Secure Mobile Code.*). 73

Hamming Weight Attacks on Cryptographic Hardware – Breaking Masking Defense*

Marcin Gomułkiewicz[1] and Mirosław Kutyłowski[1,2]

[1] Cryptology Centre, Poznań University
[2] Institute of Mathematics, Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. It is believed that masking is an effective countermeasure against power analysis attacks: before a certain operation involving a key is performed in a cryptographic chip, the input to this operation is combined with a random value. This has to prevent leaking information since the input to the operation is random.
We show that this belief might be wrong. We present a Hamming weight attack on an addition operation. It works with random inputs to the addition circuit, hence masking even helps in the case when we cannot control the plaintext. It can be applied to any round of the encryption. Even with moderate accuracy of measuring power consumption it determines explicitly subkey bits. The attack combines the classical power analysis (over Hamming weight) with the strategy of the saturation attack performed using a random sample.
We conclude that implementing addition in cryptographic devices must be done very carefully as it might leak secret keys used for encryption. In particular, the simple key schedule of certain algorithms (such as IDEA and Twofish) combined with the usage of addition might be a serious danger.

Keywords: cryptographic hardware, side channel cryptanalysis, Hamming weight, power analysis

1 Introduction

Symmetric encryption algorithms are often used to protect data stored in insecure locations such as hard disks, archive copies, and so on. The key used for this purpose is stored inside a cryptographic device and should not leave the device in an unencrypted form during its lifetime. The device must offer reasonable protection against retrieving the key, since security cannot be based on physical security of the device. For this reason, investigating cryptanalytic attacks against such devices has significant importance.

The threats for devices considered are not confined to such classical "mathematical" methods as differential or linear cryptanalysis, not less important are other methods taking into account not only plaintexts and ciphertexts, but also

* This research was initiated when the second author visited University of Mannheim

any sources of information due to physical nature of computation. Particularly dangerous might be information sources that are overlooked during designing of the algorithm or those that emerge at the time an algorithm is implemented.

Cryptanalysis of cryptographic algorithms based on such *side channel* information was initiated a few years ago. In the meantime it became a recognized and extremely dangerous line of attacks.

Timing attack. Execution time of encryption may leak information on the data involved. It has been first observed for asymmetric ciphers [11] – the operations such as modular exponentiation are optimized in order to reduce the computation time (which is a problem in this case). However, computation time becomes strongly dependent on data involved and in this way may leak information on the key. In the case of a symmetric ciphers timing attack may also be relevant, as shown in the example of non careful implementation of IDEA [10].

Power analysis attack. The second side channel source of information on computations inside cryptographic device is power consumption [12]. It depends very much on the operations performed and the data manipulated inside the device.

For technical reasons, we cannot measure consumption occurring at a chosen place in the circuit (although it is possible to do similar measurements for electromagnetic radiation [8]). We can measure only global consumption and only up to some extend. However, the power consumption can be sampled over the time providing information on different stages of computation separately.

Simple power analysis (SPA) takes the data on the global power consumption of the device during different moments of the computation. A more sophisticated method is **differential power analysis** (DPA) [12]. In that case, one considers the power consumption data and some other data computed, for instance, for all candidate subkeys that may occur at some point of the computation. The idea is that for the right subkey, power consumption and the data computed for the subkey should be somehow statistically related. On the other hand, for the wrong key relation between these data should be purely random. In this way, one hopes to retrieve information on the local keys despite the fact that only global information on power consumption is available.

Hamming attack. Some authors [10] indicate that for certain hardware technologies there is a strong correlation between the number of bits set to one and power consumption. Based on this phenomenon, they propose a *Hamming-weight attack* on DES-chips (and similar ciphers) with the aim to derive a secret key stored inside the device. The attack is potentially quite dangerous for block ciphers such that small portions of a key act independently on small portions of data – just like in the case of DES and its S-boxes.

It is required for the attack to determine the total Hamming weights of the output of the Feistel function of DES during the last round in a series of encryptions. This weight consists of weight of the output of an S-Box S we are attacking and the output of the remaining S-boxes. For finding the key one guesses the subkey bits which go into the XOR gate immediately before S. Then it is possible to compute the (hypothetical) outputs of S for the ciphertexts at hand. If the subkey bits were guessed correctly, then there is a statistical correlation between the weights of the computed outputs of S and the Hamming weights measured. For a wrong guess, there is no such a correlation. This is a way to determine which guess is correct and thereby to derive the key bits.

This attack cannot be applied as it is for non-Feistel ciphers. Also operations performed on larger data blocks increase the complexity of an attack significantly.

Saturation attack. Somewhat related to this paper is saturation attack, which was applied to such ciphers as Square [5], Crypton [17], Rijndael [6] and Twofish [13]. It can be used if a part, say U, of an encryption circuit performs a (key dependent) permutation on blocks of bits of a fixed length, say m. The general idea is that if we input each possible bit string of length m into U exactly once, then each bit string of length m occurs as an output exactly once. The order in which these outputs appear depends on an unknown key, but the set of values is known.

The idea of a saturation attack is that "global" behaviour of encryption algorithm of some part is key independent, even if "local" behaviour (that is, for a single input) is key dependent. This is used to "get through" U. However, a little bit different strategy can be applied: since "global" behaviour does not depend on particular input, it may depend on the key only. In this way it may leak some information on the key involved.

Masking. Some of the attacks mentioned rely on specific properties of certain operations. The methods such as timing attack, SPA, DPA and Hamming attack explore the property that certain parameters of an operation depend heavily on input/output data. Having found a specific situation we may conclude on data involved in the operation. A part of this data might be the key bits.

In order to prevent such attacks two techniques have been introduced. The first one proposes to "split" or "duplicate" all intermediate results [9, 2, 3]. However, this method has large memory or time overhead. The second much more efficient method, called *masking*, "blinds" input data to arithmetic and boolean operations [14, 4, 7]. The idea is to combine the input with a random value, then to perform an operation with the key, and finally extract the random factor. For instance, adding subkey k to an intermediate result a can be implemented by the following operations:

```
choose $r$ at random
 $z = a + r$
 $z = z + k$
 $z = z - r$
```

Note that in the above procedure subkey k is added to a random number. This prevents the attacks mentioned. Similarly, one may mask bitwise XOR and multiplication operations. It is also possible to mask together addition and XOR [7], even if algebraically these operations are remote to each other.

1.1 New Hamming Weight Attack

We present a Hamming weight attack on addition operation where one operand is a subkey. In this way we derive efficiently most significant bits of the subkey. The attack has the following features:

- complexity of the attack depends mainly on the size of the words added, for the ciphers with 16-bit operations and simple key schedule (like IDEA) the attack reveals the whole main key; when key schedule is not reversible, then the attack reduces key space significantly,
- accuracy of measuring Hamming weight need not to be high, even poor preciseness suffices,
- it is irrelevant when the addition operation is performed, the attack does not use input and output data for the analysis,
- the data entering addition with the subkey may be masked, it does not prevent the analysis; moreover, it even helps to since the data to be added to the subkey should be fairly random: so in the situation when it is not possible to encrypt a random input, masking helps to get useful side-channel information,
- the attack computes directly subkey bits.

The side-channel information used for the attack is the Hamming weight of the sequence of carry bits. Please note: apart from Hamming weight of inputs and outputs, we also consider Hamming weight of internal data. If that weight can be measured in some way (even with a quite large error), then we are done. It does not disturb, if this weight is measured together with some other data like weight of input and output of addition, of the subkey, or any of these quantities. For the sake of clarity of exposition we describe the simplest case, in which addition is done with the school method. However, it follows from our considerations that the Hamming weight of a small number of operands existing in the addition circuit may be treated as an error of measurement and thereby our attack works in the same way.

Even if Hamming weight attacks seem at the moment to be more theoretical than practical, one has to be very careful when implementing addition in cryptographic devices. For instance, resetting registers holding the carry bits to zero before addition might help to use the standard power analysis devices: in this case power consumption due to putting the correct values of carry bits is closely related to the number of bit values changed, that is to the Hamming weight of the sequence of carry bits.

On top of that, although our attack is aimed to addition, it is probably not impossible to mount similar attack on e.g. (modular) multiplication. Multiplication is of course much more complicated operation, and finding and proving the closed formula similar to the one we prove for addiction seems very hard at the moment. Nevertheless, it is another possible threat for the hardware implementation of several block ciphers.

2 Properties of Addition

We consider addition of n-bits numbers modulo 2^n with the carry bit for position n thrown away. One of the operands will be the subkey to be derived, while the other one a random n-bit number.

We analyse the Hamming weight of numbers that occur during addition. We assume that addition is performed according to the school method. The effects specific to particular architectures of addition circuits are postponed to further research.

2.1 Expected Value of Hamming Weight

We adopt the following notation convention: random variables are denoted with capital letters, and their realisations with small letters. We make an exception for K - an unknown but fixed key.

First let us consider the total number of the ones that occur when we add a fixed key K to (chosen uniformly from $\{0, 1, \ldots, 2^n - 1\}$) number a. Note that we **do not** care about the value of a: we only want it to be chosen uniformly, and we need not know either input a or output $K + a$. If internal data are masked (as protection against power analysis), then uniform distribution is guaranteed by masking. If the device does not use masking we should use randomly chosen inputs (e.g. perform a chosen-plaintext attack).

Let $|x|$ denote the Hamming weight of number x. We consider the ones that occur while a is being added to (fixed) K. These ones fall into four categories: the ones occurring in K itself, the ones occurring in a, the ones occurring in $K + a$ and in the carry bits. Let $c = c(K, a)$ denote the number of carry-bits with the value 1. So the Hamming weight corresponding to adding a and K equals:

$$w = |K| + |a| + |K + a| + c(K, a)\,.$$

In terms of random variables we have:

$$W = |K| + |A| + |K + A| + C(K)\,,$$

where $C = C(K)$ is a random variable whose distribution depends on K alone: C realises as $c = c(K, a)$ which is a function of two arguments, one (K) fixed, and the other chosen at random (from a known distribution). Let $\mathbf{E}\,[\,X\,]$ denote the expected value of X. Since a is chosen uniformly from the set of all n-bit strings, we expect its Hamming weight to be close to $\frac{n}{2}$ and $\mathbf{E}\,[\,|A|\,] = \frac{n}{2}$. Similarly, Hamming weight of $K + a$ is supposed to be close to $\frac{n}{2}$ and $\mathbf{E}\,[\,|K + A|\,] = \frac{n}{2}$. Since expected value of a sum is a sum of expected values (even for dependent variables), we have:

$$\mathbf{E}\,[\,W\,] = |K| + \tfrac{n}{2} + \tfrac{n}{2} + \mathbf{E}\,[\,C(K)\,] = |K| + \mathbf{E}\,[\,C(K)\,] + n\,.$$

$\mathbf{E}\,[\,W\,]$ depends on the unknown key K only, so we shall denote it as a function of K: $\varphi(K)$. Basically, an attack could look as follows:

1. Collect a large pool of Hamming weight data corresponding to addition to the unknown key K. Compute average value $\widehat{\varphi}(K)$.
2. For any K' compute the theoretical values of $\varphi(K')$ and compare with $\widehat{\varphi}(K)$. If $|\varphi(K') - \widehat{\varphi}(K)|$ is large, reject K'.

Of course, such an attack would be not very practical. The attack we present is much simpler. We do not have to execute the second step: we derive certain bits of K directly from $\widehat{\varphi}(K)$. Obviously, the key point is to determine the relation between $C(K)$ and the (sub)key used.

2.2 Formula for Expected Hamming Weight of Carry Bits

In this subsection we prove the following key lemma.

Lemma 1. *Let $K = (k_{n-1}, \dots, k_0)$ be a n-bit string. Let $C = C(K)$ denote the number of carry bits set to 1 that occur during computation of $K + a \bmod 2^n$ with the school method, where a is chosen uniformly at random from the set of all n-bit strings. Then*

$$\mathbf{E}\,[\,C(K)\,] = \sum_{i=0}^{n-1} k_i - 2^{1-n} \cdot K\,. \tag{1}$$

Proof. Let $A = (a_{n-1}, \dots, a_0)$. Let c_i denote the ith carry bit generated while adding K to A. Obviously, $c_0 = 0$ (there is no carry at the beginning), and c_1 depends only on k_0 and a_0 while for $i > 0$ the bit c_i depends on a_{i-1}, k_{i-1} and c_{i-1}. This dependence complicates computation of $\mathbf{E}\,[\,C(K)\,]$.

In order to count the number of carry bits set to 1 we consider $a_{n-1}, \dots, a_0$ as independent random variables with values $0, 1$ and uniform distribution. Let $\mathbf{P}\,[\,X\,]$ denote probability of an event X. For $i = 1$ we have:

$$\begin{aligned}
\mathbf{P}\,[\,c_1 = 0\,] &= \mathbf{P}\,[\,k_0 + a_0 \le 1\,] = 1 - \tfrac{1}{2} \cdot k_0\,,\\
\mathbf{P}\,[\,c_1 = 1\,] &= \mathbf{P}\,[\,k_0 + a_0 > 1\,] = \tfrac{1}{2} \cdot k_0\,.
\end{aligned}$$

For $i > 1$ we may easily derive that

$$\begin{aligned}
\mathbf{P}\,[\,c_i = 0 \mid c_{i-1} = 0\,] &= \mathbf{P}\,[\,c_{i-1} + k_{i-1} + a_{i-1} \le 1 \mid c_{i-1} = 0\,]\\
&= \mathbf{P}\,[\,k_{i-1} + a_{i-1} \le 1\,] = 1 - \tfrac{1}{2} \cdot k_{i-1}\,,\\
\mathbf{P}\,[\,c_i = 0 \mid c_{i-1} = 1\,] &= \mathbf{P}\,[\,c_{i-1} + k_{i-1} + a_{i-1} \le 1 \mid c_{i-1} = 1\,]\\
&= \mathbf{P}\,[\,k_{i-1} + a_{i-1} \le 0\,] = \tfrac{1}{2} - \tfrac{1}{2} \cdot k_{i-1}\,,\\
\mathbf{P}\,[\,c_i = 1 \mid c_{i-1} = 0\,] &= \mathbf{P}\,[\,c_{i-1} + k_{i-1} + a_{i-1} > 1 \mid c_{i-1} = 0\,]\\
&= \mathbf{P}\,[\,k_{i-1} + a_{i-1} > 1\,] = \tfrac{1}{2} \cdot k_{i-1}\,,\\
\mathbf{P}\,[\,c_i = 1 \mid c_{i-1} = 1\,] &= \mathbf{P}\,[\,c_{i-1} + k_{i-1} + a_{i-1} > 1 \mid c_{i-1} = 1\,]\\
&= \mathbf{P}\,[\,k_{i-1} + a_{i-1} > 0\,] = \tfrac{1}{2} + \tfrac{1}{2} \cdot k_{i-1}\,.
\end{aligned}$$

One may treat the random variables $c_1, c_2, \ldots$ as a non-homogeneous Markov chain where transition probabilities depend on the values of $k_0, k_1, \ldots, k_{n-1}$. For each i we consider vector $P_i = [1 - p_i, p_i]$ where p_i stands for $\mathbf{P}[\,c_i\,] = 1$. Then

$$P_1 = [1 - \tfrac{1}{2} \cdot k_0\ ,\ \tfrac{1}{2} \cdot k_0]\ ,$$

and for $i > 1$

$$P_i = P_{i-1} \cdot \begin{bmatrix} 1 - \frac{1}{2} \cdot k_{i-1} & \frac{1}{2} \cdot k_{i-1} \\ \frac{1}{2} - \frac{1}{2} \cdot k_{i-1} & \frac{1}{2} + \frac{1}{2} \cdot k_{i-1} \end{bmatrix}\ .$$

We get

$$\begin{aligned} 1 - p_i &= (1 - p_{i-1}) \cdot (1 - \tfrac{1}{2} \cdot k_{i-1}) + p_{i-1} \cdot (\tfrac{1}{2} - \tfrac{1}{2} \cdot k_{i-1}) \\ &= 1 - \tfrac{1}{2} \cdot (p_{i-1} + k_{i-1}) \end{aligned}$$

and so

$$p_i = \tfrac{1}{2} \cdot (p_{i-1} + k_{i-1})\ . \qquad (2)$$

By equality (2) we get easily

$$p_i = \tfrac{1}{2}(k_{i-1} + \tfrac{1}{2} \cdot (k_{i-2} + \tfrac{1}{2} \cdot (\ldots (k_1 + \tfrac{1}{2} k_0) \ldots)))\ . \qquad (3)$$

Since c_i is a random variable with values $0, 1$, we have $\mathbf{E}[\,c_i\,] = p_i$. Then, by linearity of expectation,

$$\begin{aligned} \mathbf{E}\left[\, \textstyle\sum_{i=1}^{n-1} c_i \,\right] &= \sum_{i=1}^{n-1} p_i \\ &= \tfrac{1}{2} \cdot k_0 + \tfrac{1}{2} \cdot (k_1 + \tfrac{1}{2} \cdot k_0) + \tfrac{1}{2} \cdot (k_2 + \tfrac{1}{2} \cdot (k_1 + \tfrac{1}{2} \cdot k_0)) + \ldots \\ &\qquad + \tfrac{1}{2} \cdot (k_{n-2} + \tfrac{1}{2} \cdot (\ldots)) \\ &= k_0 \left(\tfrac{1}{2} + \tfrac{1}{4} + \ldots + \tfrac{1}{2^{n-1}}\right) + k_1 \left(\tfrac{1}{2} + \tfrac{1}{4} + \ldots + \tfrac{1}{2^{n-2}}\right) + \ldots \\ &\qquad + k_{n-3} \left(\tfrac{1}{2} + \tfrac{1}{4}\right) + k_{n-2} \left(\tfrac{1}{2}\right) \\ &= k_0 \left(1 - \tfrac{1}{2^{n-1}}\right) + \ldots + k_{n-2} \left(1 - \tfrac{1}{2}\right) + k_{n-1} (1 - 1)\ . \end{aligned}$$

So we see that

$$\mathbf{E}\left[\sum_{i=1}^{n-1} c_j \right] = \sum_{i=0}^{n-1} k_i - 2^{1-n} \cdot K\ .$$

This concludes the proof of Lemma 1. □

We note that the expected value $\mathbf{E}[\,C(K)\,]$ depends on K in a very nice way – the bits of K influence separate bites of $\mathbf{E}[\,C(K)\,]$ (except the most significant part of $\mathbf{E}[\,C(K)\,]$).

2.3 Deviations from Expected Value on Random Inputs

The expected value $\varphi(K)$ may be experimentally determined as a mean value of a series of random experiments. Now it is crucial to determine how close this approximation is.

Lemma 2. *With probability at least* $1 - n \times 6.33 \times 10^{-5}$*, the observed value* C *does not deviate from its expectation in* N *experiments more than* $2n \cdot \sqrt{N}$.

Proof. Now we consider N stochastically independent experiments in which a random number is added modulo 2^n to the same (sub)key K. In each experiment some carry bits are set, e.g. like this ($n = 16$):

	c_{15}	c_{14}	c_{13}	c_{12}	c_{11}	c_{10}	c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
Exp. 1	1	0	0	0	1	0	0	0	0	1	1	1	0	1	0	0
Exp. 2	0	0	0	1	1	1	1	1	0	1	1	0	0	1	1	0
Exp. 3	1	1	0	0	0	1	0	1	0	1	0	0	1	0	0	0
...																
Exp. N	0	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0

($c_0 = 0$)

We shall think of columns separately, e.g. column i represents outcomes of independent experiments which yield result 1 with (some) probability p_i and 0 with probability $1 - p_i$. Variation of one such experiment is $p_i \cdot (1 - p_i)$. We sum up the elements in the column, let a random variable X_i denote this sum. Clearly, $\mathbf{E}\,[\,X_i\,] = N \cdot p_i$. We shall estimate the probability of large deviations from this expected value.

If $p_i \in \{0, 1\}$, then we always get the expected value. If $p_i \notin \{0, 1\}$, by Central Limit Theorem, $X_i \sim N(\mu, \sigma^2)$, where $\mu = Np_i$ and $\sigma = \sqrt{Np_i(1 - p_i)}$, and $N(\mu, \sigma^2)$ denotes normal distribution with expectation μ and variation σ^2. Since the maximum of function $p \mapsto p\,(1 - p)$ is in $p = \frac{1}{2}$ and equals $\frac{1}{4}$, we see that $\sigma \leq \sqrt{\frac{N}{4}}$. For a normal distribution the probability that observation deviates from expectation by more than 4σ is about $6.33 \cdot 10^{-5}$. Therefore with the probability of at least $1 - 6.33 \cdot 10^{-5}$:

$$\mathbf{E}\,[\,X_i\,] - 2\sqrt{N} \ \leq\ X_i \ \leq\ \mathbf{E}\,[\,X_i\,] + 2\sqrt{N}$$

for every $i \in \{1, 2, \ldots, n - 1\}$. By summing up n these inequalities we get

$$\mathbf{E}\,[\,C\,] - 2n\sqrt{N} \leq C \leq \mathbf{E}\,[\,C\,] + 2n\sqrt{N}\,.$$

(since $c_0 = 0$, we could sum $(n - 1)$ inequalities only; we sum n to simplify further calculations a bit)

This event occurs with probability at least $1 - n \times 6.33 \times 10^{-5}$ (more precisely: at least $1 - (n - 1) \times 6.33 \times 10^{-5}$). This concludes the proof of Lemma 2. $\square$

For $n = 16$ mentioned probability equals about 0.99898, for $n = 32$ about 0.99797.

3 Deriving Subkeys

3.1 Ideal Case

As we have shown above, the expected value of Hamming weight in N experiments equals

$$\begin{aligned} N \cdot n + N \cdot \textstyle\sum_{i=1}^{n} k_i + N \cdot (\sum_{i=1}^{n} k_i - 2^{1-n} K) \\ = N \cdot n + 2N \cdot (\textstyle\sum_{i=1}^{n} k_i - 2^{-n} \cdot K)\,. \end{aligned} \tag{4}$$

The values N and n are known, $K = (k_{n-1} \ldots k_1 k_0)$ is sought. For a moment let us assume that $N = 2^n$, the Hamming weights are measured exactly, and each input number occurs exactly once (just like in saturation attack). Let x be the total Hamming weight measured. Then:

$$x = 2^n \cdot n + 2 \cdot (2^n \sum_{i=1}^{n} k_i - K).$$

Since $K < 2^n$,

$$\sum_{i=1}^{n} k_i = \left\lceil \frac{x - 2^n \cdot n}{2^{n+1}} \right\rceil \tag{5}$$

and

$$K = \left(-\frac{x - 2^n \cdot n}{2} \right) \bmod 2^n\,. \tag{6}$$

As we can see, it would be possible to compute directly key K from x using information gathered in our experiment.

3.2 Taking Deviations into Account

Of course, the likelihood of an ideal situation considered above is rather small. We will get $N \cdot |K|$ ones (belonging to K), but we will not get exactly $N \cdot n/2$ ones before and after addition, nor exactly $N \cdot (\sum_{i=1}^{n} k_i - 2^{1-n} K)$ carry bits. On top of all, we have to consider measurement errors. Nevertheless, we shall see that it is still possible to gather significant information about the key.

Let us assume we want to find some information about n-bit key and can make $N = 2^m$ additions. We expect about $2^m \cdot n/2$ ones in the input (and the same amount of ones in the output) of the adding circuit. We shall think of input bits as of $2^m \cdot n$ independent random variables (say, X_i) with uniform distribution on $\{0, 1\}$. Then $\mathbf{E}\,[\,X_i\,] = \frac{1}{2}$ and $\mathbf{Var}[X_i] = \frac{1}{4}$. By Central Limit Theorem we may say that:

$$\textstyle\sum_{i=1}^{n \cdot 2^m} X_i \sim N(\mu, \sigma^2)$$

where $\mu = 2^m \cdot \frac{n}{2}$ and $\sigma^2 = \left(\sqrt{\frac{1}{4} \cdot 2^m \cdot n} \right)^2 = 2^{m-2} \cdot n$.

It is known that with probability close to 1 (that is at least $1-6.33\cdot 10^{-5}$) the distance between the actual value and expectation of a normal random variable is not greater than 4σ. In our case, $4\sigma = 4\cdot\left(\sqrt{2^{m-2}\cdot n}\right) = 2^{m/2+1}\cdot\sqrt{n}$. The same calculations are valid for the output ones.

In the previous subsection we have shown that the total number C of carry bits falls into the interval

$$\left[\mathbf{E}\,[\,C\,] - 2n\cdot 2^{m/2}\;,\;\mathbf{E}\,[\,C\,] + 2n\cdot 2^{m/2}\right]$$

with the probability of at least $1 - n\cdot 6.33\cdot 10^{-5}$.

Finally, there are errors due to measurement inaccuracies. We assume that error ε_i in experiment i is a random variable with the values in the interval $[-u, u]$ uniformly distributed. Then $\mathbf{E}\,[\,\varepsilon_i\,] = 0$ and $\mathbf{Var}\,[\varepsilon_i] = \frac{u^2}{3}$. The values of u depend on the equipment available, but for $n = 16$ it is sound to assume for instance that $u \le 2$ (error of magnitude 12.5%). By Central Limit Theorem

$$-4\cdot 2^{m/2}\cdot\frac{u}{\sqrt{3}} \;\le\; \sum_{i=1}^{2^m}\varepsilon_i \;\le\; 4\cdot 2^{m/2}\cdot\frac{u}{\sqrt{3}}$$

with the probability of at least $1 - 6.33\cdot 10^{-5}$.

Together, with the probability of at least $1-(n+2)\cdot 6.33\cdot 10^{-5}$ all kinds of deviations from the expected value sum up to a value not exceeding

$$2\cdot 2^{m/2+1}\cdot\sqrt{n} + 2n\cdot 2^{m/2} + 4\cdot 2^{m/2}\cdot\frac{u}{\sqrt{3}} = 2^{m/2+1}\left(2\sqrt{n} + n + 2\cdot\frac{u}{\sqrt{3}}\right)$$

We have to compare the above value with the expected value of the Hamming weight (4):

$$2^m\cdot n + 2^{m+1}\cdot\left(\sum_{i=1}^{n} k_i - 2^{-n}\cdot K\right).$$

In the last expression we are interested in bit positions $m-1, m-2, \ldots, m-n$ representing the key K . On the other hand, the deviations from the expected value and measurement errors influence the positions up to $m/2+1+\log(2\sqrt{n}+n+2\cdot\frac{u}{\sqrt{3}})$. For a very large m we can obviously read all key bits. However, we have to balance the number of experiments N and gaining key bits. In the next subsection we discuss some practically relevant settings of parameters.

3.3 Examples

Below we put concrete parameters for N, n and u and check how many key bits we may gain from the experiment.

We describe separately three kinds of deviations from the expected value: *error 1*– deviations from the expected Hamming weight of carry bits, *error 2*–

deviations from the expected value of the total Hamming weight of the inputs and outputs, *error 3*– deviations due to measurement errors. *Total error* denotes the estimated value of the largest deviations that may occur. *Signal level* 2^i means that i is the first bit position corresponding to the key K in the expression for the expected value of the total Hamming weight (we mean that the least significant bit of a binary number has position 0).

The following table describes the situation for $n = 16$ (the subkey length is like for IDEA) and $u \approx 2^{2.5}$:

N	error 1 level	error 2 level	error 3 level	total error	signal level	key bits found
2^{16}	$\sim 2^{13}$	$\sim 2^{12}$	$< 2^{9.2} \cdot u$	$< 2^{14}$	2^1	3
2^{18}	$\sim 2^{14}$	$\sim 2^{13}$	$< 2^{10.2} \cdot u$	$< 2^{15}$	2^3	4
2^{20}	$\sim 2^{15}$	$\sim 2^{14}$	$< 2^{11.2} \cdot u$	$< 2^{16}$	2^5	5
2^{22}	$\sim 2^{16}$	$\sim 2^{15}$	$< 2^{12.2} \cdot u$	$< 2^{17}$	2^7	6
2^{24}	$\sim 2^{17}$	$\sim 2^{16}$	$< 2^{13.2} \cdot u$	$< 2^{18}$	2^9	7
2^{26}	$\sim 2^{18}$	$\sim 2^{17}$	$< 2^{14.2} \cdot u$	$< 2^{19}$	2^{11}	8

For $n = 32$ and $u \approx 2^{3.5}$ we get:

N	error 1 level	error 2 level	error 3 level	total error	signal level	key bits found
2^{16}	$\sim 2^{14}$	$\sim 2^{12.5}$	$< 2^{9.2} \cdot u$	$< 2^{15}$	2^{-15}	2
2^{20}	$\sim 2^{16}$	$\sim 2^{14.5}$	$< 2^{11.2} \cdot u$	$< 2^{17}$	2^{-11}	4
2^{24}	$\sim 2^{18}$	$\sim 2^{16.5}$	$< 2^{13.2} \cdot u$	$< 2^{19}$	2^{-7}	6
2^{28}	$\sim 2^{20}$	$\sim 2^{18.5}$	$< 2^{15.2} \cdot u$	$< 2^{21}$	2^{-3}	8
2^{32}	$\sim 2^{22}$	$\sim 2^{20.5}$	$< 2^{17.2} \cdot u$	$< 2^{23}$	2^1	10
2^{36}	$\sim 2^{24}$	$\sim 2^{22.5}$	$< 2^{19.2} \cdot u$	$< 2^{25}$	2^5	12
2^{40}	$\sim 2^{26}$	$\sim 2^{24.5}$	$< 2^{21.2} \cdot u$	$< 2^{27}$	2^9	14
2^{44}	$\sim 2^{28}$	$\sim 2^{26.5}$	$< 2^{23.2} \cdot u$	$< 2^{29}$	2^{13}	16

It is quite astonishing that measurement accuracy u might be quite poor, but we are still able to find a significant number of key bits.

4 Vulnerability of Popular Algorithms

4.1 IDEA

IDEA encryption algorithm uses 18 subkeys of length 16 for addition. The key schedule is so simple that the subkey bits are directly the bits of the main key.

If our aim is to break completely the main key, we should find at least 4 (preferably 5) bits per subkey – that would yield 72 or 90 bits of the main key. Given that we could find the remaining bits of the main key by an exhaustive search (at most 2^{56} or 2^{38} trials). By inspecting the tables in the previous section we see that we need about 2^{20} experiments per subkey, and the error in measuring Hamming weight should not exceed some 12%. Note that subkey bits are **exactly** the bits of the main key.

4.2 Twofish

Twofish uses two 32-bit subkeys in each of its 16 rounds for addition. The key schedule, although not as simple as in IDEA, is still reversible. It can be easily shown that to find the main key it suffices:

128-bit key: 4 subkeys used for addition during the first two rounds and about $8 \cdot 2^8$ very simple operations,

192-bit key: 6 subkeys used for addition during the first three rounds and about $8 \cdot 2^{16}$ very simple operations,

256-bit key: 8 subkeys used for addition during the first four rounds and about $8 \cdot 2^{24}$ very simple operations.

As we know, finding more than 16 bit per 32-bit long subkey is rather infeasible; however, for breaking 128-bit main key we do not require more. We find 16 most important bits of 4 keys, and guess the rest. Then we reverse the key schedule, and test the keys. Of course, we should begin our guessing from value suggested by the procedure: if we have measured x_i, $1 \le i \le 4$ ones during 2^m additions, then we should start with $K_i = \left(-\frac{x_i - 2^m \cdot 32}{2}\right) \bmod 2^{32}$ (see equation 5), and then increase and decrease guessed keys by 1; also note, that expression $\left\lceil \frac{x_i - 2^m \cdot 32}{2^{m+1}} \right\rceil$ (see equation 6) gives us an approximation of K_i's Hamming weight; keys with Hamming weights "much" different than those may be checked later. In worst case we will have to check 2^{64} possibilities. Since an exhaustive search of 2^{64} queries is possible, we are able to recover the whole main key. The total number of operations required is in the range of 2^{64}. This is not few, but compared with the claimed 128-bit security it is very attractive.

Complete breaking 192 and 256-bit keys are not practically possible: on average it would require 2^{95} or 2^{127} attempts (note, that reversing of the key schedule is also more tedious). However, it is still a great improvement over the exhaustive search in the set of 2^{192} or 2^{256} keys.

4.3 MARS

MARS encryption algorithm [1] uses several 32-bit long subkeys for addition: in fact key addition is the first, and key subtraction (which may be also implemented as addition) is the last operation performed on data. Apart from that, another subkey addition takes place inside the E function used in each of 16 rounds of the keyed transformation. Summarily we have $4 + 16$ to $4 + 16 + 4$ subkeys used for addition. As shown above, at a cost of about 2^{44} encryptions we may find half of each subkey, i.e. 320 to 384 bits of total 1280 bits of an expanded key. Due to its complex structure, the key-schedule does not seem to be reversible, so an attack would require to execute key expansion scheme and falsify wrong key guesses. Although it does not seem very practical, it is still an improvement over the exhaustive search.

4.4 RC5 and RC6

RC5 and RC6 use addition as one of their building blocks. RC5 uses $2 + 2r$ subkeys for addition: two at the beginning and two more per each round. RC6 works similarly, on top of that it uses another two subkeys at the end of computation.

Case of RC5 and RC6 is in a way similar to MARS: since the key expansion scheme seems to be irreversible, our attack is rather a potential than an actual threat. It is worth noting, though, that there are no other subkeys than those used for addition. This property would theoretically allow us to break the cipher by finding all the subkeys. Let us assume that we have equipment of great (preferably: indefinite) accuracy, so we can measure (almost) exact Hamming weights (as in section **Ideal case**). Therefore we start with a saturated set of plaintexts and find the keys one by one from the first round to the last (we peel off consecutive rounds starting from the beginning; since we need to recover $2 + 2r$ keys, we want to minimize all the errors as much as possible). In that case, even though we still do not know the main key, we are able to duplicate the encrypting device or decrypt encrypted messages. Of course, such a theoretical attack would not be possible against Twofish which uses key-dependent permutation, or MARS which uses another subkeys for operations different from addition.

5 Conclusions

Our attack reveals that addition should be used with special care in hardware implementations of symmetric block ciphers. Potentially, a side channel attack may retrieve completely the secret key stored in a protected device, or at least provide significant amount of information on this key.

It also indicates that masking with random values the inputs of the arithmetic operations is not a universal technique that may be used as a defense against side channel attack on protected cryptographic hardware storing secret symmetric keys. Unexpectedly, masking may even help to perform the attack.

The attack presented is one more argument that the subkey schedules of symmetric ciphers should evolve into irreversible schemes.

References

[1] Burwick C., Coppersmith D., D'Avignon E., Gennaro R., Halevi S., Jutla C., Matyas S., O'Connor L., Peyravian M., Safford D., Zunic N., *MARS — A Candidate Cipher for AES*, `http://www.research.ibm.com/security/mars.html`. 101

[2] Chari S., Jutla Ch., Rao J. R., Rohatgi P., *A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards*, Second Advanced Encryption Standard (AES) Candidate Conference. 92

[3] Chari S., Jutla Ch., Rao J. R., Rohatgi P., *Towards sound approaches to counteract power-analysis attacks*, CRYPTO'99, Lecture Notes in Computer Science 1666. Springer-Verlag, 398-412. 92

[4] Coron J. S., *On Boolean and arithmetic masking against differential power analysis*, CHES'2000, Lecture Notes in Computer Science 1965. Springer-Verlag, 231–237. 92

[5] Daemen J., Knudsen L., Rijmen V., *The block cipher Square*, Fast Software Encryption'97, Lecture Notes in Computer Science 1267. Springer-Verlag, 149–165. 92

[6] Daemen J., Rijmen V., *The block cipher Rijndael*, `http://www.esat.kuleuven.ac.be/~rijmen/rijndael`. 92

[7] Goubin L., *A sound method for switching between Boolean and arithmetic masking*, CHES'2001, Lecture Notes in Computer Science 2162. Springer-Verlag, 3–15. 92

[8] Gandolfi K., Mourtel Ch., Olivier F., *Electromagnetic Analysis: Concrete Results*, CHES'2001, Lecture Notes in Computer Science 2162. Springer-Verlag, 251–261. 91

[9] Goubin L., Patarin J., *DES and Differential Power Analysis (The "Duplication" Method)*, CHES'99, Lecture Notes in Computer Science 1717. Springer-Verlag, 158-172. 92

[10] Kesley J., Schneier B., Wagner D., Hall Ch., *Side channel cryptanalysis of product ciphers*, Journal on Computer Security 8 (2000), 141–158. 91

[11] Kocher P., *Timing Attacks on Implementations of Diffie-Hellman*, RSA, DSS, and Other Systems. CRYPTO'96, Lecture Notes in Computer Science 1109. Springer-Verlag, 104-113. 91

[12] Kocher P., Jaffe J., Jun B., *Differential power analysis*, CRYPTO'99, Lecture Notes in Computer Science 1666. Springer-Verlag, 388–397, also: *Introduction to differential power analysis and related attacks*, `http://www.cryptography.com/dpa/technical`. 91

[13] Lucks S., *The saturation attack - a bait for Twofish*, `http://eprint.iacr.org/2000/046/`. 92

[14] Messerges Th., *Securing AES finalists against power analysis attack*, FSE'2000, Lecture Notes in Computer Science 1978. Springer-Verlag, 150–164. 92

[15] Rivest R., Robshaw M., Sidney R., *The RC6 Block Cipher*, `http://theory.lcs.mit.edu/~rivest/rc6.ps`.

[16] Schneier B., Kesley J., Whiting D., Wagner D., Ch. Hall, N.Ferguson, *The Twofish Encryption Algorithm: a 128-Bit Block Cipher*, Wiley, 1999, ISBN 0-471-35381-7.

[17] AES Development Effort, NIST, `http://www.nist.gov/aes`. 92

A Fully Compliant Research Implementation of the P3P Standard for Privacy Protection: Experiences and Recommendations

Giles Hogben, Tom Jackson, and Marc Wilikens

Institute for the Security and Protection of the Citizen
Joint Research Centre of the EC[**]
Marc.Wilikens@jrc.it

Abstract. This paper describes experiences gained from development of a fully compliant implementation of the W3C's XML based P3P standard. P3P aims to make privacy policies of web sites transparent for automated agents, and thereby to improve transactions of personal data on the Internet. We look at some of the most important issues that have arisen from our development work, including problems with the privacy preference standard, APPEL, before concentrating on issues related to end user assurance. We look at P3P usage scenarios to show that the current P3P standard has weaknesses in this area. The paper then considers possible extensions to P3P, which could provide greater assurance to end users and facilitate dispute resolution. In particular, we present an overview of a way for increasing assurance of a privacy policy's validity using signed XML.

Keywords: privacy enhancing technologies, P3P, XML Digital Signatures, secure electronic commerce, transaction management, security verification

1 Introduction

There have been several optimistic projections of rapid growth in e-commerce however the business-to-consumer (B2C) sector has failed to realise its expected growth potential. Surveys show that lack of information about privacy practices, and lack of trust in third party management of personal data are some of the most significant obstacles to the growth of e-commerce on the Internet. A survey performed by Consumers International [8] revealed that a third of investigated web sites made no mention of their privacy practices and the quality of actual practices where they existed had not been audited. A Harris poll [1] showed that when asked specifically about their online privacy concerns, respondents said they were most worried about web sites providing their personal data to others

[**] The views expressed in this paper are the authors' own, and may not be taken in any way as representative of those of the European Commission.

without their knowledge (64 percent) and web sites collecting information about them without their knowledge (59 percent).

The P3P standard has been proposed and developed by the World Wide Web Consortium (W3C) in response to the growing technological need to manage the privacy concerns of consumers engaging in on-line commerce. This paper will describe the experience gained in the development of a P3P implementation that is fully compliant with the standard, identify some of the outstanding problems, and outline recommendations for addressing them. It should be stressed that P3P offers only one (albeit important) technical solution in a wider spectrum of privacy enhancing technologies and security technologies that can be applied by IT systems for managing on-line privacy issues. In particular it provides a framework, which allows users and, importantly, automated agents on behalf of users to assess the terms and conditions of a web site's Privacy Policy via unambiguous machine readable statements which explicitly declare a the web site's data handling practices for personal data. These statements may be processed automatically, but visibly, before any personal data is sent to a remote web site.

After a brief introduction to P3P, we describe the JRC P3P reference implementation in section 2. Section 3 reviews some of the findings deriving form the implementation. Section 4 describes how some important questions of end user trust remain inadequately addressed by the current P3P specification. For example, the privacy statement of a web site might state that they will not contravene certain privacy principles (such as disclosure to third parties) but currently P3P does not contribute to the process of making this an agreement that can be enforced by providing non-repudiability. We claim that the P3P process has the potential to deliver far more in terms of a legally binding statement.

1.1 Information Privacy

An important concept in the context of online privacy protection and P3P is informational self-determination; i.e. empowering the user to assert his/her rights in the use of personal data [11]. According to EU data protection legislation, data are personal data when they deal with any information relating to an identifiable natural person. To determine whether a person is identifiable is subject to interpretation and therefore the technical definition of personal data is in continuous flux, not least because in the emerging Information Society an increasingly wide range of human characteristics (demographics, behaviour, biological features, etc.) will find themselves reflected in some kind of electronic equivalent. For the purpose of web site interactions, personal data can be broadly categorised in terms of the following categories: Communications traffic data indicating the source and destination of a communication (e.g. http headers, clickstream data, cookies, user password, etc); demographic data (e.g. name, address, etc.); contextual data, i.e. additional data needed for processing a particular transaction (e.g. credit card, income, health status, location, etc.).

1.2 Platform for Privacy Preferences – Brief Introduction

The W3C specification document gives the following description of P3P [3]: "The Platform for Privacy Preferences Project (P3P) enables Web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents. P3P user agents will allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices when appropriate. Thus users need not read the privacy policies at every site they visit." The aims of P3P can be summarised as follows:

- Transparency: To increase trust in data transactions on the Internet by providing statements of Data Protection policies.
- Automation: To facilitate data flow by making statements machine readable.
- Dispute resolution: To facilitate this by providing unambiguous statements.

A schematic overview of P3P is given in figure 1. In words:

- A web service/site (1.) (e.g. e-commerce shop) defines its Privacy Policy that is then translated into a P3P XML version often using an off-the-shelf tool. The privacy policy specifies a web site's data processing practices of the personal data of its customers. Typical issues specified are length of data retention, disclosure to unrelated third parties and user access to their personal data.
- The user defines their Privacy user preferences (4.) which are translated into a P3P XML version possibly using a graphical tool such as the JRC ruleset editor[1]. This typically specifies a set of data processing characteristics to look for in P3P policies and behaviors to execute if they are matched. The characteristics that the rules look for can be any of those possible within P3P e.g. user access rights, data retention, etc.
- For any http request by the user, before data is transmitted, the agent fetches the P3P policy (3.) and evaluates it against the user's preference set.
- Depending on the evaluation result, data such as http header info or demographic data (6.) may be released to the web site's data store and/or the client may be given information on evaluation results. Note that current options in P3P only specify that requests SHOULD be blocked, executed or limited. There is no interaction with a user data store or granular information filtering.

2 Description of the P3P Reference Implementation

2.1 Objectives

The JRC has developed a complete implementation[1] of the P3P standard for the purposes of developing research extensions to P3P, as an educational platform

[1] See the JRC P3P resource center `http://p3p.jrc.it`

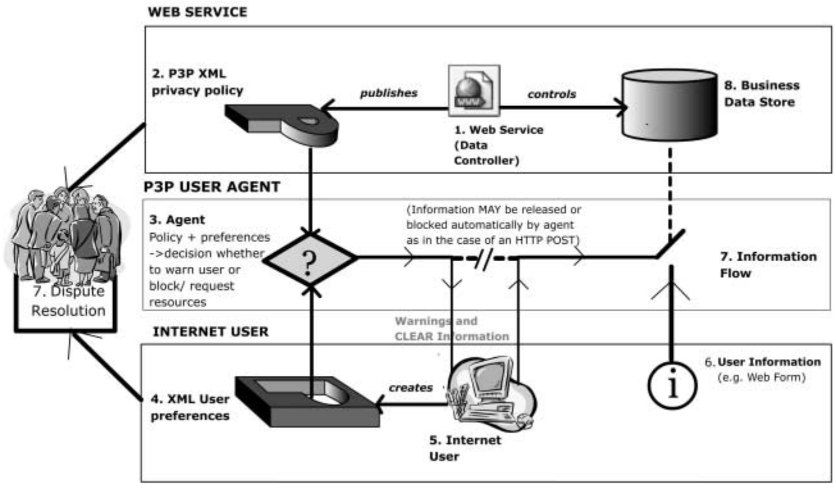

WEB SERVICE
2. P3P XML privacy policy
publishes
controls
1. Web Service (Data Controller)
8. Business Data Store
P3P USER AGENT
3. Agent Policy + preferences ->decision whether to warn user or block/ request resources
(Information MAY be released or blocked automatically by agent as in the case of an HTTP POST)
7. Information Flow
7. Dispute Resolution
INTERNET USER
Warnings and CLEAR Information
6. User Information (e.g. Web Form)
creates
4. XML User preferences
5. Internet User

Fig. 1. Schematic overview of the P3P Process

and as a way of investigating the level of the standard's compliance with EU Personal Data Protection legislation. This implementation was the first to be fully compliant to the standard, and was adopted by the W3C[2] as a reference model to demonstrate the maturity of the standard. A demonstration site and evaluation environment are available to provide a broad range of assessment and familiarisation functions.

2.2 Architecture

The implementation is written in Java 1.2 with a proxy architecture, ensuring compatibility with most browsers. A modular architecture allows easy experimentation with different components. For example the rule based evaluation system is completely decoupled from the proxy handler.

1. APPEL preference matching component: The intelligent agent which performs matches between the web site P3P privacy policy and the user's APPEL [4] privacy preferences. APPEL allows a user to express his/her privacy preferences in a set of XML rules (called a ruleset), which can then be used by a user agent to make automated or semi-automated decisions regarding the acceptability of machine-readable privacy policies from P3P enabled web sites. Only one rule is permitted to fire in any set, thus strict ordering is crucial. This is a drawback in our view because although there may be many reasons for a policy to be rejected within one ruleset, only one is given to the user. It does however make for faster performance and obviates the possibility of conflicting behaviors.

[2] See the W3C web site: `http://www.w3c.org/p3p`

2. Preference creation component (ruleset creation): A tool allowing users to translate their privacy preferences into XML format.
3. Syntax validation component: Performs numerous syntax checks on the Privacy Policy and the privacy preference ruleset.
4. Proxy component: Mediates between client and server, filtering content and providing feedback to the user, according to P3P. Also handles the different users accessing the machine simultaneously so that content is sent back only to the user who requested it. Manages user profiles, each with a username, password, preferences and history. This is thus a simple identity management system.
5. Policy discovery component. Looks for Policy Reference Files (PRF's) (files specifying which policies apply to which resources) and decides which P3P policy applies to which resource.

The set up also contains a mock-up of a typical P3P enabled commercial web site to demonstrate a realistic process and end user interactions.

3 Main Findings Deriving from the P3P Implementation

3.1 User Interface and Performance Issues

Perhaps the most interesting part of the development effort was in trying to build an interface through which users can easily set up their privacy preferences. P3P can express various aspects of human readable privacy policies in an unambiguous machine readable XML format. Building on this, APPEL is able to create rules, which will match any aspect of P3P's granular structure. Therefore in setting up an interface for creating APPEL rules, we were forced to present a very large number of possibilities to the user in order to allow them to match any aspect of a P3P policy. In terms of ECA (Event, Condition, Action) rule theory the problem is that the rule condition – is very complex.

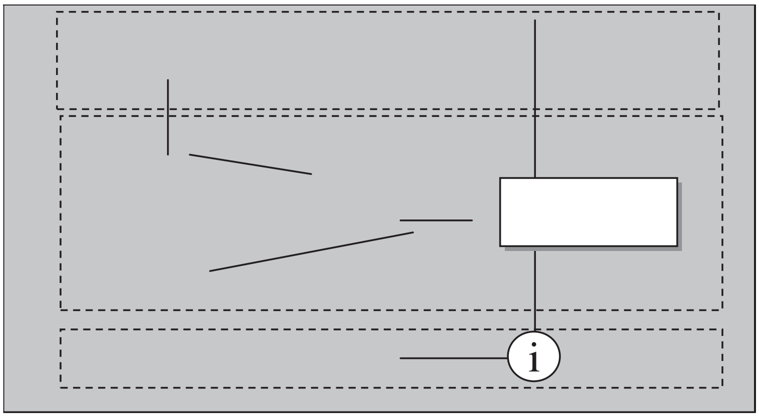
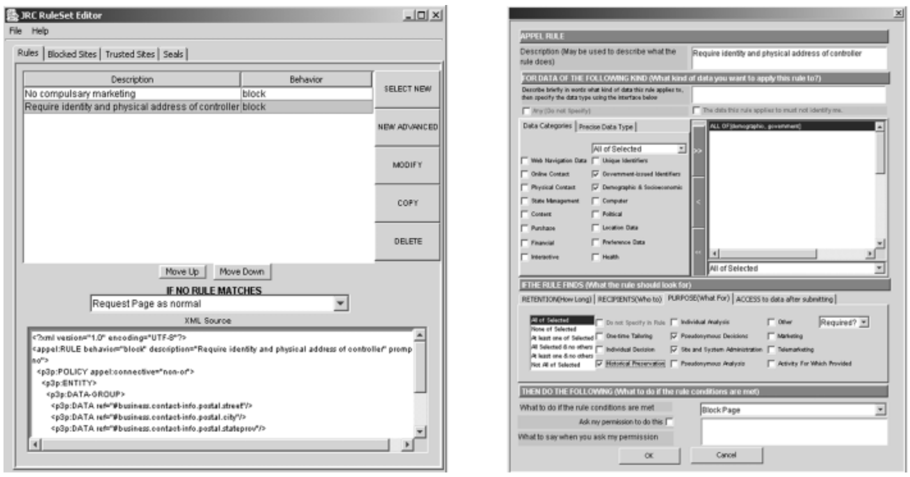


Fig. 2. Architecture components of the JRC P3P



Description	Behavior
No compulsary marketing	block
Require identity and physical address of controller	block

IF NO RULE MATCHES: Request Page as normal

XML Source

```
<?xml version="1.0" encoding="UTF-8"?>
<appel:RULE behavior="block" description="Require identity and physical address of controller" promp
no">
  <p3p:POLICY appel:connective="non-or">
   <p3p:ENTITY>
    <p3p:DATA-GROUP>
     <p3p:DATA ref="#business.contact-info.postal.street"/>
     <p3p:DATA ref="#business.contact-info.postal.city"/>
     <p3p:DATA ref="#business.contact-info.postal.stateprov"/>
```

Description (May be used to describe what the rule does): Require identity and physical address of controller

FOR DATA OF THE FOLLOWING KIND (What kind of data you want to apply this rule to?)

Describe briefly in words what kind of data this rule applies to, then specify the data type using the interface below

☐ Any (Do not Specify) ☐ The data this rule applies to must not identify me.

All of Selected

☐ Web Navigation Data	☐ Unique Identifiers
☐ Online Contact	☑ Government-issued Identifiers
☐ Physical Contact	☑ Demographic & Socioeconomic
☐ State Management	☐ Computer
☐ Content	☐ Political
☐ Purchase	☐ Location Data
☐ Financial	☐ Preference Data
☐ Interactive	☐ Health

ALL OF(demographic, government)

All of Selected

IF THE RULE FINDS (What the rule should look for)

All of Selected / None of Selected / At least one of Selected / All Selected & no others / At least one & no others / Not All of Selected

☐ Do not Specify in Rule	☐ Individual Analysis	☐ Other	Required?
☐ One-time Tailoring	☑ Pseudonymous Decisions	☐ Marketing	
☐ Individual Decision	☑ Site and System Administration	☐ Telemarketing	
☑ Historical Preservation	☐ Pseudonymous Analysis	☐ Activity For Which Provided	

THEN DO THE FOLLOWING (What to do if the rule conditions are met)

What to do if the rule conditions are met: Block Page

Ask my permission to do this ☐

What to say when you ask my permission

Fig. 3. Simple and Advanced rule creation interfaces

User Interface Issues with APPEL A quick survey shows that existing commercial implementations have not implemented full versions of APPEL. One of the possible reasons for this is that the possibility space provided by APPEL is too large for any application to be able to offer an easy to use interface for creating privacy rules. For example it is possible to state for any one of over 100 data types whether their collection is optional or not. To give the ordinary user the ability to control the optionality of every one of the data types would present him with an impossibly complex array of choices. However for experts having to use APPEL in a more professional capacity such as regulators, the ability to control such choices is vital.

Any user interface therefore needs to be firmly divided between the expert and the general user. After creating our Rule creation GUI, we were confident that we had an interface of use to informed experts. We also found that it was possible to create an interface for general users based on selecting from a set of predefined rules (e.g. "block all cookies which link to personal information"). The main obstacle we needed to overcome was making sure that the complexity of full rule creation was not presented to ordinary users, whilst still allowing it as a possibility for advanced users. User testing showed that it was very important to separate the condition and action parts of a rule for it to make sense to users. Figure 3 shows the two sections of the GUI, one based on choosing between a simple selection of predefined rules (aimed at the general user), the other offering complete control over all aspects of a privacy rule (aimed at expert users). The second interface also separates the choices clearly into those to do with the condition and those to do with the action.

Complexity of Algorithm Another possible reason for the lack of full implementations of APPEL is that in order to provide a completely general matching algorithm capable of dealing with any possible privacy preference within the semantic space provided by P3P, a complex sub-tree matching algorithm is required. This is costly to implement in terms of development time and difficult to make efficient.

At first sight, the APPEL algorithm appears to be very heavy because it is highly recursive. This may have put off many implementers. However, because P3P policies never reach more than 5 levels in depth and generally only extend to 4, this recursivity does not produce excessive load. We found that with a full policy validation in our Java implementation, the APPEL evaluation process on a standard PC can take of the order of 1 second for a real world ruleset and policy. However, timing tests revealed that this was more due to inefficiencies in document parsing than in excessive load created by the algorithm. The actual matching process only took typically 20 milliseconds. We would suggest therefore that implementers should not be put off using APPEL because of its apparent complexity. Already some implementers have been successfully using our evaluator which can be used as a stand alone component.

Performance in P3P User Agents The fact that most implementations of P3P user agents (most notably Microsoft's IE 6) have chosen to implement a cut down version of P3P, and in general policies are processed after resources have been loaded, highlights performance problems faced by implementers of P3P. The actions of a P3P user agent for a typical page can involve 6 extra http calls for PRF's, policies for embedded resources etc.

W3C's proposed solution to this has been the P3P compact policy. This is an abbreviated version of the P3P policy contained in the http header, which allows a much quicker evaluation time. However these only apply to cookies and are therefore by no means a complete solution.

One solution is to capitalize on the fact that many companies are tending to use standardized ("catch-allö) privacy policies for reasons of legal security and to save work in creating policies. For example, on Yahoo.com, the policy, which covers the majority of pages within their domain states that information collected may be used for opt out telemarketing. But on most pages within Yahoo, no personal data is actually collected. This reflects the fact that it is easier for companies to create a "lowest common denominator" policy, which will cover anything that might possibly happen on their site, than to create policies tailored to individual pages.

If this trend develops, it may be that companies will use standardized privacy policies kept by third parties at URI's, which may be well known to a user agent, in the same way as data documents can use standard XML schemas. This would allow a major performance enhancement because clients could then store a list of trusted policy URI's and simply match against these rather than having to fetch a policy and do complex sub-tree matching. A related solution to the problem would be possible if the use of trusted seals became widespread. P3P, in its disputes resolution tag, allows policies to specify a trusted seal. Were these to be widely adopted, as seems likely, it would again reduce the policy matching load to a simple string match.

Other possible avenues for performance enhancement might include the extension of compact policies to cover all resources, and the use of binary XML.

3.2 APPEL Specification Issues

Ambiguity of Logic Inference System At present it is possible to write two semantically equivalent P3P policies where one will be blocked by a given rule and the other will not. This is clearly unacceptable. For example, the following rule looks for any information which is not the user's IP address or user agent string and blocks resources which ask for it.

```
<appel:RULE behavior="block">
  <p3p:POLICY>
       <p3p:STATEMENT><p3p:DATA-GROUP appel:connective="non-and">
            <p3p:DATA ref="#dynamic.clickstream.clientip.fullip"/>
            <p3p:DATA ref="#dynamic.http.useragent"/>
       </p3p:DATA-GROUP></p3p:STATEMENT>
  </p3p:POLICY>
</appel:RULE>
```

This RULE will cause a block behavior for the following web site policy (only relevant parts quoted),

```
<POLICY>
     <STATEMENT><DATA-GROUP appel:connective="and">
          <DATA ref="#dynamic.clickstream.clientip.fullip"/>
      <DATA ref="#dynamic.http.useragent"/>
      </DATA-GROUP></STATEMENT>
</POLICY>
```

but not for this one

```
<POLICY>
     <STATEMENT><DATA-GROUP>
            <DATA ref="#dynamic.clickstream.clientip.fullip"/>
     </DATA-GROUP></STATEMENT>
     <STATEMENT><DATA-GROUP>
            <DATA ref="#dynamic.http.useragent"/>
     </DATA-GROUP></STATEMENT>
</POLICY>
```

Note the presence of the "non-and" connective, which means "only if not all sub-elements in the rule are present in the sub-elements of the matched element in the policy". This is true for the first policy snippet but not the second, which given that they have the same meaning is clearly unacceptable. We will look at solutions which address this problem below.

Expressive Inadequacy of Logical Connectives The problem of ambiguity outlined above is just one example of the inadequacy of the logical system contained within APPEL. One of our lines of research will shortly be in bi-lateral negotiation systems. This will require a much more sophisticated capability in terms of logic and inference.

For instance we would like to be able to express the following: *If the privacy policy says it will retain my information "indefinitely", try changing this to "for the stated purpose only" and send the policy back to see if the server's logic engine will agree to this in exchange for the same services.* At present, APPEL contains only the ability to match arbitrary sub-trees beginning at the POLICY's root node, and is not well suited for such inferences.

Proposed Solutions to Problems with APPEL Any rule system able to match the full range of possibilities within P3P will by always be as complex as P3P. So we do not believe simplifying APPEL is an answer. Instead, we suggest the following lines of research:

1. As a quick fix, use a standard query language for the condition matching part of APPEL. Instead of using the somewhat quirky APPEL connective system and recursive matching algorithm the rule condition could be specified by an XPATH [12] query. These query languages are designed to match arbitrary node sets with high efficiency. They have the advantage that developers are familiar with them and efficient algorithms exist to execute the queries. As it has become very clear that APPEL is not a language that will be written by anyone other than developers or ordinary users using a GUI, this is clearly the best approach.

E.g. a rule in this format, which would solve the above ambiguity problem would be:

```
<appel:RULE behavior="block" prompt="yes" promptmsg="Rule found policy
       using your home info beyond current purpose ">
       <appel:MATCHQUERY query=
       "//DATA[not(substring(@ref,' dynamic.clickstream.clientip.fullip')
       or substring(@ref,' dynamic.http.useragent'))]"
       querylanguage="XPATH">
</appel:RULE>
```

2. Explore moving APPEL into an Resource Description Framework (RDF) [2] based system, which would relate rules to a Darpa Agent Markup Language (DAML) [7] ontology (a specific privacy ontology would have to be developed in this case.). We cite the following reasons for this:

- Semantic Web technology, as its name suggests, supports query and matching systems based on an underlying semantics. In other words queries designed to match underlying meaning and not syntactical patterns. It is therefore unlikely that the ambiguity problems outlined above would occur.
- An RDF schema already exists for P3P and several members of the P3P working group have suggested that P3P 2.0 should use RDF/Semantic Web syntax.

- Because it is based on subject, predicate object structures, RDF is well suited to expressing natural language statements such as "site x will retain your information indefinitely" and using them in intelligent inference engines like rule matchers.
- The RDF community is already rich with Rule matching systems like ruleML [9] which have a much wider development community than APPEL.

3.3 P3P and EU Data Protection Legal Compliance Issues

One of the most pressing questions for P3P in the context of the on-line business sector is the degree to which the P3P standard is able to support or comply with the different data protection legislation and regimes. P3P should be considered as a toolbox that can be configured to specific data protection regimes and as such the coding of the level of protection is up to the implementation. From a EU perspective, there are three important issues in this regard:

1. Is P3P able to provide support for e-commerce providers who wish to implement personal data handling practices in an EU compliant way?
2. Can P3P permit end users to express their privacy preferences as a statement of their rights under EU legislation?
3. How will P3P operate in trans-border environments where the privacy legislation may change to reflect member state individual interpretations of the EU directives or across global borders outside of the jurisdiction of the EU directives?

The issue of EU compliance for e-commerce providers is complex. The EU Directives set out principles of transparency for data collection, which include concepts such as purpose specification, collection minimization and purpose limitation. The EU directives do not make any specific reference to the role of Privacy Policies, which of course is the focus of the P3P standard. Clearly, Privacy Policies can contribute to the task of informing users of the data processing practices of a web site, and of defining the purposes of data collection and subsequent usage. To this extent, by automating the exchange of privacy policies, P3P can support a web site in specifying its practices to be EU compliant. However, there are some key pragmatic issues that are not yet clear in regard to P3P's compliance. The directives state that information as to purpose of data collection and usage should be declared at the point at which data is collected, so that end users are explicitly informed at the moment of data submission. P3P does not, by default, operate in this mode. There is no explicit statement, rather it is implicit in the agreement reached by the APPEL parser during the exchange of the privacy policy and a user's ruleset. As such, privacy policies do not go far enough in meeting the requirements of the EU directives, unless they are expressed and exhibited explicitly at the points of data collection. There is nothing in principle to stop P3P implementations doing this, but it is not the currently envisaged model for its deployment.

The second issue is the expression of a user's preferences in a way that reflects their rights granted under EU data protection legislation. In this respect, P3P can play a useful role, and in the course of our development work, we have produced a prototype Ruleset that reflects most of the principles of the EU directives. The APPEL language used to create the Rulesets is sufficiently flexible and expressive to facilitate this. Although it would be beyond the capabilities of most consumer users to define such a Ruleset, there is nothing to prevent them downloading a default EU Ruleset from a local agency, such as the member state Data Protection Agencies. Our EU ruleset was produced and edited using the APPEL Ruleset editor tool developed within the project. With the EU Ruleset configured for the P3P proxy server, we visited a large number of the websites that are currently advertised as P3P compliant. We observed that almost every site failed to comply with the preferences expressed in the Ruleset. The principle reason for this was that they did not allow user access to all data collected, as stipulated in the directive. Whilst this may initially be seen as a very negative result, it does have some positive implications. What it shows is that by virtue of its unambiguous transparency, P3P will bring such issues much more out into the open. In order to compete in an internet environment where P3P is used, companies will be forced to state unambiguously what their practices are, and end-users, whether consumers or Data Protection Auditors will be able to see almost instantly those whose privacy policies are not conforming to the EU Data Protection Legislation and why.

The third issue of global cross border browsing is not so much a technology issue as a policy issue. If users wish to express their privacy preference in terms of the EU directives it is to be expected that they will not find sites that are compliant outside of the EU. End users need to be aware of this fact, and should be clearly educated that their privacy rights may vary significantly if they wish to browse or participate in e-commerce outside of the EU. However, government bodies such as the EC may wish to specify minimum levels to which a particular P3P implementation should be configured. Finally, it is worth noting that P3P can in no way offer any enforcement of privacy statements. Whether or not statements actually reflect practice is not an issue the protocol can hope to address. Vice versa, if a resource has no P3P policy, it also does not mean that it does not comply with European law. So in the respect of law enforcement, P3P can never be more than an aid to transparency.

4 Current Limitations and Outlook Concerning End User Assurance

With the increasing prominence of the P3P standard, the number of P3P policies is increasing to the point where they may become a standard feature of commercial web sites. However, there are still outstanding problems related to assuring the integrity and non-repudiation of these policies.

4.1 P3P Examples of Risk Scenarios

In order to clarify these problems, we provide three scenarios to describe how the P3P process might be abused to the detriment of both end users and businesses:

Scenario 1. *I receive a large amount of unsolicited email from various sources unknown to me, despite the fact that I only ever give my email address away to sites whose P3P policies promise only to use my email address to carry out the stated purpose of the transaction. I seek to take legal action using the P3P policy, which was available on one of the offending sites at the time I made the transaction, as evidence. The corporation I am suing denies that it published this policy at the time I sent my details. I have no way of proving what Privacy Policy was published at the time and as a consequence my legal action is not successful. Furthermore, if I visit such a site, I am put off using the service because I know that I cannot ever prove that the company made the statements on its P3P policy.*

This example highlights up two specific repudiation issues in regard to the deployment of P3P:

1. If a user cannot prove that a company published a specific privacy statement, then a privacy policy is of limited benefit to him/her in pursuing a legal action;
2. If a user cannot prove the source of a loss of privacy then any number of assurances are of limited benefit to him/her.
3. Even if legal action is not actually taken, the fact that any action would be unlikely to stand up in court is enough to put many people off using a web service.

Scenario 2. *I operate a web site and collect sensitive personal information which I use for marketing purposes. In order to comply with the European Directive, I ensure that the user gives his/her consent for this. After some years, I receive a court summons from a user saying that I used their information without consent. As I have no signed agreement from the user, I have no way of proving that this was not the case.*

This example shows

1. The importance of the issue of legal consent in this arena. As consent is such an important component of compliance to the EU directives, it seems important to have some means of proving that it was given.
2. That non repudiatable assurances may not only be useful to end users but also to commercial organisations. If one considers that P3P policies are subject to security problems and that court cases may be brought on their basis, it also seems likely that there might be demand from the corporate side for P3P policies, which can provide watertight evidence.

Scenario 3. A possible extension to P3P is to include bi-lateral negotiation, which allows the end user to negotiate data processing practices (and changes to the P3P policy) for services or goods. This is also mentioned in the current W3C specification [4] as a possible route for extending the protocol. A possible specification for this type of extension is treated in detail by R.H. Thibadeau [6]. One scenario, which arises from this inclusion of bi-lateral negotiation is the following:

Private individual x's computer negotiates with server Entity y, according to the proposed P3P negotiation protocol. Private individual x agrees to give away his email address if and only if server y assures him that it will not be given to third parties. Later Private individual x discovers that Server Entity y has in fact given his information away to a third party. Server Entity y claims that he did not make such an agreement and an unresolvable dispute follows.

This example shows that policies which need to be assured may not always be fixed documents, but may be produced dynamically as part of a bilateral negotiation process, hence requiring more care in the consideration of loading on servers.

4.2 Extensions to the P3P Specification to Deal with Assurance Issues

In the following sections we discuss extensions to the P3P standard, which could provide it with more enforcement rigour, and consequently provide greater assurance.

It should be noted however, that any modifications to the P3P standard need to take into account the constraints imposed by the computing environment of end-users. This often includes a low bandwidth network environment and low-to-mid range PC platforms, particularly in the European context. They also need to be accommodate the fact that although end-users are keen to protect their privacy, they are also unwilling to accept computational overheads that might interrupt the flow of their browsing experience.

Non-repudiable Privacy Policies via P3P A principle problem for P3P is that if a company's practices contravene its stated privacy policy, there is little technical framework to prove that a company made the statements which may have existed on its server at a given time. I.e. it is too easy for a company to repudiate its policy.

While P3P does increase the level of trust felt by consumers by providing more transparent and unambiguous information, it does not however provide any assurance as to the authenticity and integrity of this information.

XML signatures offer an ideal solution to the problem of making a policy at a given URI non-repudiatable. XML signatures provide the opportunity to introduce assertions such as "X assures the content of this document" into the semantics of signed material. Also since P3P is entirely expressed in XML, it is pragmatic to use the XML version of asymmetric digital signatures to provide this assurance. The following section defines in detail how this might be achieved.

We examine and build upon the proposals of Reagle [5] for the inclusion of XML digitally signed [10] policies within P3P. As Reagle has already set out most of the mechanisms for achieving this, we make only three minor additions to the technical specification. Our main aim is to look at possible technical problems with the use of the XML signature extension, and their solutions.

XML Digitally Signed Policies P3P enabled servers could have the possibility of providing an XML digital signature as part of their policy, or as a separate document referenced within the policy. This is easily accomplished provided that the correct syntax is incorporated into the P3P specification, as shown by Reagle. We provide an example, see Annex 1, adapted from Reagle, which we have extended in the following ways:

a) We have added X.509 certificate bag to provide full non-repudiatability.
b) We have included a time stamp to comply with EU regulations.
c) By requiring an additional signature over the PRF, which details which resources the policy applies to. Any signature that does not assure this information loses much of its legal significance. Note also that this signature cannot be bundled in with the policy signature because several PRF's may refer to the same policy. Furthermore, the person responsible for producing policy signatures may not even know the location of PRF's referring to the policy (in the case of a standard issue policy used by third parties.) We suggest the addition of a "DISPUTES" element to the PRF identical to the DISPUTES element in the P3P policy which allows the specification of a signature URI using the validation attribute.

The P3P process has 2 main components on the server; an XML policy and an XML PRF, which binds policies to resources. Semantically therefore, a P3P statement is a combination of the policy and the PRF, which binds a policy to a resource at a given time. The PRF, like the policy has a validity timestamp.

However, Reagle's P3P extension does not include any binding signature for the P3P PRF. This is an oversight, because omission of a signature binding the policy to the resource it applies to negates the non-repudiability of the statements being made. The importance of the PRF cannot be ignored. We therefore suggest that a signature also be provided for any PRF's used. We show, however, in the example signature (Annex 1) the necessary extensions for a signature to be bound to a PRF.

Computational Overhead in Processing Signed Policies The P3P process involves the exchange and processing of policy files. So any increase in complexity for policies, such as addition of digital signatures, implies an increase in computational overhead for processing the data. This applies to both server and client.

On the server, the additional processing can be considered negligible. There is no dynamic load on the server from the encryption processes involved. This is because the Privacy Policy statement is usually static (that is, a site's privacy policy does not change regularly) hence the signing process only needs to be carried out once, when a new policy is created or edited. Once signed, the policy signature can be posted on a web site with no further load than is the case with a standard XML policy. For the client, there is a more significant computational overhead, as it must verify the policies received from a remote server. In tests, we observed that signing and verification operations for a client, using a typical PC, are order 100ms. Which, in isolated cases will have minimal impact on a user's browsing experience.

However, there are certain scenarios, which could theoretically lead to the processor loading becoming significant for both server and client. For example:

- If the client had to handle multiple policies, assigned to different resources within a web site, it could make the browsing process unacceptably slow. For example a page might require the client to verify signatures from several different sources to handle a page's embedded resources.
- If the client is in a mobile phone or other device with low processing capability.
- Conversely, for servers with high traffic loads, if there is an additional http request to a signature resource with every http resource requested, then the additional load becomes significant.
- In the case of bi-lateral negotiated policies, some form of filtering is an essential component in the process. In this case, P3P policies could be dynamically altered to suit the user's preferences, leading, in extreme scenarios, to a different policy being created for every request. The resultant load on the server would be a processing bottleneck.

Fortunately, there are a number of pragmatic ways in which the above issues can be constrained, to limit the number of times such operations would have to be carried out (both on the server, and the client). The first is by a proposed simple extension using the rule system. This applies initially to APPEL [4] but can easily be generalized to any subsequent rule system.

In the basic version, a P3P policy is retrieved and matched against an XML preference set according to the APPEL protocol. Once this matching process has been carried out, there are only 3 basic outcomes: page block(6.), page request (7.) and page request with http header limitation (8.). Each of these can be optionally authorized by a user prompt. An example rule is given above in section on logical ambiguity of APPEL rules.

Reducing Signed Policy Processor Loading in P3P Using Rule Action It would be easy to extend this syntax to solve the problem of loading and redundancy for signature verification in P3P. In the extended version (dotted lines, figure 4), an extra but OPTIONAL decision step has been added (4.), which may lead either to a page block (5.) in the case of an invalid or non-existent signature, or to the continuation to the standard behaviors (6,7,8). This again may be automated or may involve input from the user in the form of a prompt (for example "this page does not have a valid signed policy – do you wish to proceed").

We suggest that the addition of a signaturerequired attribute to the rule element, rules could limit the necessity of signature verification as in this example:

```
<appel:RULE prompt="no" behavior="request" signaturerequired="yes">
<p3p:POLICY appel:connective="and"><p3p:ACCESS/></ p3p:POLICY>
</appel:RULE>
```

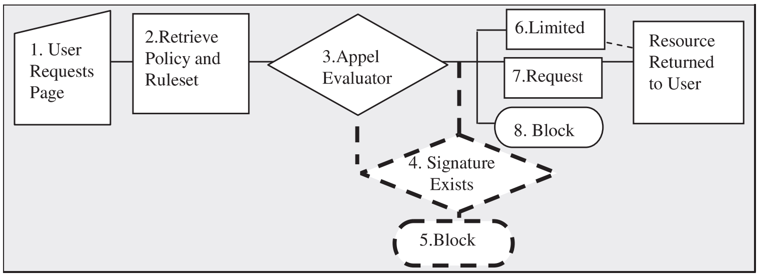


Fig. 4. APPEL matching process. Dashed lines show our proposed extension to process

Such a rule would then state in plaintext: *"If the agent finds a statement of how l access my data after it has been submitted., require that there is a valid XML policy and prf signature, retrivt and store it."*

This change reduces processor loading and network traffic. It also makes it easier to retrieve signatures when needed because the user stores only those signatures deemed necessary. A suggestion for solving problems with limited device capabilities in scenarios such as consent proof where client signatures might be required, would be to use CC/PP technology (Composite Capabilities/Preferences Profile) [13] which allows servers to make demands on devices according to their capabilities.

5 Conclusion

P3P has been a great boost to raising awareness of privacy on the Internet. By giving web users a tool for controlling the disclosure of personal data, it provides one component in a spectrum of PET's. The JRC reference implementation of the P3P standard and a neutral platform for its evaluation allows the privacy community to assess P3P. So far, we have identified a number of challenges on the road to full maturity of the standard:

- Creation of an easy to use interface for creating XML privacy preference rulesets, which will be useful to both the ordinary and the expert user.
- Creation of a language for the storage of privacy preferences which solves the problems inherent in the current APPEL specification, namely inadequate logical expressivity and ambiguity.
- Solution of performance problems faced by an implementation, which is faithful to the standard.
- Continuous evaluation of the use of P3P for enhancing compliance of privacy practices with EU privacy regulation and other regulatory initiatives.

Data flow and consumer confidence on the Internet is severely hampered by privacy related factors, which can easily be addressed. The lack of trust in privacy policies and established mechanisms for legal recourse based on the policies, are issues, which can be addressed by extensions to P3P. One practical extension, which has been discussed in detail, is the application of XML digital signatures. We suggest XML digital signatures within the framework of P3P in combination with mechanisms for automatic selection of which resources might require such evidence.

References

[1] Harris Poll carried out for National Consumer League of America October 2000 – see http://www.nclnet.org/pressessentials.htm 104

[2] RDF Specification. Resource Description Framework (RDF) Model and Syntax Specification http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ 112

[3] The Platform for Privacy Preferences 1.0 (P3P1.0) Specification W3C Working Draft 28 September 2001, http://www.w3.org/TR/2001/WD-P3P-20010928/ 106

[4] A P3P Preference Exchange Language 1.0 (APPEL1.0), W3C Working Draft 26 February 2001, http://www.w3.org/TR/2001/WD-P3P-preferences-20010226 107, 115, 118

[5] Joseph Reagle, A P3P Assurance Signature Profile, W3C Note 2, February 2001, http://www.w3.org/TR/xmldsig-p3p-profile/ 116

[6] Thibadeau, Robert. 2001 "Privacy Science", Paper presented at the European Commission Workshop on Privacy and Identity in the Information Society: Emerging technological challenges, October 4-5, Brussels, Belgium, http://yuan.ecom.cmu.edu/psp/privacy2001.ppt. 115

[7] Darpa Agent Markup Language – W3C specification – soon to become OWL (Ontology Web Language) http://www.w3.org/TR/daml+oil-reference 112

[8] Scribbins, K., "Should I Buy? Shopping online 2001: An international comparative study of electronic commerce", Consumers International, http://www.consumersinternational.org/CI_Should_I_buy.pdf, ISBN 1902391365, 2001. 104

[9] International XML rule language initiative, home page http://www.dfki.uni-kl.de/ruleml/ 113

[10] XML Digital Signatures – W3C Recommendation http://www.w3.org/TR/ xmldsig-core/ 116

[11] Berthol, O. and Marit Köhntopp, M. "Identity Management Based On P3P http://www.koehntopp.de/marit/publikationen/idmanage/BeKoe_00IdmanageBasedOnP3P.pdf, presented at the Workshop on Design Issues in Anonymity and Unobservability. Available from http://www.w3.org/P3P/ 105

[12] W3C Xpath Specification http://www.w3.org/TR/xpath 112

[13] Composite Capabilities/Preference Profile: W3C standard under development http://www.w3.org/Mobile/CCPP/ 119

Annexe 1: Sample XML signature of P3P policy. Note that a signature for the PRF would be identical except that the node marked with ***'s would refer to a policy reference file.

```
<Signature Id="Signature1" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
         Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000907"/>
    <SignatureMethod
         Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="http://www.example.org/p3p.xml">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2000/
                                             WD-xml-c14n-20000907"/>
      </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
    </Reference>
    <Reference URI="#Assurance1"
        Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties">
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>1342=-0KKAASIC!=123Adxdf</DigestValue>
    </Reference>
<!-- Reference over signature policy *** or policy reference file
                                                         if for prf ***-->
    <Reference URI="http://www.example.org/signaturePolicy.xml">
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>1234x3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
    </Reference>
<!-- Reference over Time Stamp to comply with EU directive -->
    <Reference URI="#TimeStamp1"
        Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties">
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>k3453rvEPO0vKtMup4NbeVu8nk=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>
        <X509Data>
             <X509IssuerSerial>
                <X509IssuerName>CN=Smith John, OU=TRL, O=IBM, L=Example,
                                  ST=Example, C=</X509IssuerName>
                <X509SerialNumber> 12345678 </X509SerialNumber>
             </X509IssuerSerial>
             <X509SKI> 31d97bd7 </X509SKI>
        </X509Data>
        <X509Data><!-- single pointer to certificate-B -->
            <X509SubjectName>Subject of Certificate B</X509SubjectName>
        </X509Data>
        <X509Data> <!-- certificate chain -->
```

```
            <X509Certificate>MIICXTCCA...</X509Certificate>
            <X509Certificate>MIICPzCCA...</X509Certificate>
            <X509Certificate>MIICSTCCA...</X509Certificate>
        </X509Data></KeyInfo>
    <Object>
       <SignatureProperties>
          <SignatureProperty Id="Assurance1" Target="#Signature1"
            xmlns="http://www.w3.org/2000/09/xmldsig#">
          <Assures Policy="<http://www.example.org/p3p.xml>"
            xmlns="http://www.w3.org/2001/02/xmldsig-p3p-profile"/>
          </SignatureProperty>
          <SignatureProperty Id="TimeStamp1" Target="#MySecondSignature">
             <timestamp xmlns="http://www.ietf.org/rfcXXXX.txt">
                <date> 19990908 </date>
                <time> 14:34:34:34 </time>
             </timestamp>
          </SignatureProperty>
       </SignatureProperties></Object></Signature>
```

Annexe 2: APPEL Ruleset for an example user privacy preference consistent with EU data protection directive

```
<?xml version="1.0" encoding="UTF-8"?>
<appel:RULESET xmlns:appel="http://www.w3.org/2001/02/appelv1"
               xmlns:p3p="http://www.w3.org/2000/12/p3pv1">
<appel:RULE behavior="block" description="Any marketing must be opt-in
       with prompt|Data-Type|Any" prompt="yes" promptmsg="Your privacy
       agent has detected a~site which will use your data for
       marketing if you agree to it - do you want to go to this page">
<p3p:POLICY>
<p3p:STATEMENT>
<p3p:PURPOSE appel:connective="or">
  <p3p:telemarketing required="opt-in"/>
  <p3p:contact required="opt-in"/>
</p3p:PURPOSE>
</p3p:STATEMENT>
</p3p:POLICY>
</appel:RULE>
<appel:RULE behavior="block" description="No compulsary marketing"
                                                       prompt="no">
<p3p:POLICY>
<p3p:STATEMENT>
<p3p:PURPOSE appel:connective="or">
<p3p:telemarketing required="always"/>
<p3p:contact required="always"/>
</p3p:PURPOSE></p3p:STATEMENT></p3p:POLICY></appel:RULE>
<appel:RULE behavior="block" description="Blocked because site will use
       your information for marketing purposes on an opt-out basis."
```

```
                                                          prompt="no">
<p3p:POLICY>
<p3p:STATEMENT>
<p3p:PURPOSE appel:connective="or">
<p3p:telemarketing required="opt-out"/>
<p3p:contact required="opt-out"/>
</p3p:PURPOSE></p3p:STATEMENT></p3p:POLICY></appel:RULE>
<appel:RULE behavior="block" description="Blocked because you cannot
       access all your data after submitting it" prompt="no">
<p3p:POLICY>
  <p3p:ACCESS appel:connective="non-and">
    <p3p:all/><p3p:nonident/>
  </p3p:ACCESS>
</p3p:POLICY>
</appel:RULE>
<appel:RULE behavior="block" description="Site will retain information
       collected by this resource beyond what is necessary to carry out
       the stated purpose" prompt="no">
<p3p:POLICY>
<p3p:STATEMENT>
<p3p:RETENTION appel:connective="non-and">
<p3p:stated-purpose/>
</p3p:RETENTION></p3p:STATEMENT></p3p:POLICY></appel:RULE>
<appel:RULE behavior="block" description="Require identity and physical
       address of controller" prompt="no">
  <p3p:POLICY appel:connective="non-or">
    <p3p:ENTITY>
      <p3p:DATA-GROUP>
        <p3p:DATA ref="#business.contact-info.postal.street"/>
        <p3p:DATA ref="#business.contact-info.postal.city"/>
        <p3p:DATA ref="#business.contact-info.postal.stateprov"/>
        <p3p:DATA ref="#business.contact-info.postal.postalcode"/>
        <p3p:DATA ref="#business.contact-info.postal.country"/>
      </p3p:DATA-GROUP>
    </p3p:ENTITY>
  </p3p:POLICY>
</appel:RULE>
<appel:RULE behavior="request" description="Passed all rules and final
       check on disclosure to countries outside EU which don't follow
       the same practices and if they do, Access rights must be
       stated." prompt="no">
<p3p:POLICY>
<p3p:STATEMENT>
    <p3p:RECIPIENT appel:connective="or-exact">
      <p3p:ours/>
      <p3p:delivery/>
      <p3p:same/>
    </p3p:RECIPIENT>
  </p3p:STATEMENT>
</p3p:POLICY>
```

```
</appel:RULE>
<appel:RULE behavior="block" description="Default Rule fired">
<appel:OTHERWISE/></appel:RULE></appel:RULESET>
```

Annexe 3: Example of a Web site privacy policy complying with the principles of the EU data protection directive

```
<?xml version="1.0"?>
<POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
    <EXPIRY max-age="604800"/>

<POLICY
    discuri="http://p3p.jrc.it/modelsite/htmlpolicies/
                                          officialpolicy.html"
    opturi="http://p3pproxy.jrc.it:1080"
    name="mainpolicy">
    <!-- Description of the entity making this policy statement. -->
    <ENTITY>
    <DATA-GROUP>
<DATA ref="#business.name">Joint Research Center</DATA>
<DATA ref="#business.contact-info.postal.street">Cybersecurity TP 361
Via Enrico Fermi 1</DATA>
<DATA ref="#business.contact-info.postal.city">Ispra</DATA>
<DATA ref="#business.contact-info.postal.stateprov">Lombardia</DATA>
<DATA ref="#business.contact-info.postal.postalcode">21020</DATA>
<DATA ref="#business.contact-info.postal.country">Italy</DATA>
<DATA ref="#business.contact-info.postal.organization">
                                             Giles Hogben</DATA>
<DATA ref="#business.contact-info.online.email">
                                             giles.hogben@jrc.it</DATA>
<DATA ref="#business.contact-info.online.uri">http://p3p.jrc.it</DATA>
    </DATA-GROUP>
    </ENTITY>
    <!-- Disclosure -->
    <ACCESS><all/></ACCESS>
    <!-- No dispute information -->
    <!-- Statement for group "General clickstream data" -->
    <STATEMENT>
    <!-- No consequence specified -->
    <!-- Data in this statement is marked as being non-identifiable -->
    <NON-IDENTIFIABLE/>
    <!-- Use (purpose) -->
    <PURPOSE><current/></PURPOSE>
    <!-- Recipients -->
    <RECIPIENT><ours/></RECIPIENT>
    <!-- Retention -->
    <RETENTION><stated-purpose/></RETENTION>
    <!-- Base dataschema elements. -->
    <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    <DATA ref="#dynamic.http"/>
    </DATA-GROUP>
</STATEMENT>
<!-- End of policy -->
</POLICY>
</POLICIES>
```

Authentication for Distributed Web Caches

James Giles, Reiner Sailer, Dinesh Verma, and Suresh Chari

IBM, T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, New York, USA
{gilesjam,sailer,dverma,schari}@watson.ibm.com

Abstract. We consider the problem of offloading secure access-controlled content from central origin servers to distributed caches so clients can access a proximal cache rather than the origin servers. Our security architecture enforces the access-control policies of the origin server without replicating the access-control databases to each of the caches. We describe the security mechanisms to affect such a system and perform an extensive security analysis of our implementation. Our system is an example of how less trustworthy systems can be integrated into a distributed system architecture; it provides mechanisms to preserve the whole distributed system security even in case less trustworthy subsystems are compromised. An application of our system is the cached distribution of access-controlled contents such as subscription-based electronic libraries.

Keywords: Security, Authentication, Cookies, Distributed Applications, CDN.

1 Introduction

It is well known that caching content in a distributed fashion throughout the network can provide substantial benefits such as shorter response time, smaller overall bandwidth requirements, and balanced load among servers. In content distribution networks (CDN), a content provider usually deploys central origin servers that feed multiple content cache servers which are strategically located near client machines as in Figure 1. Clients address their requests to the origin server, but are redirected to one of the content cache servers by a request router in the network. The content cache server requests content from the origin server to satisfy the client requests if the content is not already cached. In the Internet, there is an extensive infrastructure in place (see Akamai [1] and Speedera [2]) to do limited content distribution. However, most of the content caching infrastructure does not provide access control and typically assumes a complete trust relationship between origin servers and the content caches.

This paper addresses the important, practically motivated problem of providing access control for content which we wish to serve from distributed caches. We seek to integrate several proven theoretical solutions into a practical, deployable architecture keeping the following security goals in mind: i) data should be protected until it reaches the client system, ii) from the time that we learn a cache is

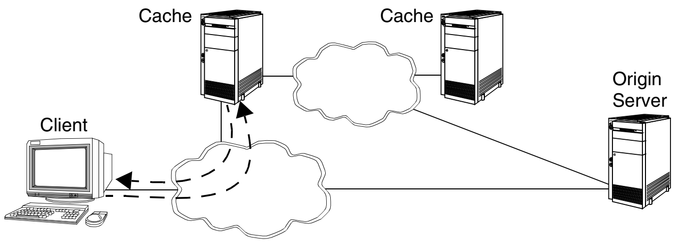


Fig. 1. Content distribution system including origin servers and caches

compromised, the cache should not be able to disclose any additional data that it does not have cached, iii) caches should not be able to masquerade as clients, and iv) the system should be self-healing when caches are known to be compromised. We distinguish trusted caches and known-compromised caches. Trusted caches become known-compromised if we detect security breaches, e.g., through deploying intrusion detection systems (IDS [3], [4]). Since there is a possibility that information stored by caches could be disclosed if a cache is compromised, we do not trust caches with sensitive long-term secrets such as user identifiers and passwords or any other user information that would enable a cache to masquerade as a user. This trust model is motivated from the practical constraint that there are a number of caches residing close to clients, making it expensive to provide administration and physical protection at each location. Our assumptions are that i) the origin server is well protected, ii) the architectures we build on such as SSL and PKI are secure and the implementation is correct, and iii) good intrusion detection systems with low false negative rates are available (infrequent false positives pose less of a problem than false negatives). Our architecture is as strong as the intrusion detection; we benefit from future improvements of IDS. In Section 7, we describe extensions to our architecture that rely less on the IDS systems and protect all information, including information stored in the caches, from disclosure by compromised caches.

An application for this architecture is the USENIX online periodical delivery system [5]. Response time for this system could be improved by distributing content to caches throughout the network using a content distribution service provider, e.g., Akamai [1]. Since most of the content is only available by subscription, the content caches must provide access control mechanisms. However, USENIX is unlikely to trust the content caches with sensitive information such as user passwords, account information, and credit card information.

The mechanisms we have designed to address this problem enable a graceful transition of caches from trusted to distrusted while balancing security against availability of content and access control information revealed to caches. The features of our system include:

– Long-term sensitive content is not revealed unprotected to caches.

– Access control information such as userid and password are not revealed directly to caches. Rather, we use access control tokens with limited validity.
– Our architecture incorporates cryptographic mechanisms to protect against replay attacks with access control tokens.

In general, the weakest subsystem determines the overall security of a system. To maintain a desired level of security for the overall content distribution system, our architecture combines observation technology [4], which monitors the security level of the subsystems, with mechanisms that neutralize subsystems whose security level falls below the desired level. The self-healing mechanisms ensure that known-compromised subsystems cannot be misused to further compromise other subsystems in the future. Because there is a risk that the security sensors will not detect compromised subsystems, we deny the less secure subsystems access to the most sensitive information such as user passwords and long-term protected data. The architecture we have implemented provides a practical solution which balances the need for security with the need for cost effective distribution of content. Our architecture is a compromise between functionality, security, and usability and we explain how these three factors relate to each other in the conclusion.

We perform an extensive security exposure analysis of our architecture and highlight which attacks the mechanisms in our architecture can mitigate. The client setting we have chosen for our solution is the standard web browser and its associated state maintenance mechanisms. We discuss how stronger security guarantees can be obtained by extension of these mechanisms.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the distributed content cache setting in detail. In Section 4, we describe the basic security architecture which emphasizes compatibility with the existing client infrastructure. In Section 5, we describe the implementation of our system. In Section 6, we show the coverage of a generic threat model by our architecture. Section 7 introduces extensions that resolve remaining exposures at the cost of extending client browsers. Finally, in Section 8, we evaluate the architecture and describe the benefits it provides for content distribution systems.

2 Related Work

There are several related authentication schemes that are used for web-based communication for distributed systems. Since extensions to client browsers are usually infeasible, most of the authentication schemes rely on functions normally available to browsers.

Fu [6] describes generic procedures for implementing secure cookie-based client authentication on the web. Our work can be considered an extension that uses the HTTP cookie mechanism [7] to receive and present user credentials to servers within the same domain. Fu's scheme includes a timestamp with the credentials to limit their usefulness and a MAC over the credential fields to protect the credential integrity. The confidentiality of the credentials is assumed since the cookies are sent only over SSL to servers that are assumed to be secure.

Since we use the credentials to access servers in multiple administrative domains (e.g., origin server and caches) and do not have the same level of security at the origin server and caches, we offer protection against cookie replay attacks. Additionally, we provide mechanisms by which cache servers can be excluded from participation in the authentication process if they become known-compromised.

Kerberos [8] makes use of authentication tokens that are granted by an authority to access services on other machines. However, Kerberos depends on distribution and maintenance of secret keys shared by each client and authority pair, which has proven intractable so far in the web environment; our scheme has no such limitations. Inter-realm authentication between different domains is quite expensive using Kerberos. Additionally, Kerberos is currently not supported by common browsers. The Kerberized Credential Translator [9] is an attempt to allow browsers to access Kerberized services over SSL without browser modification, but this scheme also relies on mechanisms for distribution and maintenance of client certificates used to authenticate the client under SSL.

Microsoft's Passport system [10] offers authentication across multiple domains through a centralized authentication server and token-based credentials. Our system differs from Passport in that we use the same token to access multiple caches rather than returning to the origin server to obtain a new token each time a new cache is accessed. Caches in our architecture have freedom to extend the lifetime of the token, but a known-compromised cache can be eliminated from participation in the authentication scheme.

3 Distributed Secure Content Caching System

We propose a self-healing distributed security architecture for content distribution systems, which allows the origin server to maintain control over authentication, while distributing content through partially trusted content caches.

In our architecture the origin server trusts content caches for certain functions, such as distributing content to clients, only after verifying their authorization at the origin site. However, our architecture assumes that distributed caches can be compromised. The failure model we assume is that up to a certain time a cache can be trusted to execute its functions and once compromised it can no longer be trusted to execute its functions. Our architecture also assumes that the compromise of caches can be effectively detected by the use of appropriate security sensor technology (see for example [4]). Thus, our architecture proposes mechanisms by which content caches distribute sensitive information as long as they are trusted by origin servers with the safeguard that a content cache known by origin servers to be compromised can be effectively neutralized from the rest of the system without further risk. The compromised cache is neutralized by disabling its access to the content on the origin server and by distributing new cryptographic material to the remaining caches allowing their continued participation in the system.

The security architecture we propose is a token-based access-control system, which we describe in the context of web servers, browsers, and web caches. There is an origin server, one or more content caches, and one or more clients as in Figure 1. Content distribution systems fall into two broad categories: those in which the client primarily contacts the origin server for content, with the origin server directing the client to retrieve certain objects from a convenient content cache as in [1], and those for which the client is automatically directed to retrieve most content from a convenient content cache without frequently returning to the origin server as in [2]. We focus on content caching systems of the second type, although the architecture works equally well with both content cache types. In addition, the architecture that we propose is limited to HTTP [11, 12] traffic, but our approach can be generalized to other protocols and distributed applications.

We implemented our security architecture using standard security techniques like Secure Sockets Layer (SSL) [13, 14], IPSEC [15], Password Authentication Protocol, public key signatures, symmetric encryption, and standard web technology like browsers and Apache web servers including SSL and servlet modules. The state maintenance mechanism we use is the HTTP cookie mechanism [7] which we employ to convey authentication tokens. We take safeguards against the well-known exposures in HTTP cookies, such as cookie theft. The significant contributions of our architecture are that it:

– protects against a large number of attacks arising from the distributed cache and server architecture.
– provides single sign-on to all the content caches for a particular origin site.
– maintains user privacy.
– enables self-healing and containment when a content cache is determined to be known-compromised.

By self-healing and containment, we mean that an origin server can disable a compromised content cache in such a way that it cannot be used to further jeopardize the security, even if it is completely controlled by an adversary.

We perform an extensive security analysis of our architecture by enumerating exhaustively the different security exposures of our system. We discuss in detail those exposures which are unique to our architecture as opposed to those common to any client-server application and highlight the features of our architecture that mitigate each of these exposures.

4 Security Architecture and Mechanisms

Our security architecture imposes two primary requirements. First, the origin servers' and clients' long-term secrets and integrity-sensitive contents are not made accessible to caches without protecting the content. Second, compromised caches must be excluded seamlessly from further participation in the content distribution network (self-healing).

4.1 Authentication and Access Control

Figure 2 illustrates the information and control flows for the basic architecture. In its simplest form, clients initially authenticate with a trusted origin server

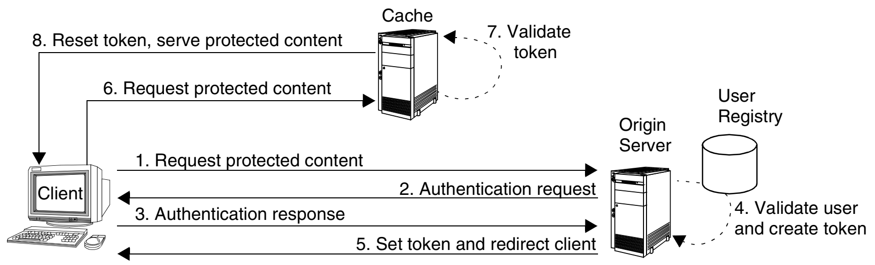

Cache
7. Validate token
8. Reset token, serve protected content
6. Request protected content
User Registry
Origin Server
Client
1. Request protected content
2. Authentication request
3. Authentication response
5. Set token and redirect client
4. Validate user and create token

Fig. 2. Requesting protected data in our architecture

using userid and password (cf. 1-4 in Figure 2) and receive an authentication token from the origin server (cf. 5 in Figure 2) containing information about the content that the client should be entitled to retrieve. The origin server redirects the client to the *content caches* with the specific content cache being resolved from the redirection URL with DNS-based request routing ([16, 17]). The client (or browser) presents this authentication token to the cache along with a request for content (cf. 6 in Figure 2). After receiving an initial request and a token, the cache then validates the token and grants access to the client if the access control information in the token indicates that the client is authorized to view the requested content (cf. 7-8 in Figure 2). After successful validation of a token, the cache stores the SSL session identifier together with the authentication token in a *session identifier cache* so that subsequent requests within the same SSL session and within a short time period Δt do not require the server to evaluate the token again. Assuming that user and client do not change within a SSL session, we do not lose security by relying on the SSL session identifier for authentication rather than evaluating the token again. A small Δt ensures that a token still expires near to the timeouts, i.e., that a token is evaluated at least every Δt seconds. At the end of an SSL session, the cache entry of this SSL identifier and the related token are deleted on the server. When the client logs-off, the cache or the origin server deletes the cookie in the client's browser.

Communication of authentication credentials, tokens, and sensitive information between origin servers and clients, and caches and clients is protected by SSL. Content and keys are exchanged by the origin servers and caches via mutually authenticated SSL or IPSEC connections. For cases when the origin servers and the caches are part of the same name-space domain, the authentication token is passed from the origin servers to the clients and from the clients to the caches through the use of HTTP cookies [7]. More elaborate forms of the system, discussed in Section 7, incorporate techniques for cross-domain cookies in cases where the cache and origin server are in different domains, as well as techniques for protecting highly sensitive content from disclosure if a cache is compromised.

4.2 Authentication Token

The authentication token implements a one-way authentication whereby the client sends a token to the cache and the cache identifies the client through the content of the token. The authentication is usually one of possession, i.e., a client possessing the token is assumed to be authenticated for the identity or capabilities claimed in the token. Therefore, unauthorized replay of authentication tokens – especially when implemented as cookies – is a major threat. To thwart unauthorized replay, we include information in the token which enables the cache to validate the sender of the token. Such information should be:

- unique to sender (sender IP address hash, Browser Process and User IDs)
- easily verifiable by cache (apparent client IP address, HTTP header)
- difficult to guess for attackers
- flexible with client privacy and security

First, we describe the structure of the cookie that carries the authentication token between clients and servers. Afterwards, we describe the authentication token in detail. Finally, we discuss the protection mechanisms for the cookie and the authentication token.

Cookie Structure. Our architecture uses cookies to transport authentication tokens (AUTH_TOKEN) from origin servers to client browsers and between client browsers and caches. Below, we show an exemplary cookie that includes an authentication token; it will be sent over SSL only to servers in the .ibm.com domain (see “secure” and “domain” fields [7]). The expires-field of the cookie is not used, i.e., the cookie is deleted when the browser closes.

```
Cookie: authtoken=AUTH_TOKEN; path=/; domain=.ibm.com; secure;
```

The cookie as illustrated above ensures the following properties:

- Cookies containing authentication tokens are sent only over SSL connections: First, cookies are sent only to sites that authenticated via SSL with a certificate issued for a machine in the stated domain. Second, servers will receive cookies and included authentication tokens only if the request comes over SSL; non-SSL traffic will not impose cookie handling on the server.
- The cookie and included authentication token will be destroyed once the web browser is closed.
- The cookie transports the authentication token transparently (the authentication token is relatively small).

Any information used for access control is stored securely in the authentication token. We do not protect the integrity or confidentiality of the *cookie* because any sensitive information is protected inside the authentication token. The contents of the authentication token are illustrated in Table 1.

The *origin domain ID* contains the domain name of the origin server. A *generation timestamp* indicates the time the token is generated in seconds since Jan 1, 1970. *Inactivity and global timeouts* are offsets from this generation timestamp

Table 1. Authentication token contents: signed (S), hashed (H), and encrypted (E)

Origin domain id	S, H, E
Generation timestamp	S, H, E
Global timeout	S, H, E
Access control information	S, H, E
Client environment hash: client IP address and HTTP header fields	S, H, E
Origin server signature	H, E
Inactivity timeout	H, E
Token hash MD5/SHA-1	E

and invalidate the cookie if either expires. The global timeout is set by the origin server and cannot be changed (e.g., 2 hours). The global timeout sets an upper limit on the validity period of authentication tokens in seconds starting from the generation time. The inactivity timeout (e.g., 180 seconds) invalidates the authentication token if there is no client request within this time period. It is advanced by servers when handling client requests after the cookie is validated.

Subscription service and access rights are stored in the *access control information* field (e.g., user/group ID). Additional information about the client such as the apparent IP address of the client or header fields is hashed and included in the *Client environment hash* field to further authenticate the sender of a cookie. The *Origin server signature* is built by signing all authentication token fields except the *Token hash*, *Inactivity timeout*, and *Origin server signature* fields using an asymmetric algorithm (e.g., 1024bit-RSA). The *Token hash* field includes a one-way hash (e.g., MD5 [18]) over the whole authentication token excluding the *Token hash* field. Finally, the authentication token is encrypted using a symmetric encryption algorithm (e.g., 2key TripleDES-CBC).

Authentication Token Tamper and Disclosure Protection. The authentication token includes three levels of protection. A first level (static integrity protection) is achieved by signing the static part of the cookie with the secret key of the origin server. This signature, stored in the *Origin server signature* field, protects against unnoticed tampering of the signed static authentication token fields. Only the origin server can create an authentication token. The second level (privacy protection) consists of encrypting the whole authentication token with a symmetric *token encryption key* shared only by the origin server and the trusted cache servers. A third level (dynamic integrity protection) is added by the encrypted hash value computed over the authentication token and stored in the *Token hash* field; this level is needed because the cache servers need to adjust the dynamic *Inactivity timeout* field. After updating the authentication token, the cache servers recompute the hash and encrypt the whole token with the known symmetric encryption key. We compute the token hash value before encrypting the authentication token and use the symmetric key both for privacy protection and for implementing the hash signature. An attacker trying to change the inactivity timeout (blindly because encrypted) would have to adapt the hash value in order to obtain a valid cookie; encrypting the hash thwarts this attack. Changing any other field would both invalidate the token hash and the origin server signature.

Considering all three levels, the authentication token is protected against unauthorized disclosure to any party other than the origin and caches servers. The dynamic fields (inactivity timeout and hash) are protected against unauthorized change by any party other than the origin and trusted cache servers. The other fields (static fields) are protected against unauthorized change by any party other than the origin server. No third party can disclose the content from the authentication token or change its contents unnoticed by other means than breaking the cryptography (brute force attacks on the keys or the algorithms).

Re-keying. We need to change the token encryption keys from time to time:

– to protect against successful known plain-text or brute force attacks on the token encryption key (normal re-keying mode)
– to exclude known-compromised caches from updating or re-using tokens (compromised re-keying mode)

Re-keying occurs about once every few hours but depends on the amount of token data encrypted under the same encryption key. We consider caches well-protected and assume that known compromises occur infrequently and are handled in an exception mode described in Section 4.3. Therefore we implemented re-keying by distributing the new key from the origin server to all trusted caches via SSL. If re-keying occurs often or the number of caches is very large, then we propose to apply fast group re-keying schemes as known from multicast architectures [19]. The public key certificates of the origin server, used by caches to verify the origin server signature, are supposed to be valid for a very long time (e.g., half a year). If the origin server private key changes, re-distributing certificates to caches can be done manually.

Authentication Token Replay Protection. As with other capability based systems, we usually want to verify the sender of valid credentials [20], [21]. One solution is to assume attackers cannot gain access to valid authentication tokens. Obviously, following the many security problems in todays web browsers, this would not be a valid assumption in our application environment - even considering non-persistent, secure cookies.

If the client-side public key infrastructure were feasible, demanding client side SSL certificate authentication would be an obvious choice; the cookies that are received via an SSL session would be authenticated by the contents of the client certificate. Besides client certificates being not very popular and PKI being not broadly trusted, this approach presumes a working revocation mechanism for client certificates. As the respective secret signature keys would reside on the client machine, exposure will happen regularly and revocation lists would become quite long and difficult to manage, so we do not take this approach.

Therefore, our authentication tokens – generated by the origin server – have several "built-in" protections against replay attacks. The timeouts restrict the replay window for clients to the inactivity timeout. The global timeout restricts replay of tokens after a client is authenticated (e.g., at latest after a global timeout, changes in the user registry – such as changing group membership or deleting users – will go into effect). Additionally, the client environment hash over the client IP address and HTTP header fields restricts replay of tokens to clients that share the same IP address (or Proxy) and the same HTTP header fields. If the client environment changes, as observed by the caches or origin server, then the client environment hash computed by the servers will change, too. Hence the cookie will be invalid when submitted from the new environment because the computed client environment hash differs from the *Client environment hash* stored in the token. Note, that dynamic IP addresses obtained over the DHCP protocol will usually not adversely effect the *Client environment hash* because the leases are most often quite long compared to typical secure web sessions and in any case clients are often re-assigned the same IP address when leases expire. For token replay by known-compromised caches see Section 4.3.

More elaborate techniques using difficult to guess environment information are presented in Section 7 as they presume enhancements in the client environment (browser extensions or applets). In environments, where the IP address changes often or multiple proxy servers are used, we propose to use the extended version described in Section 7.

Authentication Token Creation and Distribution Protection. The user is authenticated against the origin before before an authentication token is created. A variety of authentication methods, such as userid/password or one-time passwords, can be used over SSL to authenticate the user. Before submitting authentication information, the user should carefully verify the certificate obtained during the SSL server authentication to thwart server masquerading attacks. A weakness of todays web service infrastructure is that major web browsers do not automatically show information about the authenticated server to the user if the certificate is valid.

The origin server transfers the created authentication token over SSL directly to the client using HTTP cookies, or one of the other techniques described in Section 7. Clients present their tokens to caches with each information request over SSL. SSL helps in this case to protect cookies from disclosure to attackers and subsequent replay attacks.

Authentication Token Validation. Servers validate authentication tokens received in cookies in a multi-stage process as shown in Figure 3. Recall, that the cache does not need to validate every token, see Section 4.1. The authentication token is validated successfully only if it passes several checks. First, we discard authentication tokens whose length is not a multiple of the block size of our encryption algorithm (usually 8-byte aligned). Second, we discard authentication tokens that were stored previously in an *invalid authentication token cache.* Third, the authentication token is decrypted, and the decrypted token is hashed and matched against the decrypted *Token hash* field; this verifies both that the authentication token was encrypted using the valid shared key and that it was not tampered with. Next, the decrypted timestamps are checked to see if the authentication token has timed-out. Then, the client environment hash in the

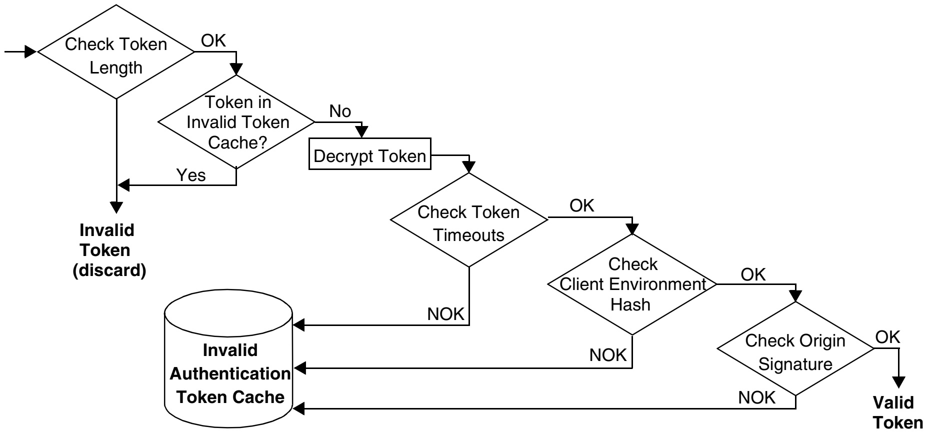


Fig. 3. Validating authentication tokens

cookie is verified by checking sender information (e.g., IP address) as observed by the server. Finally, the origin server signature is checked. The validation process stops as soon as a check fails and the request is denied. A rejected authentication token is added to a cache of invalid authentication tokens and the client is redirected to the origin server for re-authentication. We classify authentication tokens for which the client environment hash check fails as invalid to prevent attackers from simulating different client environments against the same cache.

Authentication Token Update. Inactivity timeouts must be updated regularly to ensure that tokens do not expire throughout their use. Cache servers can update tokens whenever a request is granted after validating a token. When updating a token throughout validation, the cache server must calculate the new inactivity timeout (current local time - generation time + inactivity offset), recalculate the token hash, and encrypt the new authentication token. Throughout bursts, the server would have a significant overhead by updating tokens for the same client in a single SSL session.

For server performance reasons, tokens are validated and their inactivity timeout is advanced not for each client request but rather only if a time Δt has gone by since the last update of the inactivity timeout or if the SSL session identifier changes. The time stamp of the last update is stored together with the token in the session identifier cache. Hence, a token must be re-validated if (local time - last update $\geq \Delta t$). The value of Δt must be relatively small in relation to the inactivity timeout, e.g., 5 seconds. This ensures that a server does not need to update an authentication token for each request within a burst. Depending on a valid SSL session identifier, this mechanism also ensures that the first authentication token received via a SSL session is validated.

4.3 Self-Healing in Case of Compromised Caches

Neutralizing a known-compromised cache includes (i) excluding the cache from accessing and updating authentication tokens created or updated in the future, (ii) excluding the cache from accessing the origin server or other CDN infrastructure, and (iii) ensuring that clients do not connect to distrusted cache servers.

To exclude a known-compromised cache from accessing and updating authentication tokens, a new shared cookie encryption key is distributed by the origin server to the remaining trusted caches via SSL. In doing so, the compromised cache cannot disclose or update any new authentication tokens and old tokens are no longer accepted by the other caches. To ensure immediate impact, any cache that receives a signed re-keying notification with the compromised-flag set, will not accept any tokens encrypted by the old token encryption key and will use only the new token encryption key included in the notification. Clients submitting such old tokens are redirected to the origin server for re-authentication and for creating a new token. As these cases should be rare, it seems acceptable to re-authenticate users.

To exclude a known-compromised cache from further participation in the CDN infrastructure, the origin server disconnects the SSL connection to the cache so that the cache no longer has any access to the content. The cache's certificate is invalidated so that the cache cannot set-up new SSL connections to the CDN infrastructure. To prevent clients from accessing compromised caches, the CDN request router will no longer redirect clients to known-compromised caches. If the CDN router uses DNS-based redirection [17, 16], we can ensure that clients are directed to remaining trusted caches even if they bookmarked pages from the caches – as long as the bookmarks contain server names and not IP addresses. The DNS resolves machine names to trusted caches only.

If a cache server is compromised, any content at the cache will be disclosed to the attacker. The basic system does not prevent such disclosure, but rather does not replicate long-term sensitive content to caches (e.g. replicate articles, but no passwords). An extension of this system stores only protected content on caches (encrypted and signed content with encryption and signature keys unknown to the cache); if such a cache is compromised, the content cannot be disclosed or unnoticeably changed. Such an extension needs a client browser enhancement (e.g., plug-in) that handles protected (encrypted and integrity protected) content.

5 Implementation

For our prototype implementation, we used standard client systems running Microsoft Windows or Redhat Linux operating systems with either Internet Explorer or Netscape Navigator browsers. The caches and origin servers were PC systems running Redhat Linux and the Apache [22] web server with the Tomcat servlet extensions. The initial authentication at the origin server was implemented with Java servlets and the authentication cookie was set directly in the client browser or stored in an OpenLDAP [23] directory which was accessible to

Table 2. Cryptographic performance

1024bit RSA Signature	9300 us
1024bit RSA Verification	520 us
2-key TDES CBC	5.29 MByte/s
MD5	93.5 MByte/s
SHA-1	47.51 MByte/s

the caches as part of the cross-domain cookie extension discussed in Section 7. The access control at caches was implemented with a Java servlet, and servlets were used to manage the updates for the keys shared between the origin server and the trusted caches. We used Apache SSL to protect communication between the client and both the origin and cache servers. We also used Apache SSL for all communication between the caches, the origin servers, and the LDAP directory, e.g., protecting the distribution of token encryption keys. For performance reasons, we plan to replace the servlets with Apache module extensions.

Performance. Our authentication token field lengths are as follows: Origin domain id (256 bytes), Generation timestamp (4 bytes), Global timeout (4 bytes), Access control information (userid 4 bytes, groupid 4 bytes), client environment MD5 hash (16 bytes), Origin server signature (128 bytes), Inactivity timeout (4 bytes), MD5 Token hash (16 bytes). The total length of our authentication token is 436 bytes, the cookie length will therefore around 500 bytes. We measured the following performance with OpenSSL on a Pentium III, 700 MHz, compiled with MSVC 5.0 as shown in Table 2. To create authentication tokens, origin servers need a signature (9.3ms), a 420 byte MD5 hash (4.5us), and a 436 byte 2-key TripleDES CBC encryption (82us) to create a token (total: 9.386 ms). To validate a token, a cache needs to decrypt the authentication token (82us) and to compute the token hash (4.5 us), hence 86.5 us in total. Updating a token means to reset the inactivity timeout, recalculate the token hash and encrypt the token (total 86.5 us), whereby a token is validated and updated at most every Δt seconds per client within an SSL session. For more information about proper key sizes, see [24].

6 Security Analysis

For an attack model, content outsourcing is partitioned into three components which are to be secured: static data objects, which we refer to as $\mathcal{D}$, authentication functions, which we refer to as $\mathcal{A}$, and other functions such as programs that build pages dynamically, which we refer to as $\mathcal{F}$. Parties involved in the content cache system are clients (U), caches (C), origin servers (O), and the network (N). Table 3 summarizes the features addressed in this paper.

Our security architecture addresses features of authentication ($\mathcal{A}$) at the client, data ($\mathcal{D}$), and other functions ($\mathcal{F}$) at the cache, and $\mathcal{A}$ at the origin. The

Table 3. Addressed (x), no control (†), assumed (‡), SSL (△)

	Components		
Players	$\mathcal{D}$ata	$\mathcal{A}$uth	$\mathcal{F}$unctions
User	†	x	†
Cache	x	x	x
Origin	‡	x	‡
Network	△		

architecture makes use of SSL for all communications, allowing our protocol to be independent of $\mathcal{F}$ (other than availability) and $\mathcal{A}$ and $\mathcal{D}$ on the network. We do not address $\mathcal{D}$ and $\mathcal{F}$ on the client and $\mathcal{D}$ and $\mathcal{F}$ on the origin server. On the cache, we address security breaches insofar as we assume to recognize when a cache is compromised. System security, addressing $\mathcal{F}$ and $\mathcal{D}$, can be implemented independently and complements our architecture. Availability attacks are not explicitly addressed, although there are aspects of our architecture that protect against availability attacks. The threat tree [25] in Figure 4 models possible threats to the security of our scheme. The abbreviations D, I, and $D.O.S.$ in the figure correspond to disclosure attacks, integrity attacks, and denial of service attacks, respectively. Our system is primarily concerned with protecting the authentication functions and with neutralizing caches that are known to be compromised. We will not discuss attacks such as network disclosure and integrity, because our architecture uses established techniques, such as SSL or IPSEC, to protect against them. Protections against denial of service attacks – other than token flooding – are not addressed by our scheme, but can be implemented independently. We rely on clients to carefully protect their authentication credentials such as passwords, and to cautiously verify SSL certificates to ensure that they are communicating with the correct entities. In the remainder of this section, we describe the attacks that our system specifically addresses.

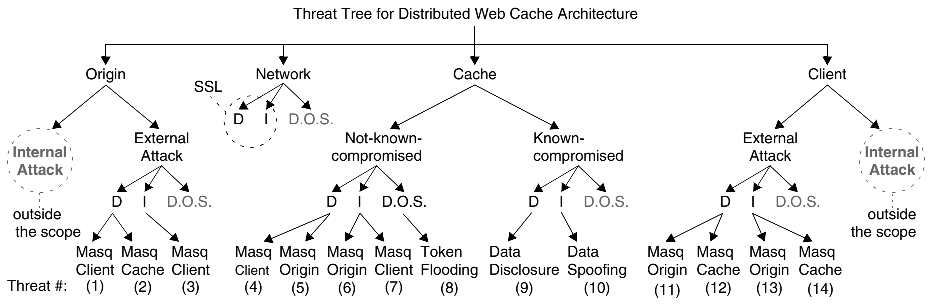


Fig. 4. Threat tree: disclosure (D), integrity (I), and denial of service ($D.O.S.$)

Since we assume clients and origin servers are secured against internal attacks, we focus on external attacks against these systems. In particular, the origin server is expected to be highly secured with intrusion detection and virus protection software, firewall protection, and log monitoring. However, we explore the case that a cache is known to be compromised and must be excluded from the architecture and consider what damage such a cache compromise can cause to overall system security in the future.

6.1 Masquerading Clients

Threat. Origin or cache servers may disclose sensitive content to a masquerading client. Additionally, masquerading clients may manipulate the data at an origin server or cache violating the data integrity. This threat occurs when a masquerading client has restricted access to put data into the origin server or cache (service profiles, passwords management, order history). See attacks 1, 3, 4, and 7 in Figure 4.

Attacks. Certain exploits of this threat, such as an attacker *obtaining or guessing and using identities and passwords* of a careless user, are common to all authentication schemes and are not considered here. We focus on attacks where an attacker tries to masquerade as a legitimate client by *submitting a valid authentication token* to the origin server or cache. If the origin server is tricked into accepting an authentication token, then the attacker can access content and possibly be granted future access to the system.

Implementing Attacks. An attacker can try to masquerade either against the origin server or against the cache – the result is the same. To masquerade, the attacker may attempt to obtain a valid authentication token by:

– *generating a valid authentication token from scratch.*
– *modifying an existing token* to gain or extend access rights.
– *stealing a valid authentication token* from a client, cache, or origin server.

Once the attacker has obtained a valid authentication token, he can try to masquerade as a client by replaying the stolen token.

Defenses. The encrypted token hash acts as a signature on the token contents. To create or modify a valid token, the attacker needs to create a new token hash in encrypted form. Only knowing the current token encryption key could this be accomplished. Hence, besides it being difficult to steal a valid authentication token, to successfully replay such a token, the attacker must stay within the validity period of the token and satisfy the server's client environment check. For this, the attacker must guess and simulate a legitimate client's environment, see Sections 4 and 7.4.

6.2 Masquerading Caches

Threat. The origin server might disclose information to a masquerading cache. This threat is especially dangerous because a cache has the authority to obtain content from the origin server on behalf of any users. A successfully masquerading cache could simply request any desired content for which it is authorized from the origin server.

Defenses. The caches and the origin server use mutually authenticated SSL to protect the communication stream and to protect against a masquerading cache by verifying SSL certificates. This ensures the identity of the cache. Each cache is trusted as long the the IDS connected to the cache does not report otherwise. This protection is as secure as SSL and the public key infrastructure, and as good as the IDS deployed.

Threat. Clients might disclose sensitive information to a masquerading cache such as user passwords or authentication tokens. These passwords could be used to change user profiles and even deny access to legitimate users, e.g. by changing access passwords. Authentication tokens could be used to launch other masquerading client attacks. Masquerading caches may also serve manipulated information to clients. See attacks 2, 12, and 14 in Figure 4.

Implementing Attacks. To masquerade as a trusted cache, an attacker can either compromise and take control of an existing cache or trick the victims into believing that another machine is a legitimate cache. To trick the victim, the attacker would need to route clients to a masquerading cache (e.g., by DNS spoofing).

Defenses. Caches are observed by intrusion detection systems. Once a cache is taken over, the IDS will report the cache as compromised to the origin server. The origin server will initiate a re-keying in compromise-mode, hence any old tokens will be invalid. This protection is as good as the IDS deployed. An attacker masquerading as a cache must authenticate throughout the SSL session setup against the client. As the cache does not have the private key of any trusted cache, the client will not send the cookie including the authentication token to such an attacker. For further protection against masquerading caches supplying forged information to clients or against disclosure of cache contents in case of cache compromise see Section 7.

6.3 Masquerading Origin Servers

An attacker masquerading as an origin server could attempt to capture information from clients and caches, and serve manipulated information to clients and caches. Since we deploy SSL authentication between caches or clients and the origin server, the caches or clients can use the contents of the origin server certificate after authentication to verify the identity of the origin server. See attacks 5, 6, 11, and 13 in Figure 4.

6.4 D.O.S. Attacks Against Caches

All publicly accessible server systems are subject to denial of service attacks, e.g. by flooding. The multi-stage validation of authentication tokens identifies invalid tokens as early as possible to prevent more expensive authorization operations such as decryption and signature verification. Furthermore, denial of service attacks from authenticated users do not result in authentication token validation overhead since SSL session identifiers are used for validating the identification of authenticated users rather than authentication tokens. Additional denial of service attack prevention should be provided by the content distribution network infrastructure. See attack 8 in Figure 4.

6.5 Compromised Caches

Threat. A compromised cache could disclose data including content, cryptographic keys, and authentication tokens. Until the cache is known-compromised, it can also retrieve content from the origin server and send manipulated data to the origin server. See attacks 9 and 10 in Figure 4.

Implementing Attacks. Remote attacks include exploiting known vulnerabilities of the services running on the caches. Local attacks can occur since caches are not located within the physical protections of the origin domain.

Defenses. We assume that each cache is monitored by the origin server with intrusion detection systems (e.g., by IDS running in physically secured coprocessors located at the cache [4]). When a compromise is detected, the origin server neutralizes the compromised cache as described in Section 4. To protect long-term secrets and integrity of sensitive data, we propose in Section 7 extensions that achieve end-to-end security between origin servers and clients and that allows clients to verify the validity of cache certificates more thoroughly.

7 Extensions to the Architecture

We have implemented our base architecture, making use of servlets on the origin servers and caches to manage the authentication and update authentication tokens stored in cookies. Our analysis of the system allowed us to develop several enhancements that help improve protection. In this section we describe several extensions to our base system.

7.1 Cross Domain Cookies

The origin server sets an authentication token stored in a cookie on the client's browser using the HTTP protocol, and then the client's browser presents the cookie to the cache using the HTTP protocol. This technique only works if the cache and the origin server are in the same domain (see [12]).

It is desirable to have caches in a different domain than the origin server because this allows users to differentiate connections to the origin servers from those to caches, helping to protect against attacks 12 and 14. One technique to enable cross domain cookies is for the origin server to store a cookie in a shared directory (we use an LDAP directory) that is accessible to the caches, in addition to setting the cookie on the client. Then, when the origin server redirects a client to use a particular cache in a different domain, the origin server includes additional fields in the redirection URL that indicate the record number of the cookie in the shared directory. The record number is a difficult to guess, large random number because an attacker guessing this number within the inactivity timeout period is equivalent to an attacker possessing a valid authentication token (see Section 4.2 for replay protection). The cache extracts the cookie identifier, fetches the cookie from the directory, and then sets the updated cookie (with new inactivity timeout) on the client so that the directory need not be consulted on future visits. In this way, authentication tokens stored in cookies can be shared among domains and the caches do not need write access to the directory. The cookies in the directory lose their validity as soon as the inactivity timeout expires and should be removed from the directory periodically.

7.2 Secure Content

The basic architecture may not be appropriate for more sensitive data as used in e-commerce with the requirement for confidentiality or stock quotes with the need for integrity since the clients cannot detect when a cache has been compromised. To serve highly sensitive content, the origin server only delivers encrypted content to the cache and provides the client with a means to decrypt this content when the client authenticates with the origin server. This extension protects contents in case of cache compromise. However, it requires extra software on the client in the form of an applet or browser plug-in for encrypting and decrypting or checking and protecting the integrity of content.

7.3 Cache Verification

Any clients interacting with a cache when it is compromised will probably not be aware of this until the global lifetime of the authentication token expires. To raise security for clients that are SSL connected to a cache that becomes compromised, an applet or browser plug-in could be used to validate the cache certificate at the origin server before sensitive operations. The extra security offered by this extension needs to be balanced against the communication overhead.

7.4 Stronger Client Identification

To enhance the protection against authentication token replay attacks, we propose either a client-side program that computes a hash of additional client-specific information or a browser enhancement that computes a unique identification that is recomputed each time the browser is started. A client environment hash including this additional information is both included in the authentication token and available on request to the caches for validating the sender of an authentication token. Both approaches increase the cost of an attacker to masquerade as a legitimate client. The attacker must simulate or guess these parameters to successfully replay an authentication token.

8 Conclusion

We have described a self-healing, distributed security architecture, which allows origin servers to maintain control over authentication, while distributing content through partially trusted content caches. The strongest feature of this architecture is the self-healing feature that allows corrupted caches to be disqualified from the content distribution network simply by changing an encryption key, with no future harm to the security of the system. In addition, the fact that the user authentication credentials are never seen by the less secure caches reduces the possibility that the user will be inconvenienced beyond re-authentication when a cache is known to be compromised. There are a number of tradeoffs of the various aspects of our architecture such as functionality, security, and usability:

– *Less functionality* on caches makes them less vulnerable to attacks because the system is able to keep a smaller amount of sensitive information on the caches (which must be assumed to become compromised).
– More client specific information kept in the authentication token implies *less privacy* and protection against caches. For example, the client environment hash requires the cache to obtain user specific information to validate that hash, but protects against replay attacks. Authorization information allows finer-grained access control, but reveals more information to the caches.
– The shorter the time-outs, the smaller the replay-windows but the more often users need to authenticate interactively; hence *less usability* and less benefit from caching.

The presented architecture was prototyped and proved easy to implement. The overhead for using the architecture is relatively small, especially if clients already communicate with the origin server before being directed to the content caches. The cookies, which are transmitted on each request, are limited in size (depending on the signature system used in the authentication token). Since the fields of the authentication token are just encrypted with a shared key mechanism, it is relatively easy for the content caches to decode and encode the authentication tokens stored in the cookie when necessary. Additionally, the signature by the origin server is done only once. No state needs to be kept by the content caches or the origin server, although state can be kept to improve performance.

Content distribution networks have proved to be viable for distributing contents to improve the user's experience. However, businesses are now moving from distributing public content to distributing subscription-based services to increase their profitability. The shared caching infrastructure and protected contents imply the need for a higher security level which can be offered by our architecture.

References

[1] Akamai Technologies, Inc. Freeflow content distribution service. `http://www.akamai.com`. 126, 127, 130

[2] Speedera. SpeedCharge for Site Delivery. `http://www.speedera.com`. 126, 130

[3] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection, 1998. `http://secinf.net/info/ids/idspaper/idspaper.html`. 127

[4] J. Dyer, R. Perez, R. Sailer, and L. van Doorn. Personal firewalls and intrusion detection systems. In *2nd Australian Information Warfare & Security Conference (IWAR)*, November 2001. 127, 128, 129, 142

[5] USENIX. USENIX online library and index. `http://www.usenix.org/publications/library/index.html`. 127

[6] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *The 10th USENIX Security Symposium*. USENIX, August 2001. 128

[7] D. Kristol and L. Montulli. HTTP state management mechanism, February 1997. Request for Comment 2109, Network Working Group. 128, 130, 131, 132

[8] J. Kohl and C. Neuman. Kerberos network authentication service (V5), September 1993. Request for Comment 1510, Network Working Group. 129

[9] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to web access control. In *The 10th USENIX Security Symposium*. USENIX, August 2001. 129

[10] Microsoft Corporation. .NET Passport 2.0 Technical Overview. `http://www.microsoft.com/myservices/passport/technical.doc`, October 2001. 129

[11] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0, May 1996. Request for Comment 1945, Network Working Group. 130

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol –HTTP/1.1, June 1999. Request for Comment 2616, Network Working Group. 130, 142

[13] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0, November 1996. `http://home.netscape.com/eng/ssl3/draft302.txt`. 130

[14] T. Dierks and C. Allen. The TLS protocol version 1.0, January 1999. Request for Comment 2246, Network Working Group. 130

[15] S. Kent and R. Atkinson. Security architecture for the internet protocol, November 1998. Request for Comment 2401, Network Working Group. 130

[16] A. Gulbrandsen, T. Technologies, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV), February 2000. Request for Comment 2782, Network Working Group. 131, 137

[17] T. Brisco. DNS support for load balancing, April 1995. Request for Comment 1794, Network Working Group. 131, 137

[18] R. Rivest. The MD5 message-digest algorithm, April 1992. Request for Comment 1321, Network Working Group. 133

[19] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures, June 1999. Request for Comment 2627, Network Working Group. 134

[20] J. Saltzer. Protection and the Control of Information Sharing in MULTICS. *Communications of the ACM*, 17:388–402, 1974. 134

[21] A. Tanenbaum, S. Mullender, and R. Renesse. Using sparse capabilities in a distributed operating system. In *The 6th IEEE Conference on Distributed Computing Systems*. IEEE, June 1986. 134

[22] Apache project. `http://www.apache.org`. 137

[23] OpenLDAP project. `http://www.openldap.org`. 137
[24] E. R. Verheul A. K. Lenstra. Selecting cryptographic key sizes. `http://www.cryptosavvy.com/Joc.pdf`. 138
[25] E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994. 139

Analysing a Stream Authentication Protocol Using Model Checking

Philippa Broadfoot and Gavin Lowe[*]

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
{philippa.broadfoot,gavin.lowe}@comlab.ox.ac.uk

Abstract. In this paper, we consider how one can analyse a stream authentication protocol using model checking techniques. In particular, we focus on the Timed Efficient Stream Loss-tolerant Authentication Protocol, TESLA. This protocol differs from the standard class of authentication protocols previously analysed using model checking techniques in the following interesting way: an unbounded stream of messages is broadcast by a sender, making use of an unbounded stream of keys; the authentication of the n-th message in the stream is achieved on receipt of the $n+1$-th message. We show that, despite the infinite nature of the protocol, it is possible to build a finite model that correctly captures its behaviour.

1 Introduction

In this paper, we consider how one can capture and analyse a stream authentication protocol using model checking techniques. In particular, we focus on the Timed Efficient Stream Loss-tolerant Authentication Protocol, TESLA, developed by Perrig *et al.* [11]. This protocol is designed to enable authentication of a continuous stream of packets, broadcast over an unreliable medium to a group of receivers; an example application would be to provide authenticated streamed audio or video over the Internet.

This protocol differs from the standard class of authentication protocols previously analysed using model checking techniques in the following interesting way: a continuous stream of messages is broadcast by a sender; the authentication of the n-th message in the stream is achieved on the receipt of the $n+1$-th message. Thus, receivers use information in later packets to authenticate earlier packets. We give a complete description of the protocol below. A particular challenge when modelling and analysing this protocol is that it uses an unbounded stream of cryptographic keys.

In [2], Myla Archer analyses TESLA using the theorem prover TAME. She states:

[*] This work was partially funded by the UK Engineering and Physical Sciences Research Council and Particle Physics and Astronomy Research Council as part of the e-science program.

> Model checking this kind of protocol is not feasible because an infinite state system is required to represent the inductive relationship between an arbitrary n-th packet and the initial packet.

We took this claim as a challenge, and the current paper is the result. In particular, we construct a finite CSP model [5, 12], which can be analysed using the model checker FDR [4], that correctly captures the unbounded stream of keys. To simulate the unbounded stream of keys in the system, we make use of the data independence techniques developed by Roscoe [13].

The rest of this paper is organised as follows. In Section 2 we describe the TESLA protocol in more detail. In particular, we describe the first two schemes (out of five) presented in [11]. In Section 3 we present a natural but infinite state model of the basic protocol (Scheme I) within our CSP/FDR framework, so as to introduce many of the basic techniques. We reduce this system to an equivalent finite version in Section 4 by applying the data independence techniques to the generation of keys; we also present a number of techniques for reducing the size of the model's state space, so as to make the analysis more efficient. In Section 5 we describe how we adapt our model to handle the second version of the protocol (Scheme II), where each key is formed as the hash of the following key. We conclude and discuss future work in Section 6. A brief introduction to CSP is included in Appendix A.

The main contributions of this paper are:

– An application of data independence techniques to the analysis of a stream protocol, demonstrating the feasibility of applying model checking techniques to protocols of this class;
– An extension of existing techniques so as to deal with hash-chaining;
– The presentation of a number of state-space reduction techniques.

2 The TESLA Protocol

The TESLA protocol is designed to enable authentication of a continuous stream of packets over an unreliable medium. One important factor is efficiency of throughput. Therefore, apart from the initial message, the protocol does not make use of expensive cryptographic primitives such as public key signatures; instead it makes use of message authentication codes (MACs) and commitments using cryptographic hashes.

Perrig *et al.* [11] introduce 5 schemes for the TESLA protocol, each addressing additional requirements to the previous one. We will consider only the first two of these schemes in this paper.

Informally, Scheme 1 of TESLA works as follows. An initial authentication is achieved using a public key signature; subsequent messages are authenticated using MACs, linked back to the initial signature.

In message $n-1$, the sender S generates a key k_n, and transmits $f(k_n)$, where f is a suitable cryptographic hash function, to the receivers, as a commitment to that key; in message n, S sends a data packet m_n, authenticated using a MAC with key k_n; the key itself is revealed in message $n+1$. Each receiver checks that the received k_n corresponds to the commitment received in message $n-1$, verifies the MAC in message n, and then accepts the data packet m_n as authentic. Message n also contains a commitment to the next key k_{n+1}, authenticated by the MAC, thus allowing a chain of authentications.

More formally, the initial messages are as follows:[1]

$$\begin{array}{lll} \text{Msg } 0a. & R \to S & : \ n_R \\ \text{Msg } 0b. & S \to R & : \ \{f(k_1), n_R\}_{SK(S)} \\ \text{Msg } 1. & S \to R & : \ D_1,\ MAC(k_1,\ D_1) \text{ where } D_1 = \langle m_1,\ f(k_2)\rangle \\ \text{Msg } 2. & S \to R & : \ D_2,\ MAC(k_2,\ D_2) \text{ where } D_2 = \langle m_2,\ f(k_3),\ k_1\rangle. \end{array}$$

n_R is a nonce generated by the receiver to ensure freshness. The signature on $f(k_1)$ in message 0b acts as an authenticated commitment to k_1. When R receives k_1 in message 2, he can be sure that it really was sent by S. This allows R to verify the MAC in message 1, and so be assured of the authenticity of the data m_1. Further, R is assured of the authenticity of $f(k_2)$, the commitment to the next key.

For $n > 1$, the n-th message is:[2]

$$\text{Msg } n. \quad S \to R \ : \ D_n,\ MAC(k_n,\ D_n) \text{ where } D_n = \langle m_n,\ f(k_{n+1}),\ k_{n-1}\rangle.$$

An authenticated commitment to k_{n-1} will have been received in message $n-2$; the receipt of this key assures R of the authenticity of the data received in message $n-1$, and also the commitment to k_n in that message. Later, when k_n is received in message $n+1$, R will be assured of the authenticity of the data m_n and the commitment $f(k_{n+1})$ in message n.

The protocol requires an important time synchronisation assumption, the *security condition*: the receiver will not accept message n if it arrives after the sender might have sent message $n+1$ (otherwise an intruder can capture message $n+1$, and use the key k_n from within it to fake a message n). Thus the agents' clocks need to be loosely synchronised; this is achieved within an initial exchange.

Scheme II differs from Scheme I by not generating a stream of fresh keys, but instead generating a single key k_m, and calculating each key k_n, for $n < m$, by hashing k_m $m-n$ times: $k_n = f^{m-n}(k_m)$. Thus each key acts as a commitment to the next: when the receiver obtains k_n he can verify that $f(k_n) = k_{n-1}$. The explicit commitment can be dropped and the protocol simplified to:

$$\begin{array}{lll} \text{Msg } 0a. & R \to S & : \ n_R \\ \text{Msg } 0b. & S \to R & : \ \{k_0, n_R\}_{SK(S)} \\ \text{Msg } 1. & S \to R & : \ m_1,\ MAC(k_1,\ m_1). \end{array}$$

[1] The message numbering scheme is a bit clumsy, but has the advantage that data packet m_n is received in message n and authenticated using key k_n.

[2] In the original version [11], the n-th MAC is authenticated using $f'(k_n)$, instead of k_n, where f' is a second hash function; we can omit the use of f' for modelling purposes.

And for $n > 1$:

$$\text{Msg } n. \quad S \to R \; : \; D_n, \, MAC(k_n, \, D_n) \text{ where } D_n = \langle m_n, \, k_{n-1} \rangle.$$

One aim of this second version is to be able to tolerate an arbitrary number of packet losses, and to drop unauthenticated packets, yet continue to authenticate later packets.

3 Modelling the Basic Protocol

In this section, we present a natural, but infinite state model of the basic protocol (Scheme I), explaining some of the techniques we use for modelling and analysing security protocols within the CSP/FDR framework. In the next section we reduce this system to an equivalent finite version, by applying data independence techniques.

The basic idea is to build CSP models of the sender and receiver, following the protocol definition. We also model the most general intruder who can interact with the protocol, overhearing, intercepting and faking messages; however, as is standard, we assume strong cryptography, so we do not allow the intruder to encrypt or decrypt messages unless he has the appropriate key. These processes synchronise upon appropriate events, capturing the assumption that the intruder has control over the network, and so can decide what the honest agents receive. We then capture the security requirements, and use FDR to discover if they are satisfied. See [9, 10, 16] for more details and examples of the technique.

We will actually consider a slightly simplified version of the protocol: we omit the old key k_{n-1} from the MAC, since it seems to be redundant:

$$\text{Msg } n. \quad S \to R \; : \; m_n, \, f(k_{n+1}), \, k_{n-1}, \, MAC(k_n, \, \langle m_n, \, f(k_{n+1}) \rangle).$$

This simplification is *fault-preserving* in the sense of Hui and Lowe [7]: if there is an attack upon the original protocol, there is also an attack upon this simplified protocol; hence if we can verify the simplified protocol, we will have verified the original protocol.

3.1 Sender and Receiver Processes

The sender and receiver nodes are modelled as standard CSP processes that can perform *send* and *receive* events according to the protocol description. We assume that the intruder has complete control over the communications medium; hence all communication goes through the intruder, as in Figure 1.

As discussed in Section 2, the TESLA protocol relies upon a time synchronisation between the sender and receiver processes (known as the *Security Property* in [11]). We capture this requirement in our models by introducing the special event *tock* that represents the passage of one time unit. The processes representing the sender S and receiver R synchronise upon *tock*, modelling the fact that

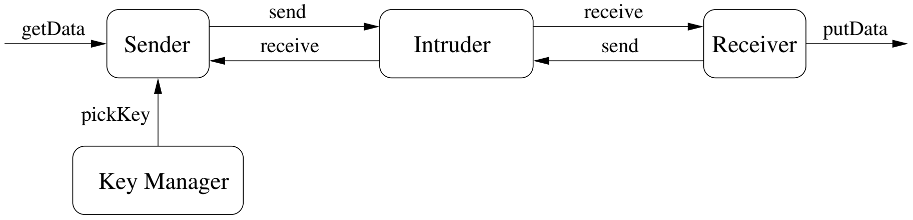


Fig. 1. Overview of the network

the two agents' clocks are loosely synchronised. This allows R to tell whether it has received message n before S might have sent message $n+1$.

We begin by presenting a process to represent the sender. It would be natural to parameterise this process by the infinite sequence of keys that are used to authenticate messages. However, in order to ease the transition to the next model, we instead arrange for the keys to be provided by an external process, *KeyManager*, and for the sender to obtain keys on a (private) channel *pickKey* (see Figure 1).

The sender is initially willing to receive an authentication request (message 0a) containing a nonce n_R; it responds by obtaining a key (k_1) from the key manager, and sending back the initial authentication message (message 0b); it then waits until the next time unit before becoming ready to send the next message; if no initial authentication request is received, it simply stays in the same state:

$$\begin{aligned}
&\mathit{Sender}_0(S) = \\
&\quad \Box\, R : \mathit{Agent},\, n_R : \mathit{Nonce} \bullet \\
&\qquad \mathit{receive} \,.\, R \,.\, S \,.\, (\mathit{Msg}_{0a},\, \langle n_R \rangle) \rightarrow \mathit{pickKey}?k_1 \rightarrow \\
&\qquad \mathit{send} \,.\, S \,.\, R \,.\, (\mathit{Msg}_{0b},\, \langle \{f(k_1), n_R\}_{SK(S)} \rangle) \rightarrow \mathit{tock} \rightarrow \mathit{Sender}_1(S, R, k_1) \\
&\quad \Box \\
&\quad \mathit{tock} \rightarrow \mathit{Sender}_0(S).
\end{aligned}$$

The sender obtains the next key (k_2), gets the first piece of data to be sent (m_{curr}) from some channel *getData*, and then sends the data and the commitment to the next key, including a MAC using the current key (message 1); it then waits until the next time unit before becoming ready to send the next message:

$$\begin{aligned}
&\mathit{Sender}_1(S,\, R,\, k_1) = \\
&\quad \mathit{pickKey}?k_2 \rightarrow \mathit{getData} \,.\, S?m_{curr} \rightarrow \\
&\quad \mathit{send} \,.\, S \,.\, R \,.\, (\mathit{Msg}_1, \langle m_{curr},\, f(k_2),\, \mathit{MAC}(k_1,\, \langle m_{curr},\, f(k_2) \rangle) \rangle) \rightarrow \\
&\quad \mathit{tock} \rightarrow \mathit{Sender}_n(S,\, R,\, k_1,\, k_2).
\end{aligned}$$

Subsequent behaviour is very similar to the previous step, except the previous key (k_{prev}) is included in each message. This is the last time the sender uses this key, so he can now forget it, modelled by the event $\mathit{forget} \,.\, k_{prev}$; we include this forgetting as an explicit event to ease the transition to the next model:

$$\begin{array}{l} \mathit{Sender}_n(S,\, R,\, k_{prev},\, k_{curr}) = \\ \quad \mathit{pickKey}?k_{next} \rightarrow \mathit{getData}\,.\,S?m_{curr} \rightarrow \\ \quad \mathit{send}\,.\,S\,.\,R\,.\,(\mathit{Msg}_n,\, \langle m_{curr},\, f(k_{next}),\, k_{prev}, \\ \qquad\qquad\qquad\qquad MAC(k_{curr},\, \langle m_{curr},\, f(k_{next})\rangle)\rangle) \rightarrow \\ \quad \mathit{forget}\,.\,S\,.\,k_{prev} \rightarrow \mathit{tock} \rightarrow \mathit{Sender}_n(S,\, R,\, k_{curr}, k_{next}). \end{array}$$

It is straightforward to model the process *KeyManager* that issues keys; it is parameterised by the sequence of keys that are issued:

$$\mathit{KeyManager}(xs) = \mathit{pickKey}\,.\,\mathit{head}(xs) \rightarrow \mathit{KeyManager}(\mathit{tail}(xs)).$$

It is precisely this mechanism of generating fresh keys that needs to be adapted in Section 4 in order to reduce the model to an equivalent finite version.

It is convenient to define the sets *AllCommits* and *AllMacs* of commitments and MACs that the receiver might receive. The receiver is unable to tell immediately whether it has received a valid commitment or MAC, so should also be willing to receive an arbitrary bit-string, modelled using a special value *Garbage*:

$$\begin{array}{rcl} \mathit{AllCommits} & = & \{f(k) \mid k \in \mathit{Key}\} \cup \{\mathit{Garbage}\}, \\ \mathit{AllMacs} & = & \{MAC(k,\, \langle m,\, f\rangle) \mid k \in \mathit{Key},\, m \in \mathit{Packet}, f \in \mathit{AllCommits}\} \\ & & \cup \{\mathit{Garbage}\}. \end{array}$$

The receiver begins by sending a nonce to the sender as an authentication request (message 0a). It then becomes willing to receive an appropriate initial authentication message (message 0b); it becomes ready to receive the next message in the next time unit. If the initial authentication message is not received before the end of the current time unit, the receiver should abort.

$$\begin{array}{l} \mathit{Receiver}_0(R,\, n_R) = \\ \quad \sqcap\, S : \mathit{Agent} \bullet \mathit{send}\,.\,R\,.\,S\,.\,(\mathit{Msg}_{0a},\, n_R) \rightarrow \mathit{Receiver}_0'(R,\, S,\, n_R), \\ \mathit{Receiver}_0'(R,\, S,\, n_R) = \\ \quad \square\, f_{next} : \mathit{AllCommits} \bullet \\ \qquad \mathit{receive}\,.\,S\,.\,R\,.\,(\mathit{Msg}_{0b},\, \{f_{next},\, n_R\}_{SK(S)}) \rightarrow \\ \qquad \mathit{tock} \rightarrow \mathit{Receiver}_1(R,\, S,\, f_{next}) \\ \quad \square \\ \quad \mathit{tock} \rightarrow \mathit{Abort}(R). \end{array}$$

The receiver is then willing to receive an appropriate message 1; note that it should be willing to accept an arbitrary key commitment and MAC, because it is not yet able to verify either. If the message is not received before the end of the time unit, the receiver should abort.

$$\begin{array}{l} \mathit{Receiver}_1(R,\, S,\, f_{curr}) = \\ \quad \square\, m_{curr} : \mathit{Packet},\, f_{next} : \mathit{AllCommits},\, mac_{curr} : \mathit{AllMacs} \bullet \\ \qquad \mathit{receive}\,.\,S\,.\,R\,.\,(\mathit{Msg}_1, \langle m_{curr},\, f_{next},\, mac_{curr}\rangle) \rightarrow \\ \qquad \mathit{tock} \rightarrow \mathit{Receiver}_n(R,\, S,\, f_{curr},\, f_{next},\, m_{curr},\, mac_{curr}) \\ \quad \square \\ \quad \mathit{tock} \rightarrow \mathit{Abort}(R). \end{array}$$

The receiver has now reached the phase where it can receive a message each time unit. For each message received, the authenticity of the previous message is verified by checking that (i) $f(k_{prev})$ is equal to the key commitment in the previous message, f_{prev}; and (ii) the MAC is authentic, i.e. $MAC(k_{prev}, \langle m_{prev}, f_{curr} \rangle)$ is equal to the MAC sent in the previous message, mac_{prev}. If these conditions are satisfied, then the receiver outputs the data on a channel *putData*; he then forgets the old key and moves to the next time unit. If either of these checks fails, or no message arrives, then the protocol run is aborted.

```
$Receiver_n(R,\ S,\ f_{prev}, f_{curr}, m_{prev}, mac_{prev}) =$
    $\square\ m_{curr} : Packet,\ f_{next} : AllCommits,\ k_{prev} : Key,\ mac_{curr} : AllMacs \bullet$
        $receive\,.\,S\,.\,R\,.\,(Msg_n, \langle m_{curr},\ f_{next},\ k_{prev},\ mac_{curr} \rangle) \rightarrow$
        $if\ f(k_{prev}) = f_{prev} \wedge MAC(k_{prev}, \langle m_{prev}, f_{curr} \rangle) = mac_{prev}\ then$
            $putData\,.\,R\,.\,S\,.\,m_{prev} \rightarrow forget\,.\,R\,.\,k_{prev} \rightarrow$
            $tock \rightarrow Receiver_n(R,\ S,\ f_{curr},\ f_{next},\ m_{curr},\ mac_{curr})$
        $else\ Abort(R)$
    $\square$
    $tock \rightarrow Abort(R).$
```

If the receiver aborts a protocol run, it simply allows time units to pass.

$$Abort(R) = tock \rightarrow Abort(R).$$

3.2 Intruder Process

The intruder model follows the standard Dolev-Yao [3] model. He is able to act as other agents, which might or might not behave in a trustworthy way; overhear all network messages; prevent messages from reaching their intended recipients; and finally, fake messages to any agent, purporting to be from any other.

The formal model is built around a set *Deductions* representing the ways in which the intruder can deduce new messages from messages he already knows, for example by encrypting and decrypting with known keys: each element (m, S) of *Deductions* represents that from the set of messages S, he can deduce the message m. The intruder process is parameterised by the set *IK* of messages that he currently knows. He can intercept messages (on the channel *send*) and add them to his knowledge; send messages that he knows to honest agents (on the channel *receive*); and deduce new messages from messages he already knows.

```
$Intruder(IK) =$
    $send?A\,.\,B\,.\,m \rightarrow Intruder(IK \cup \{m\})$
    $\sqcap$
    $\bigsqcap m : IK,\ A, B : Agent \bullet receive\,.\,A\,.\,B\,.\,m \rightarrow Intruder(IK)$
    $\sqcap$
    $\bigsqcap (m, S) : Deductions,\ S \subseteq IK \bullet infer\,.\,(m, S) \rightarrow Intruder(IK \cup \{m\}).$
```

For reasons of efficiency, the actual intruder implementation used is built as a highly parallel composition of many two-state processes, one process for each fact that the intruder could learn. This model was developed by Roscoe and Goldsmith [15] and is equivalent to the process above.

The intruder's knowledge is initialised to a set *IIK* containing values that the intruder could be expected to know, including all agents' identities, all public keys, all data values, his own secret key, a nonce and key different from those used by the honest agents, and the value *Garbage* that represents an arbitrary bit-string.

3.3 Overall System and Specification

The above processes are combined together in parallel, as illustrated in Figure 1, and all events except for the *getData*, *putData* and *tock* events are hidden (made internal):

$$\begin{aligned}
Agents &= Sender_0(S) \underset{\{\!|pickKey|\!\}}{\parallel} KeyManager(Ks)) \mathbin{|||} Receiver_0(R, N_1),\\
System_0 &= Agents \underset{\{\!|send, receive|\!\}}{\parallel} Intruder(IIK),\\
System &= System_0 \setminus \{\!|\; send, receive, pickKey, forget \;|\!\}.
\end{aligned}$$

We now verify that the system meets a suitable specification. According to Perrig *et al.* [11], TESLA guarantees that the receiver does not accept as authentic any message m_n unless m_n was sent by the sender; in fact, m_n must have been sent in the previous time interval.

We capture this by the following CSP specification. The specification captures our security requirement by defining the order and timing by which messages can be sent by the sender and get accepted by the receiver as authentic. We can use FDR to verify that all traces (i.e. sequences of external events) of the system are traces allowed by the specification.

In the initial state, the specification allows the sender S to input a packet m_{curr}, before evolving after one time unit into the state $Spec'(m_{curr})$. Alternatively, time is allowed to pass without changing the state of the specification.

$$\begin{aligned}
Spec = \;& getData\,.\,S?m_{curr} \rightarrow tock \rightarrow Spec'(m_{curr})\\
& \sqcap\\
& tock \rightarrow Spec.
\end{aligned}$$

Subsequently, the receiver can, but might not, output on *putData* the value m_{prev} received in the previous time unit. The sender can input a new value m_{curr}. These events can happen in either order, giving the following specification.

$$\begin{aligned}
& Spec'(m_{prev}) =\\
& \quad getData\,.\,S?m_{curr} \rightarrow tock \rightarrow Spec'(m_{curr})\\
& \quad \sqcap\\
& \quad getData\,.\,S?m_{curr} \rightarrow putData\,.\,R\,.\,S\,.\,m_{prev} \rightarrow tock \rightarrow Spec'(m_{curr})\\
& \quad \sqcap\\
& \quad putData\,.\,R\,.\,S\,.\,m_{prev} \rightarrow getData\,.\,S?m_{curr} \rightarrow tock \rightarrow Spec'(m_{curr}).
\end{aligned}$$

The model checker FDR can be used to verify that the system meets this specification when the key manager is initialised with a finite sequence of keys; that is, we verify $\mathit{Spec} \sqsubseteq \mathit{System}$, which is equivalent to $\mathit{traces}(\mathit{System}) \subseteq \mathit{traces}(\mathit{Spec})$.

The following section describes how to adapt the model to deal with an infinite sequence of keys.

4 A Finite Model of an Infinite System

In this section we show how to transform the previous model of TESLA to an equivalent finite model, which can be analysed using FDR. We make use of the data independence techniques developed by Roscoe [13].

The data independence techniques allow us to simulate a system where agents can call upon an unbounded supply of fresh keys even though the actual type remains finite. In turn this enables us to construct models of protocols where agents can perform unbounded sequential runs and verify security properties for them within a finite check.

The basic idea is as follows: once a key is no longer held by any honest participant (i.e., each agent who held the key has performed a corresponding *forget* event), we *recycle* the key: that is, we allow the same key to be subsequently re-issued to the sender; hence, because there is a finite bound on the number of keys held by honest participants, we can make do with a finite number of keys.

However, at the point where we recycle a key k, the intruder's memory will include messages using k, since the intruder overhears all messages that pass on the network; in order to prevent FDR from discovering bogus attacks, based upon reusing the same key, we need to transform the intruder's memory appropriately. The transformation replaces k with a special key K_b, known as the *background key*; more precisely, every message containing k in the intruder's memory is replaced by a corresponding message containing K_b. The effect of this is that anything the intruder could have done using k before recycling, he can now do using K_b. Thus we collapse all old keys down to the single background key.

Note that this key recycling is a feature of the model rather than of the protocol. We argue below that it is safe in the sense that it does not lose attacks. Further, it turns out not to introduce any false attacks.

The technique is implemented within the CSP protocol model by adapting the key manager process (which issues keys to the sender), so that it keeps track of which honest agents currently hold which keys. It synchronises on the messages where agents acquire new keys (*pickKey* events in the case of the sender, and receipt of all messages in the case of the receiver) and release keys (*forget* events); when a key has been released by all agents, the manager triggers the recycling mechanism, remapping that key within the intruder's memory, as described above. (We need to adapt the model of the receiver slightly so that it releases all keys when it aborts.) That key can then be re-issued.

We write System' for the resulting system and System'_0 for the system without the top level of hiding (analogous to System_0). Since System' is a finite model, we can check that it refines *Spec* automatically using the model checker FDR.

In [14], Roscoe proves that this transformation is sound for protocol models that satisfy the special condition *Positive Conjunction of Equality Tests* [8]; informally, a process P satisfies this property when the result of an equality test proving false will never result in P performing a trace that it could not have performed were the test to prove true. The processes in our model satisfy this condition.

The critical property that justifies this transformation is

$$\mathit{System}' \sqsubseteq \mathit{System}. \tag{1}$$

If we use FDR to verify $\mathit{Spec} \sqsubseteq \mathit{System}'$, then we can deduce that $\mathit{Spec} \sqsubseteq \mathit{System}$, by the transitivity of refinement.

Let ϕ be the function over traces that replaces each key from the infinite model with the corresponding key in the finite model, in particular replacing keys that have been forgotten by all honest agents with the background key K_b. Equation (1) holds because of the following relationship between System_0 and System_0', noting that all the keys are hidden at the top level:

$$\mathit{traces}(\mathit{System}_0') \supseteq \{\phi(\mathit{tr}) \mid \mathit{tr} \in \mathit{traces}(\mathit{System}_0)\}. \tag{2}$$

In turn, property (2) holds because the corresponding property holds for each individual agent in the system[3]:

$$\begin{aligned} \mathit{traces}(\mathit{Agents}') &= \{\phi(\mathit{tr}) \mid \mathit{tr} \in \mathit{traces}(\mathit{Agents})\}, \\ \mathit{traces}(\mathit{Intruder}'(\mathit{IIK})) &\supseteq \{\phi(\mathit{tr}) \mid \mathit{tr} \in \mathit{traces}(\mathit{Intruder}(\mathit{IIK}))\}. \end{aligned}$$

The first property holds by construction of the new key manager, and because the sender and receiver processes are data independent in the type of keys. The latter holds because of the following property of the deduction system:

$$(m, S) \in \mathit{Deductions} \Rightarrow (\phi(m), \phi(S)) \in \mathit{Deductions}.$$

which says that for any deduction the intruder can make in the original system, he can make the corresponding deduction in the reduced system.

Modelling the protocol directly as above is fine in theory, but in practice leads to an infeasibly large state space. Below we discuss a number of implementation strategies that help overcome this problem. For ease of presentation, we discuss the optimisations independently, although in practice we combined them.

Number of data packets One interesting question concerns the number of distinct data packets required in our model. It turns out that two distinct packets M_0 and M_1 suffice: if the protocol fails in a setting where more than two packets are used, then it will also fail in the model with just two packets. To see why this is the case informally, suppose there were an error in the former case represented by trace tr; this error must be because the receiver receives a particular packet $\widehat{M}$

[3] We write "*Agents*′" and "*Intruder*′" for the new models of the honest agents and intruder, respectively.

incorrectly; consider the effect of replacing in tr all occurrences of $\widehat{M}$ by M_1, and all occurrences of other packets by M_0; it is easy to see that this would be a trace of the model with just two packets, and would be an error because the receiver would accept M_1 incorrectly. This argument can be formalised, making use of a general theorem by Lazić [8] [12, Theorem 15.2.2].

Splitting protocol messages Large protocol messages comprising many fields are expensive to analyse using FDR, because they lead to a large message space and a large degree of branching in the corresponding processes. Splitting such messages into several consecutive ones is a simple, yet effective reduction technique that we frequently use to handle this situation. In the case of TESLA, we split the main message n into two messages with contents $\langle m_n, f(k_{n+1}), k_{n-1}\rangle$ and $\langle MAC(k_n, \langle m_n, f(k_{n+1})\rangle)\rangle$. This strategy speeds up checking by an order of a magnitude. Hui and Lowe prove in [7, Theorem 11] that this transformation is sound in the sense that a subsequent verification of the transformed protocol implies the correctness of the original protocol.

Associating incorrect MACs The model of the receiver allows him to receive an arbitrary MAC in each message:

$$\begin{aligned}
&\mathit{Receiver}_n(R,\ S,\ f_{prev}, f_{curr}, m_{prev}, mac_{prev}) = \\
&\quad \Box \ldots mac_{curr} : \mathit{AllMacs} \bullet \\
&\qquad \mathit{receive} \,.\, S \,.\, R \,.\, (\mathit{Msg}_n, \langle m_{curr},\ f_{next},\ k_{prev},\ mac_{curr}\rangle) \to \ldots,
\end{aligned}$$

where *AllMacs* contains all valid MACs and also the special value *Garbage* that models a bit-string not representing a valid MAC. However, if the MAC does not correspond to m_{curr} and f_{next} then it will be rejected at the next step. Therefore the above model allows the receiver to receive many different incorrect MACs, all of which are treated in the same way. We can reduce the state space by an order of magnitude by collapsing all of these incorrect MACs to *Garbage*, noting that there is no essential difference between a behaviour using this value and a behaviour using a different incorrect MAC. Thus we rewrite the receiver to:

$$\begin{aligned}
&\mathit{Receiver}_n(R,\ S,\ f_{prev}, f_{curr}, m_{prev}, mac_{prev}) = \\
&\quad \Box \ldots mac_{curr} : \{MAC(k, \langle m_{curr}, f_{next}\rangle) \mid k \in \mathit{Key}\} \cup \{\mathit{Garbage}\} \bullet \\
&\qquad \mathit{receive} \,.\, S \,.\, R \,.\, (\mathit{Msg}_n, \langle m_{curr},\ f_{next},\ k_{prev},\ mac_{curr}\rangle) \to \ldots.
\end{aligned}$$

This transformation can be justified in a similar way to the transformation that reduced the key stream to a finite stream. Equation (2) holds when we use the reduction function ϕ that maps all incorrect MACs onto *Garbage*.

Placement of tests One can also obtain a large speed up by careful placement of the test that verifies the previous MAC. In the earlier model, the receiver was willing to receive any key in message n, and then checked the previous MAC. It is more efficient to simply refuse to input any key that does not allow the

MAC to be verified, as follows:

$$\begin{array}{l} Receiver_n(R,\ S,\ f_{prev}, f_{curr}, m_{prev}, mac_{prev}) = \\ \quad \Box\ k_{prev} : Key,\ f(k_{prev}) = f_{prev} \wedge MAC(k_{prev}, \langle m_{prev}, f_{curr}\rangle) = mac_{prev} \bullet \\ \quad \Box\ m_{curr} : Packet,\ f_{next} : AllCommits,\ mac_{curr} : AllMacs \bullet \\ \qquad receive\,.\,S\,.\,R\,.\,(Msg_n, \langle m_{curr},\ f_{next},\ k_{prev},\ mac_{curr}\rangle) \rightarrow \ldots . \end{array}$$

This reduction can be justified in a similar way to previous reductions.

Placement of forget actions Recall that the sender and receiver perform *forget* events to indicate that they have finished using a key. It turns out that placing this *forget* event as early as possible—i.e. immediately after the last event using that key—leads to a large reduction in the size of the state space.

Combining events In our previous model, we included different events within the sender for obtaining the key from the key manager and for sending the message using that key. It is more efficient to combine these events into a single event, arranging for the sender to be able to use an arbitrary key, and arranging for the key manager to allow an arbitrary send event using the key it wants to supply next, and synchronising these two processes appropriately. It is also possible to combine the *getData* event with the *send* event, meaning that the sender's environment chooses the value sent. Once the whole system has been combined, a renaming can be applied for compatibility with the specification. It is also possible to combine the *forget* events with appropriate messages, although we have not done this.

Combining these state space reduction techniques with the key recycling mechanism, we have been able to construct a reduced model of TESLA that simulates the key chain mechanism. The agent processes are able to perform an unbounded number of runs; the use of the recycling mechanism gives the appearance of an infinite supply of fresh keys within a finite state model.

5 Modelling Key Chaining and Re-Authentication

In this section we show how to adapt the protocol so as to deal with Scheme II, where explicit key commitments are omitted, but instead each key is formed as the hash of the following key. Most of the adaptation is straightforward, except for the key chaining, which we describe below.

5.1 Modelling Key Chaining

Recall that the n-th key k_n is calculated by hashing some fixed key k_m $m-n$ times. The receiver should accept a key k_{curr} only after checking that it hashes to the previously accepted key k_{prev}. Clearly we cannot model this hash-chaining through explicit modelling of the hashes, for there is no fixed bound on the number of hashes, and so the state space would be infinite.

Our model of the hash-chaining instead makes use of the following observation: k_{curr} hashes to k_{prev} precisely when k_{prev} and k_{curr} are keys that were issued consecutively by the key manager. We therefore introduce a *key checker* process that keeps track of the order in which keys were issued, and allows the event $check\,.\,k_1\,.\,k_2$ if k_1 and k_2 were consecutively issued; we then model the checking of hashes by having the receiver process attempt the event $check\,.\,k_{prev}\,.\,k_{curr}$; synchronising the two processes upon *check* events ensures that only appropriate checks succeed. In more detail:

– The key checker allows an event $check\,.\,k_1\,.\,k_2$ for fresh keys k_1 and k_2 precisely when those keys were consecutively issued.

Hence the receiver will accept a foreground key precisely when it was issued immediately after the previously accepted key, which corresponds to the case that it hashes to the previous key. The key checker should allow additional *check* events in the situation where the intruder has managed to get the receiver to accept a key that has already been recycled:

– The key checker allows the event $check\,.\,K_b\,.\,k$ for fresh or background keys k.

The key checker should at least allow $check\,.\,K_b\,.\,k$ if k is the key issued immediately after a fresh key that could correspond to this K_b (because a corresponding *check* event would have been possible if we were not using the recycling mechanism). We allow more such *check* events, which is sound, because it only gives the intruder more scope to launch attacks; this turns out not to introduce any false attacks, essentially because if the intruder had managed to get the receiver to accept K_b, then he would have already broken the protocol anyway.

One might have expected the key checker to also allow events of the form $check\,.\,k_1\,.\,K_b$ for fresh keys k_1 when the key issued after k_1 has been recycled. However, it turns out that allowing such checks introduces false attacks into the model: if the key following k_1 has been recycled, the intruder can fake a MAC using the background key, even though there is no corresponding behaviour in the infinite state model. We avoid this problem by enforcing that keys are recycled in the same order in which they are issued.

We can then model the receiver process as follows:

$$
\begin{aligned}
&Receiver_n(R,\ S,\ k_{prev}, m_{prev}, mac_{prev}) = \\
&\quad \Box\ k_{curr} : Key,\ MAC(k_{curr}, m_{prev}) = mac_{prev} \bullet \\
&\qquad check.k_{prev}.k_{curr} \rightarrow forget.R.k_{prev} \rightarrow \\
&\qquad \Box\ m_{curr} : Packet,\ mac_{curr} : AllMacs \bullet \\
&\qquad\quad receive\,.\,S\,.\,R\,.\,(Msg_n, \langle m_{curr},\ k_{curr},\ mac_{curr}\rangle) \rightarrow \ldots.
\end{aligned}
$$

The reduction, from a model that uses explicit hashes to form the keys to the finite model described above, can be justified as before. Equation (2) holds for the function ϕ over traces that replaces keys and inserts *check* events appropriately. In particular

$$traces((Receiver' \underset{X}{\parallel} KeyChecker) \setminus Y) \supseteq \{\phi(tr) \mid tr \in traces(Receiver)\},$$

where $X = \{\!| \; check \; |\!\}$ and $Y = \{\!| \; getKey \; |\!\}$.

We can also adapt the model of the protocol to allow the receiver to attempt new initial re-authentications. However, the receiver should use a different nonce for each re-authentication, and so we need to model an infinite supply of nonces. Doing so is simple: we simply recycle nonces in the same way that we recycled keys, using a nonce manager that issues fresh nonces to the receiver, and recycles nonces that are no longer stored by any agent.

6 Conclusions

In this paper, we described how to model and analyse the Timed Efficient Stream Loss-tolerant Authentication Protocol presented in [11], using model checking techniques within the CSP/FDR framework. This was initially motivated by a challenging statement by Archer [2] claiming that this class of protocols is infeasible to analyse using model checking techniques.

We started by presenting a CSP model for Scheme I of the protocol using an unbounded supply of fresh cryptographic keys. We then showed how one can apply the data independence techniques presented in [13, 14] to reduce this system to an equivalent finite version. This was achieved by implementing the recycling mechanism upon a finite set of fresh keys, creating the necessary illusion of having an unbounded stream of them. A number of reduction strategies were developed to keep the state space within a feasible range.

This protocol model was then extended to capture Scheme II. In particular, we had to develop a technique for modelling unbounded hash-chains. We also had to adapt the model to allow recycling of nonces, so as to allow an unbounded number of repeat authentications.

Our analysis showed that the protocol is correct, at least when restricted to a system comprising a single sender and a single receiver. It is interesting to ask whether we can extend this result to larger systems:

– It is easy to extend this result to multiple receivers: the intruder would not learn anything from interactions with additional receivers except more nonces. These would not help him, because he could only use them in the same way that he could use his own nonces; further, any failure of the protocol when executed by multiple receivers would correspond to a failure for one of those receivers, which we would have discovered in the above analysis.
– It is also easy to extend the result to multiple senders with different identities: if the intruder were able to replay messages from a server S_2 and have them accepted as coming from another server S_1, then in the model of this paper he would have been able to use his own messages in the same way as the messages of S_2 so as to have them accepted as coming from S_1, which again we would have discovered in the above analysis.
– Finally, it is not possible to extend the result to the case where a single server runs the protocol multiple times simultaneously, issuing different data streams, but using the same key for the initial authentications; indeed in this

circumstance it is possible for the intruder to cause confusion between these data streams.

Our model of Scheme II allows the receiver to recover from the loss or corruption of packets only by performing a repeat authentication. TESLA itself allows the receiver to recover from d successive lost or corrupted packets, by checking that the next key received, when hashed $d + 1$ times, gives the previously accepted key. It would be interesting to extend our model to consider this case, at least for a limited range of values of d. However, we believe that it is not possible to consider this extension for arbitrary values of d, because the system seems inherently infinite state. Further, even for small values of d, one would run into problems with a state space explosion; however, a parallel version of FDR, designed for execution on a network of computers, is now available, which could be employed on this problem.

We would also like to investigate an alternative method for recycling the keys: use a fixed set of keys, $K_1, \ldots, K_n$ for a suitable value of n (in the setting of Section 5, we need $n = 5$) and arrange that at each stage the new key issued is K_1; in order for this to work, we would need to arrange a re-mapping of keys on each *tock* event, both within the states of the honest agents and the intruder: we replace K_1 with K_2, replace K_2 with K_3, ..., and replace K_n with the background key K_b. Thus each key K_i represents the key introduced $i - 1$ time units ago. The effect of this is that each message sent by the sender would introduce K_1, have the MAC authenticated using K_2, and disclose K_3. This increased regularity should cause a decrease in the size of the state space.

TESLA is based upon the Guy Fawkes Protocol [1]; it would seem straightforward to adapt the techniques of this paper to that protocol. A further protocol of interest is the second one presented in [11], the Efficient Multi-chained Stream Signature Protocol (EMSS). TESLA does not provide non-repudiation; EMSS is designed to fulfill this requirement. Non-repudiation is a property that is easily captured within the CSP/FDR framework; see [6].

References

[1] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, 1998. 160

[2] M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME. In *Workshop on Issues in the Theory of Security*, 2002. 146, 159

[3] D. Dolev and A. C. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983. 152

[4] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR2 User Manual*, 2000. At `http://www.fsel.com/fdr2_manual.html`. 147

[5] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. 147

[6] M. L. Hui. *A CSP Approach to the Analysis of Security Protocols*. PhD thesis, University of Leicester, 2001. 160

[7] M. L. Hui and G. Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Science*, 9(1, 2):3–46, 2001. 149, 156
[8] R. Lazić. *Theorems for mechanical verification of data-independent CSP*. D.Phil, Oxford University, 1999. 155, 156
[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996. 149
[10] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997. 149
[11] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, May 2000. 146, 147, 148, 149, 153, 159, 160
[12] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997. 147, 156
[13] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *11th IEEE Computer Security Foundations Workshop*, pages 84–95, 1998. 147, 154, 159
[14] A. W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(2, 3):147–190, 1999. 155, 159
[15] A. W. Roscoe and M. H. Goldsmith. The perfect 'spy' for model-checking crypto-protocols. In *Proceedings of DIMACS workshop on the design and formal verification of cryptographic protocols*, 1997. 153
[16] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Pearson Education, 2001. 149

A CSP Notation

An event represents an atomic communication; this might either be between two processes or between a process and the environment. Channels carry sets of events; for example, $c.5$ is an event of channel c. The event *tock* represents the passage of one unit of time.

The process $a \rightarrow P$ can perform the event a, and then act like P. The process $c?x \rightarrow P_x$ inputs a value x from channel c and then acts like P_x.

The process $P \;\Box\; Q$ represents an external choice between P and Q; the initial events of both processes are offered to the environment; when an event is performed, that resolves the choice. $P \sqcap Q$ represents an internal or nondeterministic choice between P and Q; the process can act like either P or Q, with the choice being made according to some criteria that we do not model.

The external and internal choice operators can also be distributed over a set of processes. $\Box\, i : I \bullet P_i$ and $\bigsqcap i : I \bullet P_i$ represent replicated external and nondeterministic choices, indexed over set I. The range of indexing can be restricted using a predicate; for example, $\Box\, i : I, p(i) \bullet P_i$ restricts the indexing to those values i such that $p(i)$ is true.

$P \underset{A}{\parallel} Q$ represents the parallel composition of P and Q, synchronising on events in A. $P \;|||\; Q$ represents the parallel composition of P and Q without any synchronisation. $P \setminus A$ represents P with the events in A hidden, i.e. made internal.

Equal To The Task?

James Heather[1] and Steve Schneider[2]

[1] Department of Computing, University of Surrey
Guildford, Surrey GU2 7XH, UK
`j.heather@eim.surrey.ac.uk`

[2] Department of Computer Science, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
`steve@cs.rhul.ac.uk`

Abstract. Many methods of analysing security protocols have been proposed, but most such methods rely on analysing a protocol running only a finite network. Some, however—notably, data independence, the strand spaces model, and the rank functions model—can be used to prove correctness of a protocol running on an unbounded network.

Roscoe and Broadfoot in [17] show how data independence techniques may be used to verify a security protocol running on an unbounded network. They also consider a weakness inherent in the RSA algorithm, discovered by Franklin and Reiter [3], and show that their data independence approach cannot deal with an intruder endowed with the ability to exploit this weakness.

In this paper, we show that neither can the use of honest ideals in the strand spaces model or the use of rank functions in the CSP model be easily adapted to cover such an intruder. In each case, the inequality tests required to model the new intruder cause problems when attempting to extend analysis of a finite network to cover an unbounded network. The results suggest that more work is needed on adapting the intruder model to allow for cryptographic attacks.

Key words: cryptographic protocols; rank functions; strand spaces; data independence; inequality tests; RSA; formal methods in security; security models; security verification.

1 Introduction

In [3], Franklin and Reiter present an attack on the RSA algorithm [15], allowing decryption of a pair of distinct encrypted messages, in the case where the two ciphertexts stand in a known linear relation and are encrypted under the same RSA public key with a public-key exponent of 3.

Suppose that we have two encrypted messages

$$c_1 = \{m\}_k$$
$$c_2 = \{am + b\}_k$$

and that we know a and b (with $a \neq 0$, and not both $a = 1$ and $b = 0$), and the public key k, but we do not know m. Suppose further that the exponent of k is

3, and that the modulus is n. Then we have

$$\begin{aligned} c_1 &= m^3 \bmod n \\ c_2 &= (am+b)^3 \bmod n \end{aligned}$$

We can recover m by observing that

$$\begin{aligned} m \bmod n &= \frac{3a^3bm^3 + 3a^2b^2m^2 + 3ab^3m}{3a^3bm^2 + 3a^2b^2m + 3ab^3} \\ &= \frac{b(c_2 + 2a^3c_1 - b^3)}{a(c_2 - a^3c_1 + 2b^3)} \end{aligned}$$

Coppersmith et al. generalise this attack in [1] to deal with an arbitrary public-key exponent e, but the complexity of the attack is $O(e \log^2 e)$. They claim that the attack should be practical for exponents of up to 2^{32}. In particular, it is efficient with an exponent of $2^{16}+1$, a popular choice in many applications.

This paper explores the consequences of this attack for security protocol analysis. Specifically, it investigates the feasibility of tailoring existing protocol verification methods to allow an intruder to take advantage of the RSA attack described above. For one of these methods—data independence—it has already been shown by Roscoe and Broadfoot that the attack cannot be incorporated into the model without destroying the foundation on which the model is built. We demonstrate in this paper that two other current methods—rank functions for CSP models, and honest ideals for strand spaces—have surprisingly similar problems in allowing for this strengthened intruder.

The structure of the paper is as follows. In the next section, we describe how an intruder might use the RSA attack to break a security protocol, and give a brief overview of how one might attempt to model this capability when analysing a protocol. In Section 3, we summarise Roscoe and Broadfoot's findings with regard to the RSA attack and the data independence model. Section 4 describes the rank functions model, and investigates the possibility of including the RSA attack among the rank functions intruder's weapons. Section 5 deals with the same question for the strand spaces model. Finally, in Section 6, we sum up and draw conclusions.

2 Exploiting the Attack to Break Security Protocols

If concatenations of smaller protocol messages are formed by simple concatenations of the bit strings representing those messages, then the concept of a linear relation between messages is clear: forming the compound message $x.y$ will involve multiplying the value of x by $2^{\ell(y)}$ and adding the value of y (where '$\ell(y)$' denotes the length of the bit string representing y). The most natural formulation of this is when $m_1 = a.X.b$ and $m_2 = c.X.d$, with the intruder already knowing a, b, c and d (any of which could be null, but not both $a = c$ and $b = d$). For then

$$m_2 = 2^{\ell(d)-\ell(b)}(m_1 - b - 2^{\ell(X.b)}a) + d + 2^{\ell(X.d)}c$$

An intruder who knows $\{m_1\}_k$ and $\{m_2\}_k$ for some known RSA key k with a low encrypting exponent can make use of the attack given above to deduce the value of X.

2.1 Modelling the New Intruder

In each of the three models under consideration, the intruder has complete control over the entire network, in the style of the Dolev-Yao model [2]. The intruder may

– allow messages to be sent normally from agent to agent (but take note of the contents of the message in the process);
– intercept messages and fail to deliver them;
– construct and deliver spurious messages purporting to come from anyone he pleases.

In this last case, he may send any message that he has already seen in the network or that he can produce using only messages that he has seen. For instance, if he has observed N_A and N_B as separate messages, then he may construct $N_A.N_B$ from them and deliver this concatenation. (This concatenation operator is defined to be associative.)

To allow for the extra intruder capability of exploiting the RSA weakness when conducting protocol analysis, one has to introduce a new deduction rule, or type of penetrator strand, or the analogue in the model in question. With a rank functions approach, this will be an extra $\vdash$ rule (explained more fully in Section 4):

$$\{\{a.X.b\}_k, \{c.X.d\}_k, a, b, c, d\} \vdash X \qquad\qquad (a \neq c \text{ or } b \neq d)$$

In the strand spaces model we would extend the definition of a penetrator strand to include a new type **L** (see Section 5):

L ***Low-exp RSA*** $\langle -\{a.X.b\}_k, -\{c.X.d\}_k,$
$-k, -a, -b, -c, -d, +X\rangle$ for $a, b, c, d, X \in \mathsf{A}$; $a \neq c$ or $b \neq d$; and $k \in \mathsf{K}$.

The data independence CSP model would contain a new set of deductions along the following lines, describing how the intruder, already in possession of messages from a set Z, can extend his knowledge by utilising the RSA attack to deduce a new message f. The set *deductions*(Z) is a set of ordered pairs of the form $\langle S, m\rangle$, corresponding to $S \vdash m$ in the rank functions world (Section 3):

$$\begin{aligned} \mathit{deductions}(Z) = \{\langle\{a, b, c, d, k, \{a.f.b\}_k, \{c.f.d\}_k\}, f\rangle \mid \\ \{a.f.b\}_k \in Z, \{b.f'.d\}_{k'} \in Z, f = f', k = k', a \neq c \vee b \neq d\} \end{aligned}$$

This definition differs from the one given in [17] in three respects, none of great moment:

1. The name given in [17] is '`deductions5`'.

2. The key k has been erroneously omitted in [17] from the set of messages required in order to make the deduction; it is here included.
3. Roscoe and Broadfoot give the deduction from $\{a.f\}_k$ and $\{b.f\}_k$, and leave the more general case (with c and d included) for another (implied) deduction rule. Here, we give the full case with $\{a.f.b\}_k$ and $\{c.f.d\}_k$, and assume that the model will be constructed so that a message of the form $a.f$ can be parsed as $a.f.b$ with $b = \langle\rangle$ (the null message—and we naturally further assume that an intruder can produce the null message whenever required).

Note the inequality tests in each case. Without them, the rule would degenerate: in the case where $a = b = c = d = \langle\rangle$, we would have rank functions deductions of the form

$$\{\{X\}_k\} \vdash X$$

and penetrator strands of the form

$$\langle -\{X\}_k, +X\rangle$$

and a subset of *deductions*(Z) as

$$D(Z) = \{\langle\{\{f\}_k, k\}, f\rangle \mid \{f\}_k \in Z\}$$

In other words, the intruder would be able to break every encryption.

The remainder of this paper deals with the issues involved in attempting to include this new deduction rule or strand type in the analysis, and in particular with the effect on proving correctness of a security protocol running on an unbounded network.

3 Data Independence

Roscoe and Broadfoot [17] have demonstrated how one may analyse a security protocol running on a finite network, and then use Lazić's work on data independence [11, 12] to extend the results to cover unbounded networks.

Broadly speaking, a CSP process P, parameterised by a data type T, is said to be *data independent* with respect to T if the operation of P does not depend in any way on T. The results that Lazić presents give conditions under which the abstract data type T may be replaced with a small, concrete data type T' without affecting the whether P has particular properties.

Roscoe and Broadfoot put forward the concept of a *positive deductive system.* A deductive system is positive relative to a type T essentially if it treats all members of T equivalently, and never requires inequality between two members of type T in the set from which the deduction is made. They then show that if the intruder model is built over a positive deductive system, it satisfies Lazić's *Positive Conjunctions of Equality Tests* (**PosConjEqT**) property (or a slight variant), which requires that a process should not perform explicit or implicit equality tests except for equality tests after which the process halts whenever the test gives a negative result. For data independence techniques to apply to a model of a network running a security protocol, we need the process representing the intruder, and also every process representing an honest agent, to satisfy this property.

The authors use this result to replace infinite sets of nonces and keys with finite sets while ensuring that no attacks will be defeated by the replacement. Model checking can then be used to show that there are no attacks on a small system, and the data independence theorems provide the guarantee that there will consequently be no attacks on an unbounded system.

As Roscoe and Broadfoot make clear, however, giving the intruder the power to exploit the weakness in low-exponent RSA, by allowing him to make use of the set $deductions(Z)$ defined earlier, results in an intruder process that does not satisfy this condition. The intruder must check that he holds two distinct encryptions in order to perform the attack; and this check (the '$a \neq c \vee b \neq d$' in the definition of $deductions(Z)$) violates the **PosConjEqT** condition.

The necessity of the inequality check in specifying the RSA attack makes it impossible to work this new intruder capability into the data independence model without invalidating the results.

4 Rank Functions

In this section, we describe the rank functions approach to security protocol analysis, as set forth in [18, 9] and discuss the consequences of attempting to strengthen the rank functions intruder model to endow him with the ability to use the attack on low-exponent RSA.

Some knowledge of CSP is assumed. For an introduction to CSP, see [10, 16, 19].

4.1 The Network

The network considered in [9] consists of a (usually infinite) number of honest agents, and one dishonest enemy. The behaviour of an agent J is described as two CSP process U_J^I and U_J^R, controlling respectively J's actions as initiator and responder. These user processes will vary according to the protocol under consideration; they will consist of communication along channels *trans*, representing transmission of a message, and *rec*, representing reception.

We shall write '$\mathcal{U}$' for the set of user identities on the network (including identities of any dishonest agents), and '$\mathcal{N}$' for the set of nonces that can be used in protocol runs.

We define a 'generates' relation $\vdash$, writing '$S \vdash m$' to denote that the enemy may construct message m if he possesses every message in the set S. If m and n are messages, k is a key, and k^{-1} is the inverse of k, then $\vdash$ is the smallest relation that satisfies

$$\begin{aligned}
\{m, n\} &\vdash m.n \\
\{m.n\} &\vdash m \\
\{m.n\} &\vdash n \\
\{m, k\} &\vdash \{m\}_k \\
\{\{m\}_k, k^{-1}\} &\vdash m
\end{aligned}$$

and also satisfies the closure conditions

$$\begin{aligned}
m \in S &\Rightarrow S \vdash m \\
(\forall\, v \in T \bullet S \vdash v) \wedge T \vdash m &\Rightarrow S \vdash m
\end{aligned}$$

The enemy (already in possession of a set of messages S) is then described by the recursive definition:

$$\mathit{ENEMY}(S) =$$

$$\mathit{trans}?i?j?m \rightarrow \mathit{ENEMY}(S \cup \{m\}) \; \Box \; \mathop{\Box}_{i,j,S \vdash m} \mathit{rec}!i!j!m \rightarrow \mathit{ENEMY}(S)$$

Here the enemy can either receive any message m transmitted by any agent i to any other agent j along a *trans* channel, and then act as the enemy with that additional message; or pass any message m that it can generate from S to any agent i along its *rec* channel, remaining with the same information S.

The whole network is then

$$\mathbf{NET} = (\mathop{|||}_{J \in \mathcal{U}} (U_J^I \;|||\; U_J^R)) \parallel \mathbf{ENEMY}$$

For any given protocol, there will be a (possibly infinite) set of all atoms that could ever appear in a message of the protocol. This set will encompass all the user identities, nonces and keys, and any other types of atom used in the protocol (for instance, timestamps). From this set we can construct the *message space*, usually denoted by '$\mathcal{M}$', which is the space of all messages that can be generated from these atoms.

We use '*INIT*' to denote the set of atoms known to the enemy right from the start. Some users will be under the control of the enemy, and hence their secret keys and all nonces that they might produce will be in *INIT*; other users will be free of enemy control, and so their secret keys and nonces will not be in *INIT*.

4.2 Authentication

For an authentication protocol to be correct, we usually require that a user B should not finish running the protocol believing that he has been running with a user A unless A also believes that he has been running the protocol with B. (For a discussion of different forms of authentication, see [18].) Conditions such as this can easily be expressed as trace specifications on **NET**, requiring that no event from a set T has occurred unless another event from a set R has previously occurred.

Definition 1. *For sets $R, T \in \mathcal{M}$, we define the trace specification R* **precedes** *T as*

$$P \textbf{ sat } R \textbf{ precedes } T \Leftrightarrow \forall\, tr \in traces(P) \bullet (tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle)$$

and note that, since all processes are prefix-closed, this guarantees that any occurrence of $t \in T$ in a trace will be preceded by an occurrence of some $r \in R$.

4.3 Rank Functions

Definition 2. *A rank function, as defined in [7], is a function*

$$\rho : \mathcal{M} \rightarrow \{0, 1\}$$

from the message space to the binary-valued set $\{0, 1\}$. In addition, we define

$$\mathcal{M}_{\rho^-} = \{m \in \mathcal{M} \bullet \rho(m) = 0\}$$
$$\mathcal{M}_{\rho^+} = \{m \in \mathcal{M} \bullet \rho(m) = 1\}$$

The point of a rank function is to partition the message space into those messages that the enemy might be able to get hold of, and those messages that will certainly remain out of his grasp. Anything with a rank of one will be something that the enemy might get his hands on; anything of zero rank will be unavailable to him.

4.4 The Central Theorem from [9]

For a process P to *maintain the rank* with respect to a rank function ρ, we mean that it will never transmit any message m with $\rho(m) = 0$ unless it has previously received a message m' with $\rho(m') = 0$. Essentially, this means that the process will never give out anything secret unless it has already received a secret message.

Definition 3. *We say that P* **maintains** *ρ if*

$$P \textbf{ sat } rec.\mathcal{U}.\mathcal{U}.\mathcal{M}_{\rho^-} \textbf{ precedes } trans.\mathcal{U}.\mathcal{U}.\mathcal{M}_{\rho^-}$$

Theorem 1. *If, for sets R and T, there exists a rank function $\rho : \mathcal{M} \rightarrow \{0, 1\}$ satisfying*

1. $\forall\, m \in INIT \bullet \rho(m) = 1$
2. $\forall\, S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet ((\forall\, m' \in S \bullet \rho(m') = 1) \wedge S \vdash m) \Rightarrow \rho(m) = 1$
3. $\forall\, t \in T \bullet \rho(t) = 0$
4. $\forall\, J \in \mathcal{U} \bullet USER_J \;|[\, R \,]|\; Stop$ **maintains** ρ

then ***NET*** **sat** *R* **precedes** *T.*

The proof is omitted; the interested reader is advised to consult [18, 9]. For a demonstration of how Theorem 1 can be used to verify security protocols, see [9].

4.5 Strengthening the Intruder Model

The following protocol is based on Lowe's fixed version [13] of the Needham-Schroeder Public Key Protocol [14]:

$$\begin{array}{lrcl}
\text{Msg 1.} & a \rightarrow b & : & \{a.na\}_{PK(b)} \\
\text{Msg 2.} & b \rightarrow a & : & \{na.nb.b\}_{PK(a)} \\
\text{Msg 3.} & a \rightarrow b & : & \{nb\}_{PK(b)} \\
\text{Msg 4.} & b \rightarrow a & : & nb' \\
\text{Msg 5.} & a \rightarrow b & : & \{SK(a).na'.nb'\}_{PK(a)}
\end{array}$$

The first three messages of this protocol are exactly the same as those of the fixed Needham-Schroeder Public Key Protocol. In Message 4, agent b sends a new nonce nb' to the initiator; in the final message, agent a packages up his own secret key, a new nonce of his choice, and nb', and sends all three to b encrypted under his own public key.

If the RSA weakness is not exploitable, this protocol is secure. Nothing of moment occurs in the last two messages, except for the sending of a long-term secret key; and this secret key is never sent except encrypted in such a way that this key itself is required in order to perform the decryption. (There may be type flaw attacks on this protocol; but such attacks are easily and cheaply avoided by following the implementation scheme recommended in [8].)

Even in the presence of an intruder who can exploit the RSA weakness, the protocol remains secure (see [6] for analysis of this protocol on a small network, in the presence of such an intruder). No matter how many times the intruder learns an encryption from a Message 5, he cannot find two with a known linear relation between them, because the value of na' will be different (and unknown) in each case.

However, it is not possible to find a rank function to prove that the protocol is secure in the presence of such an intruder. Consider the following protocol run, in which agent A initiates a run with the intruder, agent C:

$$\begin{array}{lrcl}
\text{Msg 1.} & A \rightarrow C & : & \{A.N_A\}_{PK(C)} \\
\text{Msg 2.} & C \rightarrow A & : & \{N_A.N_C.C\}_{PK(A)} \\
\text{Msg 3.} & A \rightarrow C & : & \{N_C\}_{PK(C)} \\
\text{Msg 4.} & C \rightarrow A & : & {N_C}' \\
\text{Msg 5.} & A \rightarrow C & : & \{SK(A).{N_A}'.{N_C}'\}_{PK(A)}
\end{array}$$

In the fourth message, the intruder chooses some nonce ${N_C}'$ to send to A, and this is reflected in the message $\{SK(A).{N_A}'.{N_C}'\}_{PK(A)}$ that C receives as a result: A chooses a nonce ${N_A}'$, and sends it to C along with C's nonce and A's secret key, all encrypted under A's public key (the inverse of which C does not hold).

Now, it is clear from the above sequence of messages that we must have

$$\rho(\{SK(A).{N_A}'.{N_C}'\}_{PK(A)}) = 1$$

for A is willing to send this message out at the end of the run: if the process representing A is to maintain the rank, then the final message that it sends out during this protocol run must have a rank of one. But what if the intruder had chosen a different nonce, in place of ${N_C}'$? What if he had instead chosen ${N_C}''$? Then A would have sent $\{SK(A).{N_A}'.{N_C}''\}_{PK(A)}$ instead. We must therefore also have

$$\rho(\{SK(A).{N_A}'.{N_C}''\}_{PK(A)}) = 1$$

—despite the fact that, during the operation of the network, at most one of these messages can be sent out. Agent A will choose a different value for na' in each run, and will never use the same value twice; but because C can send either ${N_C}'$ or ${N_C}''$, each of the above encryptions could be (independently) available to the intruder, and so each of them must be given a rank of one. But now, of course, our new deduction rule tells us that

$$\Big\{\{SK(A).{N_A}'.{N_C}'\}_{PK(A)}, \{SK(A).{N_A}'.{N_C}''\}_{PK(A)}, {N_C}', {N_C}'', PK(A)\Big\} \\ \vdash SK(A)$$

and so A's secret key must have a rank of one—as if the intruder could learn the secret key. He cannot in fact learn it, because he cannot get hold of both of these messages in order to exploit the weakness. If the intruder really could get hold of $SK(A)$, then of course he could masquerade as A as often as he pleased, and the protocol would be broken. The rank functions model cannot, it seems, be extended to deal with the attack on low-exponent RSA because its inability to distinguish mutually exclusive possibles from compossibles wreaks havoc with this type of deduction rule.

The root cause of this is the inequality test in the deduction rule. The failure to distinguish a case in which either of two (but not both) messages can be sent out from a case in which both can be sent out seems not to cause a problem when inequality tests are not permitted in the actions of the agents or the intruder; but with deduction rules such as the RSA rule, where it is advantageous to the intruder to possess two distinct messages of the same type, this failure gives the rank functions model trouble with verifying some correct protocols.

5 Strand Spaces

We next turn to the strand spaces model, as described in [22, 24, 21, 23], and conduct an analogous line of enquiry: can we fit our strengthened intruder into the strand spaces model without encountering similar difficulties?

5.1 Basic Definitions

A *strand* models the actions of an agent on the network, or an atomic action performed by the penetrator. (The *penetrator* in the strand spaces model corresponds to the intruder in the data independence and rank functions models.) A *regular strand* represents a run of the protocol considered from the point of view of one of the agents involved. A *penetrator strand* models an atomic action performed by the penetrator; for instance, concatenating two messages that he knows, or sending a message out over the network. The techniques available to the penetrator in the strand spaces model are essentially the same as those available to the intruder in the rank functions and data independence models.

Definition 4. *We write '*A*' to denote the space of messages that are communicable across the network. An element* $t \in \mathsf{A}$ *is called a* term.

Definition 5. *The set of cryptographic keys is denoted by '*K*'; this set is a subset of* A*.*

Definition 6. *The atomic messages that are not keys form the set* T*. This set is a subset of* A*. The sets* K *and* T *are disjoint.*

Definition 7. *A* strand *is a sequence of signed terms. We write '*$+t$*' to represent transmission of a term* t*, with reception written as '*$-t$*'. A general strand is denoted by '*$\langle \pm t_1, \ldots, \pm t_n \rangle$*'.*

A strand representing an honest agent models the transmissions and receptions involving that agent in a single run of the protocol.

Definition 8. *The signed terms of a strand are called its* nodes*. The* i*th node of a strand* s *is denoted by '*$\langle s, i \rangle$*'.*

5.2 Strand Spaces and Bundles

A collection of strands may be considered as a graph, with two edge types, $\Rightarrow$ and $\rightarrow$, representing respectively consecutive terms on the same strand, and communication between two strands.

Definition 9. *If* n_{i+1} *immediately follows* n_i *on the same strand, then we write '*$n_i \Rightarrow n_{i+1}$*'.*

Definition 10. *If* $n_1 = +a$ *and* $n_2 = -a$ *for some term* $a \in \mathsf{A}$ *then we write '*$n_1 \rightarrow n_2$*'.*

Definition 11. *A* strand space *is then a collection of strands considered as a graph ordered by* $\Rightarrow \cup \rightarrow$*.*

A *bundle* in the strand space model corresponds to a trace of the whole network in the rank functions approach. It is a finite set of strands, ordered by $\Rightarrow \cup \rightarrow$, on which certain conditions are imposed to ensure that

- reception events never occur unless the corresponding transmission event has occurred;
- whenever an agent starts a protocol run, he starts from the beginning of the protocol;
- there is no backwards causation; that is, there are no loops in the graph.

Definition 12. *If* $\mathcal{C} \subseteq (\rightarrow \cup \Rightarrow)$ *is a finite set of edges, and* $\mathcal{N}$ *is the set of nodes that appear on edges in* $\mathcal{C}$*, then* $\mathcal{C}$ *will be called a* bundle *if*

- *whenever* $n_2 \in \mathcal{N}$ *and* n_2 *is a negative node, then there exists a unique* $n_1 \in \mathcal{N}$ *with* $n_1 \rightarrow n_2 \in \mathcal{C}$*;*
- *whenever we have* $n_2 \in \mathcal{N}$ *and* $n_1 \Rightarrow n_2$*, then* $n_1 \Rightarrow n_2 \in \mathcal{C}$*;*
- $\mathcal{C}$ *is acyclic.*

5.3 Penetrator Strands

The analogue of the $\vdash$ relation in the rank functions world is a type of strand known as a *penetrator strand*. A penetrator strand represents a deduction that the penetrator may make under $\vdash$, with different types of penetrator strand corresponding to different types of deduction. In addition, since the penetrator in the strand spaces model has no local state, there will be a type of penetrator strand to represent duplicating a term in order to be able to use it twice; and since every network message that is transmitted is always received by another strand, we introduce one final type of penetrator strand to model hearing and disregarding a message.

Just as the rank functions intruder starts by knowing everything in $INIT$, so the penetrator will have some initial knowledge. However, in the strand spaces model, only the keys known to the penetrator are stated: these keys form the set K_P.

Definition 13. *A* penetrator strand *is one of the following:*

M ***Text message*** $\langle +t \rangle$ *for* $t \in \mathsf{T}$.
F ***Flushing*** $\langle -x \rangle$ *for* $x \in \mathsf{A}$.
T ***Tee*** $\langle -x, +x, +x \rangle$ *for* $x \in \mathsf{A}$.
C ***Concatenation*** $\langle -x, -y, +xy \rangle$ *for* $x, y \in \mathsf{A}$.
S ***Separation*** $\langle -xy, +x, +y \rangle$ *for* $x, y \in \mathsf{A}$.
K ***Key*** $\langle +k \rangle$ *for* $k \in K_P$.
E ***Encryption*** $\langle -k, -x, +\{x\}_k \rangle$ *for* $x \in \mathsf{A}$*,* $k \in \mathsf{K}$.
D ***Decryption*** $\langle -\{x\}_k, -k^{-1}, +x \rangle$ *for* $x \in \mathsf{A}$*,* $k \in \mathsf{K}$.

This definition is parameterised by the set T.

Recall from Definition 12 that for each negative node n_2 in a bundle there must be exactly one positive node n_1 such that $n_1 \rightarrow n_2$; that is, every reception of a message must be matched to exactly one transmission. The purpose of strands of type **F** is to allow the penetrator to 'absorb' transmissions of messages that he does not wish to use; strands of type **T** perform the corresponding rôle of replicating messages that the penetrator wants to transmit more than once.

Recent work on the strand spaces model in [4, 5] has dropped this requirement of matched transmissions and receptions, allowing communications to be sent to zero nodes, one node or many nodes. Consequently, the need for penetrator strands of types **F** and **T** has been abrogated. This slightly simplifies the notation, but does not alter the expressive power of the language. Here, we follow the notation of [21] and keep these types of penetrator strand; but this decision does not affect the analysis.

5.4 Regular Strands

A *regular strand* corresponds to a run of the protocol by an honest agent, or the actions of a trusted server. It will usually be, as is the case in our model, a specific instantiation of a strand template containing free variables. The formal definition of a regular strand is very simple:

Definition 14. *A* regular strand *is any strand that is not a penetrator strand.*

5.5 Subterms and Ideals

We sometimes wish to talk about the subcomponents of a compound message. The concept in the strand spaces model that allows us to do so is that of a *subterm.* The notation '$t_1 \sqsubset t_2$' will imply, informally speaking, that t_1 can be found somewhere in the makeup of t_2); but note that we shall not have $k \sqsubset \{m\}_k$ unless $k \sqsubset m$.

The $\sqsubset$ relation is defined in terms of *ideals.* Ideals in the strand spaces model allow us to talk about all messages containing a particular submessage, when encryption is restricted to a particular set of keys.

Definition 15. *Let $\mathsf{k} \subseteq \mathsf{K}$ be a set of keys, and $I \subseteq \mathsf{A}$ a set of terms. Then we say that I is a* k-ideal *of* A *if*

1. *$hg \in I$ and $gh \in I$ whenever $h \in I$ and $g \in \mathsf{A}$;*
2. *$\{h\}_k \in I$ whenever $h \in I$ and $k \in \mathsf{k}$.*

We write '$I_{\mathsf{k}}[h]$' for the smallest k-ideal containing h. Similarly, if S is a set of terms, then '$I_{\mathsf{k}}[S]$' denotes the smallest k-ideal that includes S.

Definition 16. *We say that $h \sqsubset_{\mathsf{k}} m$ if $m \in I_{\mathsf{k}}[h]$. When $h \sqsubset_{\mathsf{K}} m$, we drop the subscript and write simply '$h \sqsubset m$', and say that h is a* subterm *of m.*

5.6 Honesty

A node is an *entry point* to a set if it transmits a term without having already transmitted or received any term in the set.

Definition 17. *If, for a node n of a strand s, and a set $I \subseteq \mathsf{A}$,*

- *n is a positive node;*
- *the term of n is in I;*
- *no previous node on s has its term in I*

then we say that n is an entry point *to I.*

The idea of an *honest set* is an important one. A set is honest relative to a given bundle if the penetrator can never break into the set except by pure fluke: either he guesses the right nonce (on an **M** strand), or he guesses the right key (on a **K** strand).

Definition 18. *A set $I \subseteq \mathsf{A}$ is* honest *relative to a bundle $\mathcal{C}$ if whenever a node of a penetrator strand s is an entry point to I, then s is a strand of type **M** or type **K**.*

Although honesty is defined for sets in general, it is when the concepts of an honest set and an ideal are conjoined that they are most powerful. The main theorem from [21] gives conditions under which an ideal is honest.

Theorem 2. *Suppose*

1. $\mathcal{C}$ *is a bundle over* A*;*
2. $S \subseteq \mathsf{T} \cup \mathsf{K}$*;*
3. $\mathsf{k} \subseteq \mathsf{K}$*;*
4. $\mathsf{K} \subseteq S \cup \mathsf{k}^{-1}$.

Then $I_{\mathsf{k}}[S]$ *is honest.*

Proof. The proof is omitted; it may be found in [21].

5.7 Strengthening the Intruder

Theorem 2 is the lynchpin of a cutting-edge strand space analysis (see [21] for examples of the theorem in action); and, in fact, strand spaces proofs prior to the introduction of this theorem were also conducted along essentially these lines. The proof of the theorem is by a case-by-case analysis of the possible types of penetrator strand, demonstrating that for each type, with the exceptions of types **M** and **K**, no strand can provide an entry point to $I_{\mathsf{k}}[S]$.

Extending the strand spaces model to cover the RSA attack will involve introducing a new type of penetrator strand

L ***Low-exp RSA*** $\langle -\{a.X.b\}_k, -\{c.X.d\}_k,$
$-k, -a, -b, -c, -d, +X\rangle$ for $a, b, c, d, X \in \mathsf{A}$; $a \neq c$ or $b \neq d$; and $k \in \mathsf{K}$

modelling the penetrator's new-found ability to exploit the weakness. Before we can use this extended model to prove correctness of security protocols even in the presence of our new penetrator, we are required to show that the standard strand spaces results still hold good. In particular, this means that we need to extend the proof of Theorem 2 to include penetrator strands of type **L**.

This, sadly, cannot be done. Let $\mathcal{C}$ be a bundle over A, and let $S = \{X, K^{-1}\}$, with $K \in \mathsf{K}$, $\mathsf{k} = \mathsf{K} \setminus \{K\}$ and $X \in \mathsf{T}$. Now let $a, b, c, d \in \mathsf{T} \setminus S$. The conditions of Theorem 2 are now satisfied, and so we should have that $I_{\mathsf{k}}[S]$ is honest. However, suppose that $\mathcal{C}$ contains the strand

$$\langle -\{a.X.b\}_K, -\{c.X.d\}_K, -K, -a, -b, -c, -d, +X\rangle$$

representing the penetrator's deduction of X by making use of the RSA attack. Now the first two terms lie outside the ideal, because $K \notin \mathsf{k}$; the next five are not in the ideal because they are not contained in S. But the last term, $+X$, is positive, and $X \in I_{\mathsf{k}}[S]$. So this strand provides an entry point for $I_{\mathsf{k}}[S]$, contradicting the definition of honesty.

It appears, then, that there is no simple way to extend the strand spaces model to cover this RSA attack without falsifying the central result used to verify protocols. It is, again, the implicit inequality test that lies at the root. For the concept of an honest ideal captures the notion of the set of messages such that it is undesirable to allow the penetrator to learn any of the messages in the set: if the penetrator can pick up a single message from inside $I_{\mathsf{k}}[S]$, for some suitable set k, he will be able to learn the values of some members of S. Allowing for the RSA weakness would require specifying a set H such that the penetrator should be able to pick up *no more than one* message from H—or, in other words, if he can learn two messages c_1 and c_2 from H, *and* we have that $c_1 \neq c_2$, then he can make use of the attack. Incorporating this into the strand spaces model would require substantial reworking of the basic tools of the trade; and, in any case, it is not at all clear how it could be done.

6 Conclusion

The data independence model cannot be easily adapted to allow the intruder to take advantage of the weakness in low-exponent RSA. We have shown in this paper that both the rank functions method for CSP models, and the use of honest ideals in the strand spaces model, suffer from the same inflexibility, and that it is the implicit inequality checks that are the cause in each case.

The fact that all three models fail to allow for this attack, and all for the same reason, suggests that the mechanisms by which these three approaches to security protocol analysis manage to deal with unbounded networks have more similarities than meet the eye.

Each approach involves identifying an infinite set of messages that can all be treated in the same way. The data independence model requires a data type T such that replacing one member of T for another throughout the system should not defeat any attacks. The rank functions model identifies sets of messages such that the intruder gains no more information from learning all members of the set than he does from learning just one, and so simply keeps track of whether the intruder knows the whole set or none of it. The strand spaces model operates in much the same way: an honest ideal $I_{\mathsf{k}}[S]$ is the set of messages any of which can be used by the penetrator to learn something from the set S; and so the requirement is always to show that none of the messages in the ideal can be picked up by the penetrator. What none of the models can allow for is the idea that allowing the intruder to learn *one* of the messages might be acceptable, whereas to reveal *two* (or more) will result in a breach of security—or, more appropriately, that it is dangerous to allow the intruder to learn two such messages c_1 and c_2 if and only if $c_1 \neq c_2$. We simply cannot deal with the inequality test.

We suspect that Roscoe and Broadfoot's notion of a positive deductive system will accurately limit what can be analysed not just with the data independence model, but with each of the three models considered in this paper. Formalising this idea within the rank functions and strand spaces models, and proving that positive deductive systems do indeed define the scope of what can be verified using these models, is the subject of ongoing research.

We are interested to note that Stoller's approach [20] to identifying bounds on the number of protocol runs required for an attack can be extended to deal with the inequality tests required for the attack on RSA considered here, under certain mild conditions on the nature of the protocol under consideration. This implies that Stoller's method is different at a deeper level from the three approaches considered in this paper, and gives hope that alternative techniques for CSP models and strand spaces might be developed that can handle inequality tests.

Acknowledgement

Thanks are due to Bill Roscoe for his helpful comments on a draft of this paper.

References

[1] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. *Lecture Notes in Computer Science*, 1070, 1996. 163

[2] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983. 164

[3] M. Franklin and M. Reiter. A linear protocol failure for RSA with exponent three. 1995. Presented at the Rump Session of Crypto '95, Santa Barbara, CA. 162

[4] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication Tests. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Security Press, May 2000. 172

[5] Joshua D. Guttman and F. Javier Thayer Fábrega. Protocol Independence through Disjoint Encryption. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 24–34, June 2000. 172

[6] James A. Heather. Exploiting a weakness of RSA. Master's thesis, Oxford University Computing Laboratory, September 1997. 169

[7] James A. Heather. *'Oh! ... Is it really you?'—Using rank functions to verify authentication protocols*. Department of Computer Science, Royal Holloway, University of London, December 2000. 168

[8] James A. Heather, Gavin Lowe, and Steve A. Schneider. How to avoid type flaw attacks on security protocols. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, June 2000. 169

[9] James A. Heather and Steve A. Schneider. Towards automatic verification of authentication protocols on an unbounded network. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, June 2000. 166, 168

[10] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985. 166

[11] Ranko S. Lazić. *A semantic study of data-independence with applications to the mechanical verification of concurrent systems*. PhD thesis, University of Oxford, 1997. 165

[12] Ranko S. Lazić. Theorems for Mechanical Verification of Data-Independent CSP. Technical report, Oxford University Computing Laboratory, 1997. 165

[13] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996. 169

[14] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978. 169

[15] Ron L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 162

[16] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998. 166

[17] A. W. Roscoe and Philippa J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 1999. 162, 164, 165

[18] Steve A. Schneider. Verifying authentication protocols in CSP. *IEEE TSE*, 24(9), September 1998. 166, 167, 168

[19] Steve A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999. 166

[20] Scott D. Stoller. A bound on attacks on authentication protocols. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science (TCS)*, Kluwer, 2002. 175

[21] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. *Proceedings of 11th IEEE Computer Security Foundations Workshop*, June 1998. 170, 172, 173, 174

[22] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, May 1998. 170

[23] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 72–82, June 1999. 170

[24] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999. 170

TINMAN: A Resource Bound Security Checking System for Mobile Code*

Aloysius K. Mok and Weijiang Yu

Department of Computer Sciences, University of Texas at Austin
Austin, Texas 78712 USA
{mok,wjyu}@cs.utexas.edu

Abstract. Resource security pertains to the prevention of unauthorized usage of system resources that may not directly cause corruption or leakage of information. A common breach of resource security is the class of attacks called DoS (Denial of Service) attacks. This paper proposes an architecture called TINMAN whose goal is to efficiently and effectively safeguard resource security for mobile source code written in C. We couple resource usage checks at the programming language level and at the run-time system level. This is achieved by the generation of a resource skeleton from source code. This resource skeleton abstracts the resource consumption behavior of the program which is validated by means of a resource usage certificate that is derived from proof generation. TINMAN uses resource-usage checking tools to generate proof obligations required of the resource usage certificate and provides full coverage by monitoring any essential property not guaranteed by the certificates. We shall describe the architecture of TINMAN and give some experimental results of the preliminary TINMAN implementation.

1 Introduction

Mobile codes are becoming widely deployed to make wide-area networks extensible and flexible by adding functionality and programmability into the nodes of the network. Mobile code which is written on one computer and executed on another, can be transmitted in the form of either binary or source code. The former includes Java Applet, ActiveX, or executables for specific platforms. The latter includes applications written in high-level languages like C, scripts such as shell files or JavaScript embedded in HTML files and specific languages designed for *active network* [1] such as PLAN [2] and Smart Packets [3].

Not surprisingly, there is increasing demand on host systems to provide security mechanisms for shielding users from the damages caused by executing untrustworthy mobile code. One of the serious concerns of mobile code security is *resource bound security*. Resource bound security pertains to resource consumption limits and the prevention of unauthorized use or access to system

* This work is supported in part by a grant from the US Office of Naval Research under grant number N00014-99-1-0402 and N00014-98-1-0704.

resources that may not directly cause corruption or leakage of protected information. Failure to properly delimit resource consumption by untrusted mobile code may deny legitimate users access to system resources, as is in the case of Denial Of Service (DoS) attacks.

In this paper, we propose TINMAN, a platform-independent architecture whose goal is to efficiently and effectively perform resource bound security checks on mobile code. Although the TINMAN architecture is not restricted to mobile code developed in a particular language, we analyze mobile C code in our prototype system since many mobile programs are written in C and lack security checks. For example, a user might try to extend his system software with an external software component, or download and install open source software and applications that are often in the form of C source code, from both trusted and untrusted sources. Thus, we want to validate the TINMAN architecture first for C code; we believe that the TINMAN architecture should be applicable directly to the mobile code in scripts and byte-code as well.

Most of the work on mobile code security has not addressed resource bound security or is not effective when it is applied to a general-purpose languages such as C. For example, Java Sandbox [5], Smart Packets [3] and PLAN for PLANet strive to prevent abuse by mobile code at the programming language level by limiting a user's ability to gain access to system resources through language design. This usually comes at the price of reduced expressiveness of the programming language and may turn off programmers who are unwilling to adopt a restrictive language just for their mobile applications. Also, these approaches cannot guarantee the resource bound consumption which is essential to the hosting systems; for instance, they cannot detect buggy or malicious code that may fall into an infinite loop and use all the CPU cycles and possibly all system memory. An alternative approach to attaining resource bound security is to enforce certain *security policies* that restrict the resource utilization of mobile code execution at the run-time system level. Run-time checking of resource usage alleviates the drawback of strict reliance on language-level mechanisms but it is costly, and certain methods of access control do not apply since mobile code is not stored on the computer it is executed on. In addition, not all these systems provide resource utilization prediction and thus resource-usage security policies are difficult to enforce.

We believe that in order to ensure resource bound security for mobile source code, every hosting system should be endowed with a capability to accept usage specification (via policy-based resource bounds) and to monitor (via policy enforcement) the execution of untrusted mobile programs. This policy specification to enforcement linkage should be established in a way that cannot be spoofed, or else the system should not be trusted to transport mobile code. Our approach is to endow mobile source code with a verifiable certificate that specifies its resource consumption behavior. The key idea underlying the TINMAN architecture, however, is the recognition that exact resource usage cannot in general be derived *a priori* by compile-time static analysis (unless, say, the halting problem is decidable, which it is not) and that a combination of compile-time (off-line) and run-time (on-line) techniques is needed to link policy specification to enforcement. The real hard issue is to determine the relative roles of off-line vs. on-line analysis. To answer this question, we adopt the following principle:

"Check the verifiable. Monitor the unverifiable."

TINMAN applies the above principle. Resource bound is analyzed and verified off-line from user-supplied information and security policy, to the extent that it is practical to do so, and security enforcement is performed by coupling language-level and run-time system mechanisms to monitor what cannot be established by off-line analysis. The off-line and run-time mechanisms together provide complete coverage and guarantee resource bound security. TINMAN advances the state of the art by pushing the limit in the automation of fine-grain resource consumption security checks, by deploying a suite of tools for resource utilization prediction, certificate generation and validation, and run-time event matching.

In this paper, we focus on the system design and implementation of the TINMAN architecture. The next section describes the methodology of system design. The architecture of TINMAN is given in Section 3, followed by some details of the off-line checker and on-line checker. Some experimental results are given in Section 7. Section 8 compares TINMAN with related work. Concluding remarks and future work are in Section 9.

2 Methodology

As noted in the previous section, there are two aspects to the resource security problem: policy and enforcement. *Resource security policy* establishes resource usage constraints that mobile programs must satisfy. In TINMAN, the security policy to be satisfied by mobile program is given by a specification in three parts: (1) resource usage for each service that may be called by a program and provided by a hosting system; (2) resource limit for each program; (3) a proof system consisting of axioms and inference rules for the interpretation of the specification.

Resource security enforcement prevents a mobile program from violating the resource security policy. It pertains to the authorization of resource usage and the limitation of actual resource usage by a program. Enforcement may be performed dynamically, by monitoring the real-time resource utilization of a mobile program at run time. Complete reliance on dynamic enforcement may be too expensive in general as to be practical. An alternative way to perform enforcement is to check the certification of resource bounds for a certified program, similar to Proof Carrying Code (PCC) [6] and Typed Assembly Language (TAL) [7].

Ideally, if certification is trustworthy, the code is considered resource usage safe. The system may then grant resources to the program to complete its execution, and there is no run-time checking (as in PCC). This is in general not possible for resource bound security since exact resource usage is usually determined dynamically and cannot in general be derived by compile-time static analysis. Our approach is to a combine off-line verification and on-line monitoring.

TINMAN deploys an off-line analysis tool that attempts to derive tight resource usage bounds for user-supplied codes that may be loaded and run on hosting systems. In an ideal world, the off-line analyzer would be able to derive an exact resource usage bound and output a correctness proof of the bound which constitutes the *usage certificate*. Short of getting an exact bound, a tight bound may be obtained interactively with the programmer giving help in the form of assertions about the program's behavior.

An on-line analysis tool resides on the code recipient where the code is run. We note that if verification is possible at compile-time, all the on-line analyzer needs to do is to check the usage certificate, and dispense with monitoring. Since checking a proof or validating a usage certificate via, say, a trusted certification authority is in general much easier than deriving the proof from scratch (just as it may take exponential time to solve an NP-complete problem but only polynomial time to check the correctness of a solution), the run-time overhead is relatively small. In the case not all verification conditions can be verified at compile-time or if programmer input is required, the off-line analyzer also outputs all the assertions, and the on-line checker will automatically monitor the validity of the assertions at run time. In this way, a network node determines with certainty that a piece of mobile code is compliant with the resource security policy.

3 System Architecture of TINMAN

Figure 1 shows the TINMAN architecture and the dependencies among the system's components. We now give a high-level description of each component and explain how they fit together to perform resource security check for mobile code.

3.1 Off-Line Checker

The goals of the off-line checker are to provide programmers with a tool for resource bound prediction and usage certificate generation for their mobile program. As shown in Figure 2, resource usage prediction involves *timing analysis*

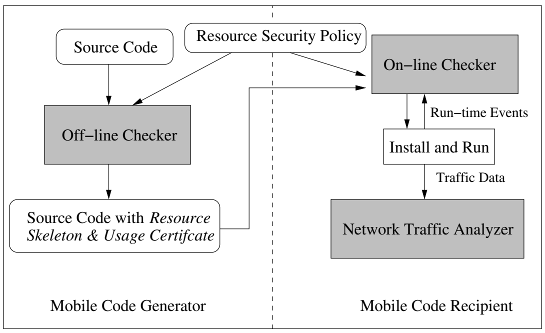


Fig. 1. TINMAN Architecture

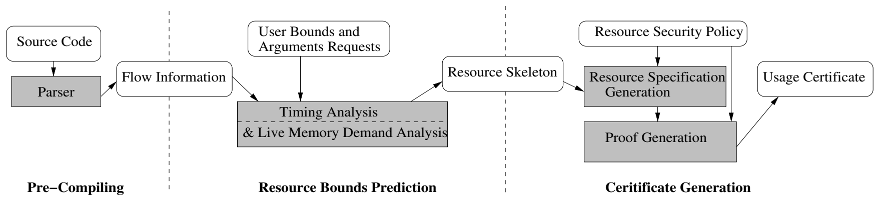


Fig. 2. Overview of Static Resource Bound Prediction and Verification

and *live-memory demand analysis*. The output is a set of annotations and assertions we call a *resource skeleton*. The resource skeleton is parameterized such that the exact values of the bounds can be determined at the code recipient site. A *resource specification generation* component automatically translates the resource skeleton into a specification consisting of a group of predicates that will be verified by a *proof generation* module to produce a usage certificate.

Once mobile code is analyzed by the off-line checker, it is ready to be sent to the recipient site.

3.2 On-Line Checker

The on-line checker validates the annotations inserted by the off-line checker, and detects any violation against the security policy by the imported mobile code before or during its execution. A typical on-line checker session involves resource usage bound calculation, resource skeleton verification, usage certificate validation and run-time assertion monitoring. The on-line checker will be discussed further in Section 6.

3.3 Network Traffic Analyzer

Unlike the CPU and memory, network resource abuse in general may affect entities outside of a host executing the mobile code and as such cannot be readily detected by checking resource usage limits on the system where the mobile programs run. It is difficult to statically check exact network bandwidth usage at the language level because of the dependency on protocol implementation. TINMAN instead prevents such attacks at the sender by performing run-time network traffic analysis in addition to static checks on network-related service routines. Run-time network resource checking is not the subject of focus in this paper, and interested readers are referred to [8] for more details.

4 Resource Bound Prediction

Resource bound prediction is done at the source code level. The off-line checker modifies the Broadway Compiler [9], a source-to-source translator for ANSI C, to parse the source code and output flow information to resource analysis modules. To strengthen the resource prediction results, the off-line checker performs type-checking to assure type safety. We make the following assumptions about the mobile code discussed throughout this paper: No cast, pointer arithmetic, address operation and recursive calls are used. All variables must be declared with types. Variables including structure fields must be initialized before being used. In addition, all array subscripting operations are checked for bound violation. The type-checking is done statically. A failed type-check will result in the termination of the off-line checker in which case no resource skeleton and usage certificate will be generated. We shall formalize the semantics of this subset of ANSI C in section 5.

4.1 Timing Analysis

Timing analysis attempts to determine the worst-case execution time (WCET) of a program. Prediction of WCETs is an extensive research area, and substantial progress has been made over the last decade. Most practical WCET techniques, however, cannot be directly applied here since they are tied to particular target architectures and do not take mobile code into consideration. The off-line checker addresses this issue by combining a source level timing schema approach with a policy-based, parameterized WCET computation method.

The basic idea of the timing schema approach [10] is that the execution time is determined by basic blocks and control structures of a program. For example, the execution time of the assignment statement $A : a \ = b - c$; can be computed as $T(A) = T(b) + T(-) + T(c) + T(a) + T(=)$, where T(X) denotes the execution time of the item X, referred to as an *atomic block*, *i.e.*, basic component in the abstract syntax tree of a program. The execution time for the control statements is computed similarly,

T($if(exp0)\ S1; else\ S2$) = T(exp0) + T(if) + max(T(S1), T(S2)).

T($for(i = 0; i < N; i++)\ stmt;$) = T(i) + T(=) + (N + 1) · (T(i) + T(<) + N · (T(stmt) + T(for) + T(i) + T(++)).

assuming i is not changed in the loop's *stmt*. The timing of compound statement is calculated recursively based on the simple statements.

We note that the above approach fails when a program contains loops whose iteration bounds cannot be directly obtained, or when system service calls are invoked; the latter is common for a mobile application. The off-line checker uses a number of techniques to alleviate these problems.

Loops

For loops, a pre-compiling analysis of the source code discovers constant loop bounds or handles dependencies between loop iteration variables of the nested loops. For example, consider the loop

$while(m < n)\ \{S1; n = n + I0; S2\}$

If both m and n are constants at the entry point, the increment/decrement of n

(i.e. $I0$) is a constant and dominates the next loop iteration, and the number of loop iterations can be statically determined. Therefore, the accurate loop bound can be calculated using constant propagation and the dominance relationship[1] analysis. For a nested loop where the number of iterations in the inner loop depends on the value of the index in the outer loop, we rely on techniques similar to [12] to give a tight prediction on loop iterations.

For loops where loop bounds are difficult to deduce automatically (or even do not exist as in an infinite loop) but are straightforward for a programmer to deduce, our tool will ask the programmer to input the asserted loop bound which will be monitored.

System Service Calls

The execution time of system service calls in a program is not known without information of the remote system where the program runs. Our approach is to specify the resource usage of a service call in a policy rule in the form of pre and post conditions defined in a formal logic (to be described in detail in section 5). The resource consumed by a service is parameterized given the ranges of its arguments. Exact values are determined at the remote site. This approach is flexible since the policy is configurable for a specific platform and run-time environment. As a result, only parameter information and the system call itself are need in the resource skeleton by the off-line checker.

4.2 Live Memory Demand Analysis

Accurate memory allocation prediction is a non-trivial task, and usually requires considerable cost [13]. The memory used by a mobile code consists of three spaces: stack space, dynamically-loaded code, and heap space. Stack space consumption is largely due to recursive calls which are currently prohibited in the current prototype of TINMAN. On the other hand, the size of a piece of mobile code is generally small and it is relatively trivial to compute the heap memory size of the mobile code. The heap space, however, is exploited by dangling pointers and may be allocated by malicious programmers. TINMAN analyzes the heap space allocation, referred to as the *live memory demand* at the language level. Considering the possible actions of malicious codes on memory, our efforts are focused on memory allocation requests in a program.

Consider for example, the memory demand of an assignment statement S:

$$head = (Node*)malloc(sizeof(Node) * exp);$$

is M(S) = sizeof(Node) * Value(*exp*). The sizeof(Node) can be determined at the entry point of S, while *exp* may not. In addition, the memory demand computation may be complicated if the *malloc*() statement is in a loop, and/or the paired free statement is not properly given. Taking all these into account, we adopt a general approach for live memory demand prediction in the following steps: (1) Analysis of memory allocation statements: All memory allocation statements

[1] A *dominates* B iff all paths from the start node to B intersects A [11].

are identified such as *malloc*(), *calloc*(), *memalign*(). The size (in bytes) of maximum memory request is either calculated automatically using flow analysis or provided by the programmer if it cannot be statically bounded, for example, the value of *exp* in S depends on program input data. (2) Path analysis of memory freed: The live memory demand may be overestimated if we simply sum all memory allocation requests. We check the dominance relationship between pairs of memory allocation and free statements, and within the same basic blocks or compound statements to tighten the live memory bound (3) Call chain analysis: The demanded memory size is increased throughout the call chain. Therefore, we compute recursively the memory demand at the points of interest, such as loops and procedure calls.

4.3 Resource Skeleton

The timing and memory bound information obtained by either automatic analysis or from programmer input need to be maintained for further use. We use a *resource skeleton* to annotate the information to help bound the resources. Basically, a resource skeleton can be viewed as an abstraction of a program in regard to its resource consumption. The resource skeleton of a program also makes it possible for the on-line checker which is invoked by the code recipient to detect violations against the resource security policy.

The Resource skeleton is created along with the construction of the flow graph of a program derived from its syntax tree. A node in a flow graph is called a *task*. Task types include basic blocks, conditional statements, loops, user-defined procedure calls and system service calls. The tasks of basic blocks and service calls are *primitive tasks*. The flow graph is hierarchical which means a program is a sequence of tasks, and each task (except primitive tasks) can be expended into a flow graph. To reduce unnecessary annotations and proof generation based on these annotations, a sequence of primitive tasks are combined in the same resource skeleton. We illustrate the construction of resource skeleton with an example, shown in Figure 3.

```
B1:  /* look up forwarding record */
     n = getrecord(group, sender);
     m = &n;
     m–>time = n.time – 10;
C2:  /* must find it to continue */
     if (m–>nodes != NULL){
L3:    /* send a copy every way */
       for (i = 0; i < m–>length; i++)
B4:       routefornode(&n, m–>nodes[i]);
     }
     else {
B5:     m–>node = (Node*)malloc(sizeof(Node)*10);
       /* or deliver to application */
        delivertoapp(&n, dpt);
     }
```

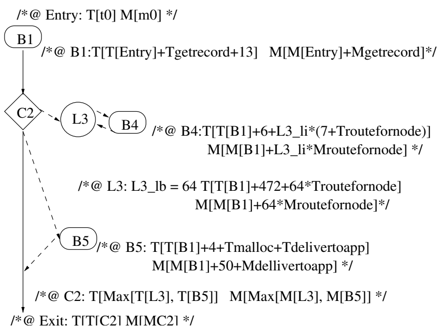

/*@ Entry: T[t0] M[m0] */
B1
/*@ B1:T[T[Entry]+Tgetrecord+13]   M[M[Entry]+Mgetrecord]*/
C2
L3
B4
/*@ B4:T[T[B1]+6+L3_li*(7+Troutefornode)] M[M[B1]+L3_li*Mroutefornode] */
/*@ L3: L3_lb = 64 T[T[B1]+472+64*Troutefornode] M[M[B1]+64*Mroutefornode]*/
B5
/*@ B5: T[T[B1]+4+Tmalloc+Tdelivertoapp] M[M[B1]+50+Mdellivertoapp] */
/*@ C2: T[Max[T[L3], T[B5]]   M[Max[M[L3], M[B5]] */
/*@ Exit: T[T[C2] M[MC2] */

Fig. 3. Example of Resource Skeleton

The task in the flow graph is labeled with its type (e.g. L = Loop task) followed by a global counter value. The solid line marks the execution sequence of related tasks while a broken line means a task expansion. Resource annotation for each task is enclosed in "/∗@ . . . ∗/" which can be identified by the on-line checker and be ignored by a standard compiler. T[*exp*] and M[*exp*] represent the time and memory-demand bounds of the corresponding task.

For example, the execution time of C2 is the maximum execution time of T[L3] and T[B5], where T[L3] and T[B5] can be recursively computed from the resource annotations for L3 and B5. The loop bound for L3 is provided by the programmer. We note that, for example, the CPU and memory usage bound for B1 are given only with the name of the service call ($Tgetrecord$). The actual bound value calculation is performed at the remote site with instantiated resource security policies.

5 Generating Usage Certificate

The mobile program is transported with a usage certificate since a code recipient cannot trust the resource skeleton which may be corrupted. Our basic strategy to establish the correctness of certificates is to use an assertional programming logic such as Hoare Logic. Our approach is to translate the resource skeleton into a resource safety specification in a formal logic, and generate a certificate by checking for compliance with security policy. In this section, we first introduce the resource specification, using extended Hoare triples. Next, we present a proof system by formulating the formal semantics of a subset of the C language (mentioned in section 4) in terms of a set of axioms and inference rules. The program correctness on satisfying a specification can be proved within the proof system.

5.1 Resource Specification

For a task T, a *resource specification* for T is an extended Hoare triple {P} T {Q}, where assertion P is the precondition, and assertion Q is the postcondition which holds if P is true and T terminates. An assertion is a first-order logic formula which is typically a conjunction of predicates over program variables, time and memory usage and terminational judgment. In order to translate a resource skeleton annotation for task T into an assertion, we first define the type $Time$ to be the nonnegative reals and $Memory$ to be the nonnegative integers as respectively domains of time and memory size values. Three special variables: *now* of type $Time$, *mem* of type $Memory$, and *terminate* of type boolean are introduced. These variables denote the moment, the allocated memory and the termination before, if they are in P, or after, if in Q, the execution of T, respectively. The initial value of *now* is *t0*, and *mem m0*. With this terminology, the resource skeleton can be represented as logical specifications. For example, the specification for task B1 in Figure 3 is

$\{\mathbf{now} = t0 \wedge \mathbf{mem} = m0 \wedge \mathbf{terminate}\}$
B1
$\{\mathbf{now} <= t0 + Tgetrecord + 13 \wedge$
$\mathbf{mem} <= m0 + Mgetrecord \wedge \mathbf{terminate}\}$
And the specification for loop task L3 is as follows. Note that *L3_lb* is the user-provided loop bound.
$\{L3\_lb = 64 \wedge \mathbf{now} <= t0 + Tgetrecord + 17 \wedge\ \mathbf{mem} <= m0 + Mgetrecord \wedge \mathbf{terminate}\}$
L3
$\{\mathbf{now} <= t0 + Tgetrecord + 472 + 64 \cdot Troutefornode \wedge$
$\mathbf{mem} <= m0 + Mgetrecord + 64 \cdot Mroutefornode \wedge \mathbf{terminate}\}$
It should be pointed out that the specification for a service call contains a precondition, if any, over the ranges of its arguments and expressions of its running time and memory usage. It is published as part of the security policy.

5.2 Proof System

Similar to Jozef Hooman's framework [14] for sequential programs, we construct a proof system for resource specification by formalizing programming constructs, or tasks in our cases. The tasks are axiomatized by inference rules and axioms. The rules in the framework are proved independently and published as part of security resource policies as well.

Clearly, the formalization of resource specification and the proof system requires mechanical support. TINMAN uses the Prototype Verification System (PVS) [15] to implement its logical framework. In order to formulate resource specification into the PVS specification language, our approach is to identify programs with their semantics, i.e., the relations on states. A *state* contains a mapping of program variables to values, current time, allocated memory, and termination indicator. We developed an extended and modified version of construction rules defined in [14]. All tasks are defined with regard to their resource specifications. For example, the task B1 in the previous example is defined in PVS as follows,

```
P0 : [State->bool] =
     ((LAMBDA s : state) : now(s) = t0
     AND mem(s) = m0 AND terminate(s) )
srvc1: program = SRVC(Tgetrecord, Mgetrecord)
bb1  : program = BB(13);
B1   : program = seq(srvc1, bb1);
Q0 : [State->bool] = (LAMBDA s :
     now(s) = t0 + Tgetrecord + 13 AND
     mem(s) = m0 + Mgetrecord AND
     terminate(s))
{P0}B1{Q0} : THEOREM
  ((FORALL s0, s1 : state):
     P0(s0) AND B1(s0, s1) IMPLIES Q0(s1))
```

The complete set of axioms and inferences rules and program constructs definition can be found in [8].

5.3 Proof Generation

A PVS specification for resource skeleton is obtained by applying the program construction rules. To prove the specifications, one would like to construct proofs interactively using the PVS prover system. For a mobile program, however, we aim at generating a proof or certificate as automatically as possible. PVS provides a mechanism for automatic theorem proving by composing proof steps into proof strategies. We note that the specification for a different type of program constructs may require different proof strategies.

For primitive constructs like basic block tasks and service call tasks, we have defined a strategy that performs a sequence of built-in proof steps. In this strategy, the theory definition including the task definitions and assertions is expended by automatic rewrite rules, and then Skolemization and Decision procedure are invoked repeatedly until the theory is proved. For example, the theorem {P0}B1{Q0} in the previous example is proved using this strategy by first auto-rewriting P0, Q0, B1, seq, srvc1, and bb1, and then repeatedly invoking prover commands `ASSERT` and `SKOSIMP*` until the theorem is proved.

The proof strategy for proving a choice task specification, say $\{P\}$**if** b **then** $T1$ **else** $T2$ $\{Q\}$, is more complicated than the primitive tasks. The **Rule 2** for choice tasks in the proof system illustrates the general steps of the strategy. Briefly, we need first to prove the two corollaries, $\{P \wedge b\}$ $COND(tb, mb)$; $T1$ $\{Q\}$, and $\{P \wedge \neg b\}$ $COND(tb, mb)$; $T2$ $\{Q\}$ by applying some other strategies and then invoke the prover command `LEMMA Rule 2` and the quantifier rule. The construction of those strategies and other proof steps in the strategy for choice tasks are non-trivial, and will not be discussed any further here.

The strategy for sequential tasks and loop tasks are constructed similarly in consideration of the corresponding rules in proof system. The strategy for loop task, however, involves a loop invariant and an assertion that holds if the loop terminates. In order to generate them automatically, we simplify a loop invariant by introducing an auxiliary loop index, say li for a loop L, where $li \in [0, loopbound_L]$ and is increased by 1 in each loop iteration. The loop invariant is constructed with li and the specification (precondition and postcondition) of task L, since we only need to assure the correctness of the resource bound, not what the program actually does.

Using these proof strategies, the resource specification for an annotated program is proved automatically. PVS outputs the proof onto a text file which constitutes the usage certificate for the program. However, due to the large size of the proofs written in the built-in PVS specification logic, we further shorten the detailed usage certificate by only keeping the strategies and related parameters required to produce the proof. We refer the shortened usage certificate as a *certificate skeleton*. Finally, the annotated mobile program only requires the certificate skeleton to be transferred to the remote site.

6 On-Line Checker

The on-line checker performs limited static verification by validating the supplied resource skeleton and usage certificate, and it detects any violation against security policy on resource utilization limits.

Figure 4 shows the major steps of static on-line verification. We note that the calculation of actual resource bound, given instantiated policies on service calls is performed before the validation of the resource skeleton. It enables detection of any violation against a resource usage limit at an early stage since if there is a violation, the mobile program will not be trusted, and no further verification is warranted.

The purpose of the resource skeleton validation is to check for consistency between the source code and the resource skeleton. Annotations are inserted at the appropriate points. Specifically, the tasks, programmer-provided information if necessary for loop bound, memory allocations and service call arguments are annotated, and there are no further manually inserted annotations.

The resource specification generation is the same as that in the off-line checker. It outputs a specification consisting of predicates on the resource skeleton. The full usage certificate is restored from the certificate skeleton. The proof checker verifies the supplied usage certificate to conform with the specifications within the PVS system. In our implementation, the proof checking is as simple as a validation run of PVS in a *batch mode* which automatically reruns all proofs in the usage certificate. An invalid usage certificate will generate errors which can be caught by the proof checker by examining the run log file.

After the verification of the resource skeleton and validation of usage certificate, the only untrusted part are the user-provided assertions on loop bounds and the ranges of function arguments. The on-line checker needs to monitor these values at run-time. In order to do this, the on-line checker translates them into related assertions for checking the range of the values of untrusted data. A run-time exception will be raised if any violation of policy is detected. Dynamic resource utilization monitoring is a tried concept used by much previous

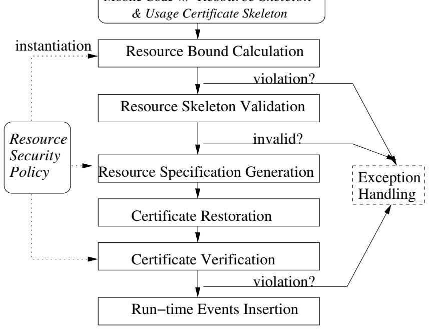

Mobile Code w/ *Resource Skeleton*
& Usage Certificate Skeleton
instantiation
Resource Bound Calculation
violation?
Resource Skeleton Validation
invalid?
Resource Security Policy
Resource Specification Generation
Exception Handling
Certificate Restoration
Certificate Verification
violation?
Run−time Events Insertion

Fig. 4. Structure of Static On-line Verification

work. Our approach avoids much of the dynamic resource utilization monitoring since the resource bound safety is guaranteed by static verification, and run-time checking is required only on a few programmer-provided annotations.

7 Experimental Results

In this section, we present some experimental results obtained by using our tools. We have translated the following two mobile programs used by other research groups [1, 4] into C. The multicast_subscribe installs forwarding pointers in each router (member of a group) it traverses so it can receive messages sent to the group. It has six services calls, one loop and six tasks. The multicast_data simply routes itself along a distribution tree, and it has five services calls, two loops and eight tasks.

We first measured the code size augments due to insertion of annotations and the usage certificate. The results are shown in Table 1. We note that the increased size depends on the number of tasks in a program, but is not directly related to the size of the program. This is because, for example, multicast_data has more complicated control structure and more tasks. It also explains the reason that multicast_data has a larger usage certificate. We also observed that the certificate size is significantly decreased by up to 94.3% by using a certificate skeleton.

Table 2 shows the cost of off-line certificate generation and on-line annotation verification and certificate checking. The certificates of both example programs are generated completely automatically. However, off-line certificate validation and on-line certificate check result in an order of magnitude slower than cer-

Table 1. Code Size with Annotations (in bytes)

Program	multicast _subscribe	multicast _data
Original Size	1508	1113
Resource Skeleton	316	462
Certificate Skeleton	569	1087
Full Certificate	9963	14595
Increase(%)	58.7	139.2

Table 2. Cost of off-line checker and on-line checker

Program	multicast _subscribe	multicast _data
Specification Generation	36.5ms	58.5ms
Off-line Cert. Construction + Validation	0.83s + 11.06s	1.45s + 23.85s
On-line Anno. Verification	62.9ms	51.1ms
On-line Cert. Check	11.09s	22.72s

Table 3. Resource Utilization Measurement

Program	Predicted WCET	Observed WCET (w/o ann.)	Observed WCET (w/ ann.)	Pess.	Predicted Mem(B)	Actual Mem(B)	Pess.
multicast_subscribe	74.08ms	57.56ms	81.73ms	29.8%	932	804	15.9%
multicast_data	1456ms	1363ms	1503ms	6.8%	12800	8192	56.3%

tificate construction. The overhead is two-fold. First, a certificate consists of PVS rules that are interpreted by the PVS prover interactively. Second, current proof strategy for loops and compound choice tasks involves catch-all prover commands like `GRIND` which is timing consuming. Table 2 also indicates that annotation verification time (e.g resource usage calculation and annotation check) is quite small and negligible compared with certificate check time.

Table 3 shows the timing and memory analysis results of the programs. The loop bounds of multicast_data are automatically calculated while the bound of multicast_subscribe are given by a programmer. The pessimism of multicast_data is only 6.8% because we have obtained a tight loop bound. On the contrary, the actual loop counts in multicast_subscribe depend on the conditions of inner break statements. Therefore, it is very conservative and is off by 29.8%. The overhead of running a mobile program with annotations comes from run-time monitoring of programmer-provided information and the communication between the annotated program and the on-line checker during the execution. All of them have small monitoring overheads.

The pessimism of memory analysis is caused by the actual execution path that affects memory allocation statements. For example, in multicast_data, the *malloc*() inside a loop is only executed upon satisfaction of some condition, and its total execution time is also decided by the loop bound. The experiments show, however, the actual memory allocated does not exceed the predicted live memory demand that is essential to our goal of resource security.

8 Related Work

Previous resource usage safety efforts for mobile code usually enforce the security policy in the run-time system to limit the resource utilization of mobile code. Smart Packets [3] checks the CPU and memory usage of active packets written in Sprocket and enforces limits on the number of instructions executed, amount of memory used, and access to MIB variables. The KeyNote in PLANet has a similar mechanism [16]. These active network systems do not provide tools to do source-code-level checking concerning resource usage, and have significant restrictions on languages features and thus limit the expressiveness of mobile code. Some researchers have developed extensions for more expressive security policies. For instance, Naccio [17] specifies security policies for Java and Win32 using a specification language. Java programs are transformed to call wrapper functions instead of the original library code in order to enforce safety policies.

In PCC, a code producer creates a formal safety proof to prove adherence to the safety rules that guarantee safe behavior of programs. A remote host depends on proof validation techniques to check that the proof is valid so that the foreign code is safe to execute. The difficulty of generating proofs for large programs and more interesting policies are the difficulties in the application of PCC. In practice, PCC has been used to verify low-level safety properties, and it does not address the resource bound security problems in terms of resource behavior prediction and program termination. Unlike PCC, TINMAN concentrates on resource security assurance in high-level to prevent DoS-like attacks and buggy or malicious codes with infinite loops or improper arguments to services calls that are not addressed by previous work. The resource security policy is flexible and configurable at code recipient site. In addition, the proof system constructed using the PVS system makes it easier for proof construction and validation for more complicated mobile applications.

9 Conclusions

In this paper we have presented TINMAN, a resource bound security checking system for mobile code. The system detects malicious mobile source code that, once installed and executed, may consume inordinate amounts of resources such as CPU, memory and network bandwidth, as is common in DoS (Denial-Of-Service) attacks.

TINMAN provides multiple levels of protection on resource security at both compile time and run time, at both the source-code level as well as run-time system level. It has been implemented by a set of tools that support resource bound prediction and certificate generation and validation. TINMAN exploits programmer input but does not depend on it for ensuring resource security; incorrect programmer input about resource bounds will be checked and detected against the resource skeleton associated with the mobile code, and the usage certificate is validated against given resource security policies. An on-line checker tool is used to detect malicious modification of resource bound annotations and certificate validation. This enables any violation of resource utilization with respect to security policy to be detected as early as possible before the execution of the mobile code. Together, the off-line and on-line checkers provide complete coverage, following the guideline of proving what can be verified and monitoring what cannot be verified.

In this paper, we consider mobile programs written in C with active network benchmarks. However, our framework is extensible and applicable, in an even simpler style, for other programming languages for mobile applications such as Javascript of which source codes are usually embedded in a HTML file. By gaining experience in TINMAN, we shall hopefully be able to customize the framework for some version of byte-code for C. Our plan is to use the script from the verifier session as the usage certificate so that the proof can be checked on-line efficiently.

References

[1] Wetherall, D., Guttag, J., Tennenhouse, D.: ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. IEEE OPENARCH (1998) 117–129 178, 190

[2] Hicks, M. W., Kakkar, P., Moore, J. T., Gunter, C. A., Nettles, S.: PLAN: A Packet Language for Active Networks. International Conference on Functional Programming (1998) 86–93 178

[3] Schwartz, B., Jackson, A. W., Strayer, W. T., Zhou W., Rockwell, D., Partridge, C.: Smart Packets: Applying Active Networks to Network Management. ACM Transactions on Computer Systems 18:1 (2000) 67–88 178, 179, 191

[4] Kornblum, J., Raz, D., Shavitt, Y.: The Active Process Interaction with its Environment. Lucent Technology (1999) 190

[5] Fritzinger, J. S., Mueller, M.: Java Security. Sun Microsystems white paper (1996) 179

[6] Necula, G. C.: Proof-Carrying Code. POPL'97 (1997) 106–119 180

[7] Morrisett, G., Walker, D., Crary, K., Glew, N.: From System F to Typed Assembly Language. IEEE Symposium on Principles of Programming Languages (1998) 180

[8] TINMAN Project. `http://www.cs.utexas.edu/wjyu/tinman` 182, 187

[9] Guyer, S. Z., Jiménez, D. A., Lin, C.: Using C-Breeze. Department of Computer Sciences, The University of Texas, (2002). `http://www.cs.utexas.edu/users/lin/cbz` 182

[10] Park, C. Y., Shaw, A. C.: Experiments with a program timing tool based on source-level timing schema. Computer J 25:5 (1991) 48–57 183

[11] Aho, A. V., Sethi, R., Ullman, J. D.: Compilers: Principles, Techniques, and Tools. Addison Wesley (1986) 184

[12] Healy, C., Sjdin, M., Rustagi, V., Whalley, D.: Bounding Loop Iterations for Timing Analysis. IEEE Real-Time Applications Symposium (RTAS'98) (1998) 12–21 184

[13] Unnikrishnan, L., Stoller, S. D., Liu, Y. A.: Automatic accurate stack space and heap space analysis for high-level languages. Computer Science Department, Indiana University TR 538 (2000) 184

[14] Hooman, J.: Correctness of Real Time Systems by Construction. FTRTFTS: Formal Techniques in Real-Time and Fault-Tolerant Systems LNCS 863, Springer-Verlag, (1994) 19–40 187

[15] Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. 11th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Springer Verlag (1992) 748–752 187

[16] Alexander, D. S., Anagnostakis, K. G., Arbaugh, W. A., Keromytis, A. D., Smith, J. M.: The Price of Safety in an Active Network. University of Pennsylvania MS-CIS-99-02 (1999) 191

[17] Evans, D., Twyman, A.: Flexible Policy-directed Code Safety. The IEEE Symposium on Research in Security and Privacy, Research in Security and Privacy IEEE Computer Society Press (1999) 32–45 191

Confidentiality-Preserving Refinement is Compositional – Sometimes

Thomas Santen[1], Maritta Heisel[2], and Andreas Pfitzmann[3]

[1] Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
`santen@acm.org`

[2] Institut für Praktische Informatik und Medieninformatik
Technische Universität Ilmenau, Germany
`maritta.heisel@prakinf.tu-ilmenau.de`

[3] Fakultät Informatik, Technische Universität Dresden, Germany
`pfitza@inf.tu-dresden.de`

Abstract. Confidentiality-preserving refinement describes a relation between a specification and an implementation that ensures that all confidentiality properties required in the specification are preserved by the implementation in a probabilistic setting. The present paper investigates the condition under which that notion of refinement is *compositional*, i.e. the condition under which refining a subsystem of a larger system yields a confidentiality-preserving refinement of the larger system. It turns out that the refinement relation is not composition in general, but the condition for compositionality can be stated in a way that builds on the analysis of subsystems thus aiding system designers in analyzing a composition.

1 Introduction

In systems and software engineering, the consent is growing that secure systems cannot be built by adding security features *ex post* to an existing implementation but that "security-aware" engineering of systems and software must take security concerns into account, starting from requirements engineering through architectural and detailed design to coding, testing, and deployment.

It is obvious that only some kind of divide-and-conquer approach makes building non-trivial systems feasible. Such an approach must support decomposing a system into subsystems, implementing those subsystems largely independently of each other, and finally composing the implementations of those subsystems to make up an implementation of the entire system. More specifically, such an approach decomposes a system *specification* into specifications of subsystems, and it composes (correct) *implementations* of subsystem specifications to yield a (correct) implementation of the system specification.

In this setting, the question arises whether correctness of the subsystem implementations with respect to their specifications is sufficient to guarantee that composing the subsystem implementations yields a correct implementation of the original specification. If this is true, then the implementation relation relating a specification to the set of its correct implementations is called *compositional*. Adapting this definition to security, the question arises whether security-property-preserving implementations of the subsystems with respect to their specifications are sufficient to guarantee that composing the subsystem implementations yields a security-property-preserving implementation of the original specification of the whole system.

When formal techniques are used for system development, both the specification and the implementation are often described in the same formalism. Then, the former is called the *abstract* specification, and the latter is called the *concrete* specification. The relation describing the correct implementations of an abstract specification is called a *refinement* relation, and a concrete specification implementing an abstract one is called a refinement of the abstract specification. For a notion of refinement, the properties of transitivity and compositionality are very important. Without these properties, the practical application of refinement is hardly possible.

In earlier work [4], we have motivated a probabilistic notion of a confidentiality-preserving refinement and sketched its formalization using an extension of CSP with probabilistic choice. Classical formal techniques, which are possibilistic, either impose sufficient conditions that are too strong or impose only necessary conditions that are too weak to realize required confidentiality properties [16].

In the present paper, we further investigate properties of confidentiality-preserving refinement, with the goal of enhancing its potential for practical applicability. For these investigations, we represent our systems using a probabilistic variant of CSP, and slightly rephrase our definition to better capture the intuition motivated in [4]. We prove that the resulting refinement relation is *transitive*. By way of a counterexample, we show that confidentiality-preserving refinement, in general, is *not compositional*. The main contribution of the paper is a necessary and sufficient condition for compositionality of confidentiality-preserving refinement, called *non-disclosure*.

A technical report [14] contains more explanatory prose and the complete proofs of all theorems and lemmas mentioned in the paper.

2 Probabilistic CSP and Behavioral Refinement

We use a probabilistic extension of the process algebra of “Communicating Sequential Processes” (CSP) to formally describe the systems we reason about. Roscoe [12] comprehensively treats classical CSP. In this section, we briefly introduce the notation and the notion of behavioral refinement on which we build confidentiality-preserving refinement in Section 3.

2.1 CSP Notation

A *process* P produces sequences of *events*, called *traces*. An event $c.d$ consists of a channel name c and a data item d. Two processes can *synchronize* on a channel c

by transmitting the same data d over c. If one process generates an event $c.d$ and the other generates an event $c.x$, where x is a variable, both processes exchange data when synchronizing on channel c: the value of x becomes d.

In the following, we describe the CSP notation used in this paper. In the following, P and Q are processes, $e \in \Sigma$ is an event, $X \subseteq \Sigma$ is a set of events, $S \in \Sigma \leftrightarrow \Sigma$ is a relation on events, and $R \in D \leftrightarrow D$ is a relation on data.

The process $e \rightarrow P$ first generates event e, and behaves like P afterwards. The process $P \mathbin{|[X]|} Q$ is a parallel composition of P and Q: if P or Q generate events on channels not in X, then those events appear in an arbitrary order; if a process generates an event on a channel in X, it waits until the other process also generates an event on the same channel; if the data transmitted by both processes are equal (or can be made equal because an event contains a variable), then the parallel composition generates that event, otherwise the parallel composition deadlocks.

In the notion of refinement we use, we are interested in changing data representations (*data refinement*), because many effects compromising confidentiality can be described by distinguishing data representations in an implementation that represent the same abstract data item (e.g., different representations of the same natural number). For a relation R on D, the process $P[\![R]\!]_D$ is the process P where each data item a in events of P is replaced by a data item b that is in relation with a, i.e. $a \, R \, b$ holds.

The process $P \setminus X$ is distinguished from P by *hiding* the channels in $X \subseteq \alpha\, P$, where $\alpha\, P$ is the set of channels used by P. The traces of $P \setminus X$ are the traces of P where all events over channels in X are removed. The external choice $P \,\square\, Q$ is the process that behaves as either P or Q, depending on the event that the environment offers.

For a family of processes $P(x)$, the process $\bigsqcap P(x)$ nondeterministically behaves like one of the $P(x)$. As an extension to classical CSP, we also need a *probabilistic* choice $\bigoplus_x^{\mathcal{P}} P(x)$: this process chooses x – and thus $P(x)$ – according to a probability distribution $\mathcal{P}$.

For behavioral refinements, we disregard distributions on choices and treat all probabilistic choices as nondeterministic ones: the possibilistic version $\widehat{P}$ of a process P is defined by replacing each occurrence of the probabilistic choice $\bigoplus$ by a corresponding nondeterministic choice $\bigsqcap$.

2.2 Refinement of Behavior and Data

There are several notions of refinement for CSP: trace refinement, failure refinement, and failure-divergence refinement. The latter two imply trace refinement. If P is refined by Q, denoted $P \sqsubseteq Q$, then – regardless of the refinement relation used – traces$(Q) \subseteq$ traces(P).

We wish to cover changes of data representations in our refinement relation. Therefore, we extend the usual CSP refinement with a *retrieve relation* mapping concrete to abstract data, and define *behavioral refinement* as a combination of CSP refinement and data renaming according to the retrieve relation.

Definition 1 (Retrieve Relation). *Let P and Q be processes over Σ. A relation $R \in D \leftrightarrow D$ between concrete and abstract data is called a* retrieve relation *from Q to P, if* $\operatorname{dom} R \subseteq \operatorname{data} Q$ *and* $\operatorname{ran} R \subseteq \operatorname{data} P$*, where* $\operatorname{dom} R$ *and* $\operatorname{ran} R$ *denote the domain and range, respectively, of relation R, and* $\operatorname{data} P$ *is the set of data occuring in events of P.*

At some places we need the set $R^{-1}(r)$ of all possible data refined versions of a trace r. Applying R^{-1} to a trace r means applying the inverse of R to the data in each event of the trace, and $R^{-1}(r)$ denotes the set of all such traces:

$$\begin{aligned} R^{-1}(r) = \{t \mid & \operatorname{dom} t = \operatorname{dom} r \wedge \\ & \forall\, i \in \operatorname{dom} r;\ c \in Ch;\ d \in D \bullet \exists\, d' \in R(d) \bullet t(i) = c.d \Rightarrow r(i) = c.d'\} \end{aligned}$$

Definition 2 (Behavioral Refinement). *Let P and Q be processes over Σ. Let R be a retrieve relation from Q to P. Then Q* refines *P via R (written $P \sqsubseteq_R Q$), if $\widehat{P} \sqsubseteq \widehat{Q}[\![R]\!]_D$, where $\sqsubseteq$ is the usual refinement of CSP.*

Behavioral refinement is transitive and monotonic [14].

We will need to consider a restriction of a "concrete" process Q to a behavior implementing a given "abstract" trace r.

Definition 3. *Let Q be a process, R be a retrieve relation abstracting the data in Q, and let r be a trace over the range of R. Then $Q|_r^R$ is a process that chooses a behavior of Q whose starting sequence is compatible with r.*

$$Q|_r^R := \operatorname{Pr}(r)[\![R^{-1}]\!]_D \mid[\alpha\, Q]\mid Q$$

The process $\operatorname{Pr}(r)$ *produces the trace r and behaves arbitrarily afterwards:*

$$\operatorname{Pr}(\langle\,\rangle) = \operatorname{Pr}(\langle\checkmark\rangle) = RUN \qquad \operatorname{Pr}(\langle e\rangle \frown s) = e \rightarrow \operatorname{Pr}(s)$$

The event $\checkmark$ at the end of a trace signifies termination of the process. The process RUN engages in any communication the environment proposes.

Behavioral refinement is defined in terms of the possibilistic versions of the involved processes. Morgan et.al. [11] define a refinement relation for probabilistic processes, which turns out to be a very delicate task when allowing both, nondeterministic and probabilistic choices.[1] Because confidentiality-preserving refinement as defined in Section 3 imposes a condition on processes that, in particular, does not require the probabilistic behavior of a process to be preserved in a refinement, we do not use Morgan, et.al.'s definition[2] of refinement.

[1] It is not easy to avoid either one when defining processes.

[2] An investigation of the relation between the two is nevertheless theoretically interesting.

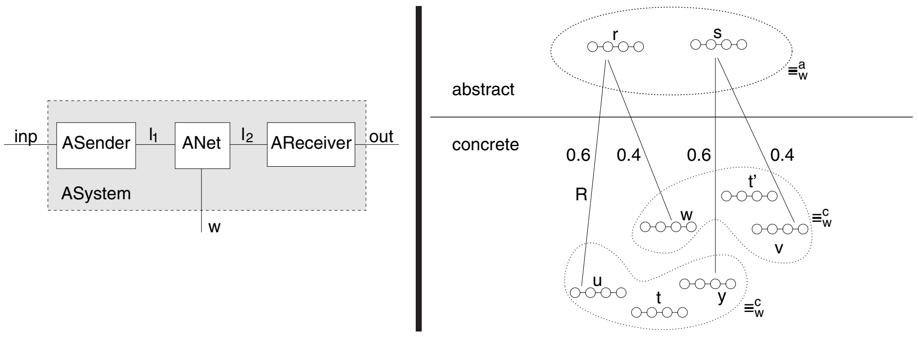


Fig. 1. System model (left), and concretization vs. indistinguishability (right)

2.3 Probability Distributions on Processes

We conclude the brief discourse on probabilistic CSP with some properties of probability distributions, which we will need later. Given a process P, which may contain external, nondeterministic, and probabilistic choices, the set of processes $\mathrm{Prob}(P)$ contains all processes that are obtained by replacing each external choice and each nondeterministic choice in P by a probabilistic choice for some (arbitrary) distribution.

When we will consider probabilistic properties of processes later, we will argue about all members $\mathrm{Prob}(P)$ for a given P, because thus we consider all possible probabilistic behavior of the environment (external choices) and all possible probabilistic behavior of an implemented system for which the process (as a specification) does not determine the distribution (nondeterministic choices).

For a process Q that contains only probabilistic choices, we define a family of probability distributions $\mathcal{P}_n(Q,t)$ that is indexed by the maximal length n of traces it considers: $\mathcal{P}_n(Q,t)$ is a distribution on the set of traces with length n or that terminate (the last event is $\checkmark$) and have a length less than n. For a given t, we write $\mathcal{P}_Q(t)$ for $\mathcal{P}_{\#t}(Q,t)$, where $\#t$ is the length of t.

The probability of Q producing a trace in a set M, which describes a certain property of Q, is given by

$$\mathcal{P}_Q(t \in M) = \sum_{t \in \mathrm{pfree}(M)} \mathcal{P}_Q(t) \tag{1}$$

The set $\mathrm{pfree}(M) \subseteq M$ is the maximal subset of M that does not contain any $t \in M$ for which there is a prefix t' of t in M.

Finally, we define $\mathcal{P}_Q(s) = 0$ for traces $s \notin \mathrm{traces}(Q)$.

3 Confidentiality-Preserving Refinement (CPR)

In this section, we present our definition of confidentiality-preserving refinement, which we have extensively motivated in an earlier publication [4]. To specify confidentiality properties we use a system model illustrated on the left-hand side of Fig. 1 for the example of a communication system between a sender and a receiver communicating over an untrusted network. We specify a system for which confidentiality is a relevant requirement by a pair of a process and a window channel.

Definition 4 (System, Window). *A* system specification *$A = (Q, w)$ is a pair of a process definition Q and a distinguished channel $w \in \alpha\, Q$, called the* window *of A.*

Specifying the system *ASystem* of Fig. 1, we define three processes *ASender*, *ANet*, and *AReceiver*, where *ASender* and *ANet* communicate via the internal channel l_1, and *ANet* and *AReceiver* communicate over the internal channel l_2. Then, the process *ASystem* is the parallel composition of those three processes. The window w is a distinguished channel of *ASystem*.

$$\begin{aligned} ASystem &= (Q_A, w) \\ Q_A &\mathrel{\widehat{=}} ((ASender \mathbin{|[l_1]|} ANet) \mathbin{|[l_2]|} AReceiver) \setminus \{l_1, l_2\} \\ \alpha\, Q_A &= \{inp, out, w\} \end{aligned}$$

The channel w models the flow of data from the system to an adversary. Observing the channel w, the adversary gains information about the system. Any distinction the adversary can make about the internal state of the system based on the observations on w is information that the system does not keep confidential. Conversely, the system keeps confidential any aspect of its behavior that an adversary cannot distinguish by observing w. We formally capture that confidentiality property by defining equivalences over system traces.

Definition 5 (Indistinguishability). *Let $A = (Q, w)$ be a system specification. Two traces $s, t \in$* traces*(Q) are* indistinguishable *by w (denoted $s \equiv_w t$) iff their projections to w are equal: $s \equiv_w t \Leftrightarrow s \upharpoonright \{w\} = t \upharpoonright \{w\}$*

In the transition from an abstract to a concrete system specification, the interpretation of a window changes. The window of an *abstract* system specifies what information is *allowed* to be visible to an adversary. The window of a *concrete* system specifies what information *cannot* be hidden from the adversary.

Here, a purely logical argument is insufficient because it is not enough to ask whether a distinction in the concrete system *definitely* allows an observer to distinguish confidential data, but we must describe whether such a distinction provides *more* information about the confidential data than the abstract window reveals. Therefore, we consider the respective probabilities of internal data that may cause a particular observable behavior on a window. The right-hand side of Fig. 1 illustrates our approach to formalizing that probabilistic argument:

Consider an abstract and a concrete system that behaviorally refines the abstract one with retrieve relation R. Let r and s be two abstract traces that are indistinguishable with respect to the window w, i.e. $r \equiv_w^a s$. According to the retrieve relation R, trace r can be represented by the concrete traces u and w, and trace s can be represented by the concrete traces v and y, where u and y as well as v and w are indistinguishable by observing the concrete window, i.e. $u \equiv_w^c y$ and $w \equiv_w^c v$. For keeping r and s indistinguishable in the concrete system, we must require that the probability that r is represented by u be the same as the probability that s is represented by y. If this were not the case, an adversary might be able to gain information whether r or s happened on the abstract layer: if the probability that r is represented by u is greater than the probability that s is represented by y, for an adversary, the observation of some element $t \equiv_w^c y$ increases the probability of r with respect to s.

Definition 6 reflects this argument: A confidentiality-preserving refinement is one that (1) is the behavioral refinement of the processes describing a system (c.f. Definition 2), and that (2) (probabilistically) preserves the indistinguishability of system traces. In the latter condition, we consider the behavior of the concrete system implementing an abstract trace r, i.e. the process $Q|_r^R$. This process may contain external choices stemming from the different implementation choices that R^{-1} assigns to the data in r. Because there is no way of knowing with what probability those implementation choices are resolved, we need to consider all possible distributions that make $Q|_r^R$ a probabilistic process, i.e. all members of $\mathrm{Prob}(Q|_r^R)$.

Remark 1. To keep the language simple, we will talk about a probabilistic property E of a process Q, when we mean that all members $Q_p \in \mathrm{Prob}(Q)$ satisfy E.

Definition 6 (Confidentiality-Preserving Refinement, CPR). *Let $A = (P, w)$ and $C = (Q, w)$ be two system specifications. Let $\equiv_w^a$ be the indistinguishability in A (wrt. w), and let $\equiv_w^c$ be the indistinguishability in C (wrt. w). The system C is a* confidentiality-preserving refinement (CPR) *of the system A via the retrieve relation R from Q to P ($A \sqsubseteq_R^{cpr} C$) iff:*

1. $P \setminus \{w\} \sqsubseteq_R Q \setminus \{w\}$, *and* *behavioral refinement,BR*
2. $\forall\, r, s \in \mathrm{traces}(P);\ t \in \mathrm{traces}(Q);$ *indistinguishability preservation,IP*
   $Q_r \in \mathrm{Prob}(Q|_r^R);\ Q_s \in \mathrm{Prob}(Q|_s^R) \bullet$
   $r \equiv_w^a s \Rightarrow \mathcal{P}_{Q_r}(u \equiv_w^c t) = \mathcal{P}_{Q_s}(v \equiv_w^c t)$

We write $P \sqsubseteq_{R,w}^{cpr} Q$ for $(P, w) \sqsubseteq_R^{cpr} (Q, w)$, which is useful when analyzing systems with respect to different windows. Although the windows of the two systems have the same name w, they may carry different data because the processes of the systems determine the data that is transmitted on a channel.

Hiding w from P and Q in Condition *BR*, Definition 6 does not require Q to refine the window w: At the implementation level, an adversary may have means of observation that are in no way related to the means of observation given at the specification level. Requiring Q to refine the window w therefore would – inadequately – allow the specification to impose restrictions on the power of an adversary at the level of implementation. Condition *IP* captures the important restriction on confidentiality-preserving implementations that adversaries must not be able to infer more information by observing the implementation than the window at the specification level allows them to. It states that, given two indistinguishable abstract traces r and s, and a concrete trace t, the probability of choosing a concrete trace u which is indistinguishable of t as an implementation of r must be the same as choosing a concrete trace v which is indistinguishable of t as an implementation of s, see Fig. 1.

To determine the probability $\mathcal{P}_{Q_r}(u \equiv_w^c t)$, we consider a subset of the set $T = \{u \mid u \in \text{traces}(Q_r) \wedge u \equiv_w^c t\}$ of all traces that are indistinguishable from t. Because T need not be prefix-free, we must consider the set pfree(T) for calculating probabilities. Then, it holds:

$$\mathcal{P}_{Q_r}(u \equiv_w^c t) = \sum_{u \in \text{pfree}(T)} \mathcal{P}_{Q_r}(u)$$

To support stepwise refinement and the independent refinement of subsystems, a refinement relation must have two properties: it must be transitive and compositional. The following Theorem 1 establishes the transitivity of CPR. Section 4 extensively discusses compositionality.

Theorem 1 (Transitivity of CPR). *Let $A = (P_a, w)$, $B = (P_b, w)$, and $C = (P_c, w)$ be system specifications. Let R_{ba} and R_{cb} be retrieve relations from P_b to P_a, and from P_c to P_b, respectively. Then $A \sqsubseteq_{R_{ba}}^{cpr} B \wedge B \sqsubseteq_{R_{cb}}^{cpr} C \Rightarrow A \sqsubseteq_{R_{cb} \fatsemi R_{ba}}^{cpr} C$, where $\fatsemi$ is the forward composition of relations.*

4 Compositionality of CPR

Compositionality of security properties and compositionality of refinement are two different notions. Section 4.1 contrasts the two. Indistinguishability is not a compositional property, and CPR is, in general, not compositional. Section 4.2 illustrates this fact by way of a counterexample. For a composed system for which refinements of subsystems are known, we can find a condition that characterizes the circumstances under which a CPR is compositional. This condition is more intuitive than the Condition *IP* of Definition 6, and it allows one to build on the analyses made for verifying the CPR of subsystems. Section 4.3 establishes this result in Theorem 2.

4.1 Compositionality of Security Properties vs. Compositionality of Refinement

Compositionality of a security property, as it is often considered in the context of secure systems, means the preservation of that property under composition of systems: if certain systems satisfy a certain security property, some variant of non-interference, say, then the composition of those systems satisfies the same property. Mantel [10] investigates the relation between compositionality results for many known information flow properties.

Compositionality of a refinement relation, in which we are interested in this paper, addresses the interplay of decomposing a system specification, and composing the implementations of the subsystems to yield an implementation of

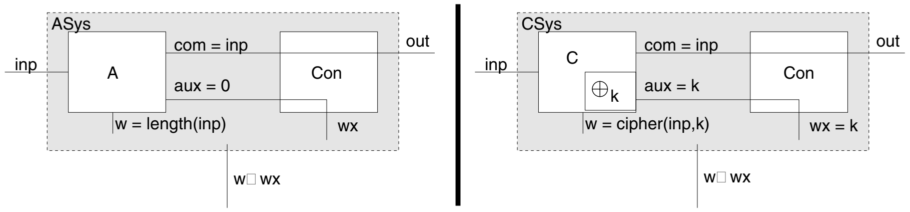


Fig. 2. Confidentiality-preserving refinement is not compositional

the original system specification. Thus it is a preservation property that relates different levels of abstraction (specification – implementation), whereas the compositionality of security properties is concerned with one level of abstraction only. As we will see in the following section, the security property indistinguishability (c.f. Definition 5) is not compositional. Since this is true for the abstract as well as the concrete level of a refinement, the non-compositionality of indistinguishability, in principle, *weakens* the requirements for a refinement to be compositional. As a consequence, embedding a system into a context but refining it in isolation leads to two questions to be answered:

1. Does the composed system still fulfill the desired security requirements?
2. Does the replacement of the abstract by the concrete system in the given context compromise security?

Question 1 means that the *composition* has to be considered and possibly be rejected, independently of later refinements. The present paper does not address this problem. Since the usual notion of correctness of refinement does provide for preservation of integrity and hopefully availability, but not at all for confidentiality, we narrow Question 2 to: Does the replacement of the abstract by the concrete system in the given context compromise confidentiality? This amounts to showing that the refinement is compositional for the given context. In the rest of the paper, we will show how to answer this question.

4.2 A Counterexample

The two systems shown in Fig. 2 illustrate that, in general, CPR is *not* compositional. The left-hand side of the figure shows an abstract system *ASys* with two communication channels *inp* and *out*, and a window $w \otimes wx$ that is a combination of the windows w and wx of the two subsystems A and *Con*: all data observable on w or wx are also observable on $w \otimes wx$. The subsystem A specifies a secure communication service. It allows its environment to observe the length of messages transmitted from channel *inp* to channel *com*, but no other information about the content of messages. The subsystem *Con* specifies the context in which A operates. The context *Con* copies the data that A produces on *com* to its output channel *out*. The systems A and *Con* also communicate via the channel *aux*: *Con* receives data from A that do not contain any relevant information (represented by a constant 0). The window *wx* of *Con* allows data received on *aux* to be observed by the environment.

The right-hand side of Fig. 2 shows an implementation *CSys* of *ASys*. The subsystem C is an implementation of the communication service that A specifies in the presence of an untrusted network. The implementation C probabilistically chooses keys k with equal probabilities and uses a (suitable) encryption function *cipher* to conceal the transmitted data from an observer who can intercept communication on the network: the window w of C allows an adversary to observe the ciphertext $cipher(inp, k)$.

Because observing that ciphertext will reveal the length of the message but nothing else about its content, the system C is a CPR of the system A, as shown in [4]. If CPR was unconditionally compositional, we would expect *CSys*, which is obtained from *ASys* by substituting C for A, to be a CPR of *ASys*. This, however, is not true: C not only implements A correctly[3], but it also transmits the selected key k over the channel *aux*. This does not compromise confidentiality if C is considered in isolation, because w does not make information about the data on *aux* available to the adversary. Composing C with *Con* as in *CSys*, however, allows the adversary to observe the key k on the *new* window *wx*! Thus, the combination of the information gained by observing w and *wx* reveals the original input message to an adversary, which is not revealed on the abstract level.

In this example, the non-compositionality of CPR is a direct consequence of the non-compositionality of indistinguishability: composing the windows w and *wx* in *CSys* makes *more* observations possible (the key becomes observable) and thus an observer can distinguish more behavior of the subsystems than by observing their respective windows alone.

The same argument, however, is also true for the abstract level: the indistinguishability *requirement* on the implementation will, in general, become weaker by combining windows, thus strengthening the premise of Condition *IP* in Definition 6, and allowing for *more* confidentiality-preserving refinements. Additionally, much more subtle effects relating to the probabilistic nature of Definition 6 may compromise the compositionality of CPR.

4.3 A Condition for Compositionality

After the somewhat discouraging result of the previous section, we will now investigate the conditions under which CPR is nevertheless compositional. We will make precise the intuition gained from analyzing the counterexample, and come up with a condition for compositionality that reduces the question of compositionality to the question what *additional* distinctions a new window on the subsystem allows an adversary to make.

[3] The retrieve relation maps each key transmitted on channel *aux* to the constant 0, which is an admissible data refinement.

Formally, we consider two systems $A = (P, w)$ and $C = (Q, w)$ with $A \sqsubseteq_R^{cpr} C$ for some retrieve relation R from Q to P. The context $Con = (Cx, wx)$ is another system that can communicate with A and C via a set of channels K. The context window is different[4] from the window on A and C: $wx \neq w$.

Combining the system A with the context Con yields the system $(P \,|[K]|\, Cx, w \otimes wx)$. We assume here that the processes P and Cx work on the same set of abstract data. Therefore, to combine C with Con, we must "concretize" the data in Cx by a data renaming consistent with the retrieve relation from Q to P: $(Q \,|[K]|\, Cx[\![R^{-1}]\!]_D, w \otimes wx)$. Viewed abstractly, the process $Cx[\![R^{-1}]\!]_D$ has the same behavior as Cx, but for each data item a that Cx transmits, the process $Cx[\![R^{-1}]\!]_D$ transmits a data item b that implements a, i.e. for which $b \, R \, a$ holds. In this setting, compositionality means that C combined with Con refines A combined with Con:

$$(P \,|[K]|\, Cx, w \otimes wx) \sqsubseteq_R^{cpr} (Q \,|[K]|\, Cx[\![R^{-1}]\!]_D, w \otimes wx) \tag{2}$$

If Condition *IP* of Definition 6 does not hold for (2), then the additional observations of the concrete system that the window wx permits via the context must allow an adversary to distinguish more behavior of P than the window w permits. Composing the context with the processes P or Q, respectively, forces them to synchronize with the context and thus reduces their possible behavior. This may change the probabilities of traces of P and Q, which might also affect Condition *IP*.

This analysis motivates the three essential tasks to solve for stating our compositionality theorem:

1. to reduce the combination of a system with a context, which itself has an additional window, to adding a window to the system;
2. to come up with a condition describing the circumstances under which a CPR between two systems is preserved under addition of a window to both systems;
3. to show that reducing behavior by adding a context preserves CPR.

To solve the first task, we wish to consider the context as a means of an adversary to observe the behavior of P or Q at the channels K. Technically, we can achieve this by hiding all channels of Cx but wx and but the ones in K from the composed systems. This leads to the notion of a system in context:

Definition 7 (System in Context). *Let $A = (P, w)$ and $Con = (Cx, wx)$ be system specifications. Let $K \subseteq (\alpha\, P \cap \alpha\, Cx) - \{w, wx\}$ be a set of channels over which A and Con can communicate, and let $X := \alpha\, Cx - (\alpha\, P \cup \{wx\})$. Then A* in context *Con, written $A \overset{K}{\lhd} Con$, is the system $(P \,|[K]|\, (Cx \setminus X), w \otimes wx)$.*

The system $A \overset{K}{\lhd} Con$ is the system A with an additional window wx and the reduced behavior that is a consequence of synchronizing with Con. Similarly,

[4] Although the channel names are different, the same data can, of course, be transmitted over those channels.

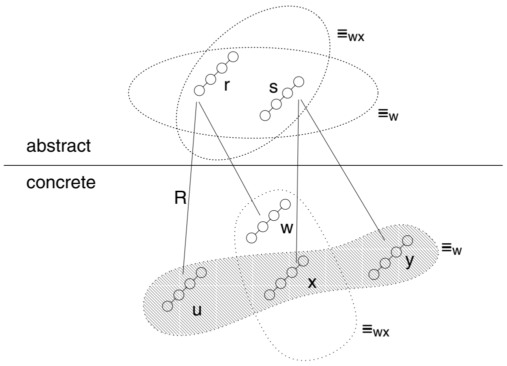


Fig. 3. Non-Disclosure

at the concrete level, the system $C \overset{K}{\lhd} Con[\![R^{-1}]\!]_D$ is the system C with an additional window and reduced behavior. Lemma 1 shows that it is sufficient to prove a CPR between those systems in context to establish (2).

Lemma 1. *Let $A = (P, w)$, $C = (Q, w)$, and $Con = (Cx, wx)$ be system specifications. Let $w \in (\alpha\, P \cap \alpha\, Q) - \alpha\, Cx$, $wx \in \alpha\, Cx$, $K \subseteq (\alpha\, P \cap \alpha\, Q \cap \alpha\, Cx) - \{w, wx\}$, and $X := \alpha\, Cx - (\alpha\, P \cup \alpha\, Q \cup \{wx\})$. Let R be a retrieve relation from Q to P. Let $(\widetilde{P}, w \otimes wx) := A \overset{K}{\lhd} Con$ and $(\widetilde{Q}, w \otimes wx) := C \overset{K}{\lhd} Con[\![R^{-1}]\!]_D$. Then*

$$P \sqsubseteq^{cpr}_{R,w} Q \wedge \widetilde{P} \sqsubseteq^{cpr}_{R,w\otimes wx} \widetilde{Q} \;\Rightarrow\; P \,|[K]|\, Cx \sqsubseteq^{cpr}_{R,w\otimes wx} Q \,|[K]|\, Cx[\![R^{-1}]\!]_D$$

To solve the second task, consider the effect of adding a window wx to the abstract and concrete systems. By the following Equivalence (3), going from w to $w \otimes wx$ makes the equivalence classes of the indistinguishability finer.

$$s \equiv_{w\otimes wx} t \Leftrightarrow s \equiv_w t \wedge s \equiv_{wx} t \tag{3}$$

As Fig. 3 illustrates, the equivalence classes at both, the abstract and the concrete level, become finer. Thus, we need to consider *fewer* pairs (r, s) of abstract traces in Condition *IP* of Definition 6, which weakens the condition, but at the same time, we need to show a *stronger* property for the pairs (r, s) that we still must consider: For all traces t of the concrete system, the probability of the concrete system to choose a behavior u implementing r that is indistinguishable from t by both w and wx must be the same as the probability of the system to choose an implementation v of s that is indistinguishable from t by both w and wx. Formally, we need to show:

$$\textstyle\sum_{u \,\in\, \mathrm{pfree}\left([t]_{\equiv^c_w} \cap [t]_{\equiv^c_{wx}}\right)} \mathcal{P}_{Q_r}(u) = \sum_{v \,\in\, \mathrm{pfree}\left([t]_{\equiv^c_w} \cap [t]_{\equiv^c_{wx}}\right)} \mathcal{P}_{Q_s}(v) \tag{4}$$

We already know $[t]_{\equiv_w^c}$ from proving $A \sqsubseteq_R^{cpr} C$. Proving Equation (4) as it stands would mean to analyze which behavior of C that is indistinguishable by w remains indistinguishable when adding wx. From a practical point of view, however, it is more suitable to analyze which observations made using wx allow an adversary to *distinguish* behavior that is indistinguishable by w, and to compare probabilities for that behavior. This means to let the sums range over the set difference of the respective equivalence classes. If the resulting equation of probabilities holds, we call wx *non-disclosing* on the system with respect to r and s.

Definition 8 (Non-Disclosure). *Let $S = (Q, w)$ be a system specification, and let $wx \in \alpha\, Q - \{w\}$ be a channel of Q that is distinct from w. Let R be a retrieve relation from Q to the data in two traces r and s. We call wx* non-disclosing *on S wrt. R, r and s, written $Q|_{r,s}^R \vdash w \triangleq wx$, iff the following condition holds:*

$$\forall\, t : \text{traces}(Q);\ Q_r \in \text{Prob}(Q|_r^R);\ \ Q_s \in \text{Prob}(Q|_s^R) \bullet$$
$$\sum\nolimits_{u \in \left(T(Q_r,t) \,-\, [t]_{\equiv_{wx}}\right)} \mathcal{P}_{Q_r}(u) \;=\; \sum\nolimits_{v \in \left(T(Q_s,t) \,-\, [t]_{\equiv_{wx}}\right)} \mathcal{P}_{Q_s}(v)$$

The set of traces $T(Q,t)$ is given by $T(Q,t) := T_0(Q,t) \cup \text{pfree}(T_1(Q,t))$, where

$$\begin{aligned} T_0(Q,t) = \{u \in \text{traces}(Q) \mid u \equiv_w t \wedge \#(u \upharpoonright \{wx\}) = \#(t \upharpoonright \{wx\}) \wedge \\ \neg\, (\exists\, u' \in \text{traces}(Q) \bullet u' \text{ prefix } u \wedge \\ u' \equiv_w t \wedge \#(u' \upharpoonright \{wx\}) = \#(t \upharpoonright \{wx\}))\} \end{aligned} \tag{5}$$

$$\begin{aligned} T_1(Q,t) = \{u \in \text{traces}(Q) \mid u \equiv_w t \wedge u \notin T_0(Q,t) \wedge \\ \neg\, (\exists\, u' \in T_0(Q,t) \bullet u \text{ prefix } u')\} \end{aligned} \tag{6}$$

The prefix-free set $T(Q,t)$ is an alternative for $\text{pfree}([t]_{\equiv_w})$ when computing the probability that Q produces a trace u with $u \equiv_w t$. Additionally, the (rather technical) construction ensures that $T(Q,t)$ contains a maximal number of traces that have as many observations over the other window wx as t has.

Lemma 2 states that – given $P \sqsubseteq_{R,w}^{cpr} Q$ – non-disclosure characterizes the circumstances under which CPR is preserved when a new window is added.

Lemma 2. *Let P and Q be processes, $w, wx \in \alpha\, P \cap \alpha\, Q$ be channels common to P and Q, and let R be a retrieve relation from Q to P. If $P \sqsubseteq_{R,w}^{cpr} Q$ then*

$$P \sqsubseteq_{R,w \otimes wx}^{cpr} Q \Leftrightarrow \left(\forall\, r, s \in \text{traces}(P) \bullet r \equiv_{w \otimes wx}^a s \Rightarrow Q|_{r,s}^R \vdash w \triangleq wx\right)$$

To solve the third task, we need Lemma 3 that relates a given CPR to a CPR of the same systems in a context that does *not* add a new window. When we apply this lemma to prove compositionality, we will regard wx not as a window but as an ordinary channel of Cx.

Lemma 3. *Let $A = (P, w)$, $C = (Q, w)$, and $Con = (Cx, wx)$ be system specifications with $w \neq wx$. Let $K \subseteq (\alpha\, P \cap \alpha\, Q \cap \alpha\, Cx) - \{w, wx\}$ be a set of* *channels over which A and Con, and C and Con, respectively, can communicate. Let $(\widetilde{P}, w \otimes wx) := A \overset{K}{\lhd} Con$ and $(\widetilde{Q}, w \otimes wx) := C \overset{K}{\lhd} Con[\![R^{-1}]\!]_D$. Then $P \sqsubseteq^{cpr}_{R,w} Q \Rightarrow \widetilde{P} \sqsubseteq^{cpr}_{R,w} \widetilde{Q}$.*

Lemmas 1, 2, and 3 allow us to prove the main result of this paper: CPR is compositional if the context window wx is non-disclosing on the refined subsystem.

Theorem 2 (Compositionality of CPR). *Let $A = (P, w)$, $C = (Q, w)$, and $Con = (Cx, wx)$ be system specifications with $w \neq wx$. Let $K \subseteq (\alpha\, P \cap \alpha\, Q \cap \alpha\, Cx) - \{w, wx\}$ be a set of channels over which A and Con or C and Con, respectively, can communicate. Let R be a retrieve relation from Q to P. Let $\widetilde{P}$ be the process of $A \overset{K}{\lhd} Con$, and let $\widetilde{Q}$ be the process of $C \overset{K}{\lhd} Con[\![R^{-1}]\!]_D$. If*

1. *$A \sqsubseteq^{cpr}_{R} C$, and*
2. *$\forall \widetilde{r}, \widetilde{s} : \mathrm{traces}(\widetilde{P}) \bullet \widetilde{r} \equiv^{a}_{w\otimes wx} \widetilde{s} \Rightarrow \widetilde{Q}|^{R}_{\widetilde{r},\widetilde{s}} \vdash w \mathrel{\hat{=}} wx$*

then $(P \,|[K]|\, Cx, w \otimes wx) \sqsubseteq^{cpr}_{R} (Q \,|[K]|\, Cx[\![R^{-1}]\!]_D, w \otimes wx)$.

Proof. Assume $P \sqsubseteq^{cpr}_{R,w} Q$. With Lemma 3, we get $\widetilde{P} \sqsubseteq^{cpr}_{R,w} \widetilde{Q}$, which implies $\widetilde{P} \sqsubseteq^{cpr}_{R,w\otimes wx} \widetilde{Q}$ by Assumption 2 and Lemma 2. From Assumption 1 and Lemma 1, we conclude $P \,|[K]|\, Cx \sqsubseteq^{cpr}_{R,w\otimes wx} Q \,|[K]|\, Cx[\![R^{-1}]\!]_D$. □

5 Related Work

Because indistinguishability and its preservation by refinement (CPR) is concerned with hiding certain information about events occurring in a system from an adversary, our work is related to research on non-interference.

Non-interference, first introduced by Goguen and Meseguer [1], is a security property that has extensively been studied. Much work on non-interference is possibilistic, i.e. it disregards probabilistic arguments. Ryan and Schneider [13] recast many known definitions of possibilistic non-interference in (classical) CSP and show that the different ways of defining non-interference are closely related to the different notions of process equivalence.

The windows in our setting can be viewed as a channel from the considered system to an outside adversary, i.e. from the "high" system to the "low" outside world. In contrast to non-interference, we do *not* require no information to flow through that channel. Our definition of CPR ensures that possible observations which adversaries may make of the implemented system do not offer them additional ways of inferring information about the system than the specification allows them to.

Ryan and Schneider [13] discuss an approach of generalizing non-interference that has a similar motivation: requiring total absence of information flow often is too strong in practice. Their generalization is parameterized by an equivalence on traces, an equivalence on processes, and a way of abstracting High's behavior from a process. Depending on the instantiation of these parameters one obtains weak versions of non-interference that allow Low to determine High's behavior up to the equivalence on traces. It may be interesting to recast our definition of indistinguishability into that framework.

Gray [3] defines probabilistic non-interference (PNI), and Jürjens [6] proves a compositionality theorem for a variant of that definition. The condition for PNI basically states that the probability of a high user producing a particular observation of a low user is the same for all behaviors of the high user. Because it formalizes the fact that certain behaviors cannot be distinguished probabilistically, this condition of PNI is similar to our refinement condition IP. The difference is that IP requires equal probabilities of (indistinguishable) concrete behavior only for implementations of indistinguishable abstract behavior.

Lowe [8] recently investigated how to quantify information flow from high to low while staying in a possibilistic setting. Using a discretely timed version of CSP, he can analyze timing channels, which we currently ignore. The aim of Lowe's work is similar to ours in that he does not per se require no information to flow from high to low: he puts bounds on the capacity of channels whereas (by the abstract window) we restrict the ways in which information may flow from high to low.

Graham-Cumming and Sanders [2] discuss the preservation of non-interference under data refinement. They specify systems using the specification language Z [15] and define security as indistinguishability on system traces with respect to a given user. They give conditions under which a refinement of the internal data of the system preserves indistinguishability. Their approach is possibilistic, and, in contrast to our setting, they consider only refinements of the internal state of a system but not of the input and output data. We emphasize refining the inputs and outputs, because an implementation must be designed in such a way that choosing particular representations of inputs and outputs does not allow adversaries to infer more information about the system than they are allowed to.

Mantel [9] considers the preservation of information flow properties under refinement. It is well-known that CSP-style refinement does not preserve information flow properties in general [5]. Mantel shows how refinement operators tailored for specific information flow properties can modify an intended refinement such that the resulting refinement preserves the given flow property. Working top-down from the specification to an implementation, the refinement operators may lead to concrete specifications that are practically hard to implement, because the changes in the refinement they induce are hard to predict and may not be easy to realize in an implementation.

Jürjens [7] uses stream processing functions to model systems, and he defines a possibilistic notion of secrecy in that setting. He identifies conditions under which certain refinement operators on stream processing functions preserve his notion of secrecy.

6 Conclusions

Security-aware engineering of systems and software needs a notion of refinement which comprises not only integrity and availability, but confidentiality as well. To contribute to providing a firm basis for security-aware engineering, we developed a precise notion of confidentiality-preserving refinement (CPR). CPR inherits all properties of behavioral refinement and additionally introduces indistinguishability preservation, which is the probabilistic characterization of confidentiality-preservation.

At the end of Section 4.1, we have identified two questions concerning the composition and refinement of secure systems: (i) Does the composed system still fulfill the desired security properties? (ii) Does the replacement of the abstract by the concrete system in a given context compromise confidentiality? Question (i) should be investigated in more detail, taking into account our definition of CPR. For this investigation, one can build on the work of Mantel [10] and Ryan and Schneider [13]. Concerning the investigation of compositionality of refinement in a probabilistic setting, i.e., question (ii), we know of no work prior to ours, as Graham-Cumming and Sanders [2], Mantel [9], and Jürjens [7] consider possibilistic refinement only.

The present paper shows that confidentiality-preserving refinement is transitive, but not compositional in general. An analysis of the situation shows that this result is not surprising. It is even inevitable, because confidentiality properties are of a fundamentally different nature than integrity (and – to a certain extent – availability) properties, which correspond to the notion of correctness as considered in classical refinement. Refining a subsystem that is embedded in a context yields a refinement of the composed system, because the refined subsystem always behaves in a way that is consistent with the behavior of the abstract subsystem. A corresponding property does not hold for confidentiality. As Section 4.2 shows, it is possible to refine a subsystem in such a way that additional ways of obtaining information about the subsystem become possible on the concrete level as compared to the abstract level. This is due to the facts that, first, the context adds an additional window to the system, and second, "non-confidential" data may be refined to data that permits an adversary to gain additional information as compared to the abstract system.

In such a situation, the only possible remedy is to investigate the conditions that must hold in addition to the confidentiality-preserving refinement of the subsystem. If proving these conditions is easier than proving the CPR for the composed systems from scratch, then the notion of confidentiality-preserving refinement is still useful for stepwise development using a divide-and-conquer approach.

With the notion of non-disclosure, we capture the additional condition that must hold to guarantee the compositionality of CPR. This condition corresponds well to the intuition that (i) the "information leaks" introduced by the context can be represented by the additional data visible in the context's window, and (ii) that adding a new window must not change the relative probabilities of indistinguishable traces. Non-disclosure does not only give insight into our notion of CPR, but also into the relationship between refinement and compositionality in general.

We can therefore conclude that the notion of CPR may be useful for security-aware engineering of systems and software, even if it cannot be compositional in general. The work presented in this paper lays the foundations for an engineering approach to CPR: identifying architectures that guarantee non-disclosure by construction will facilitate the task of proving non-disclosure. Further research must apply our definitions and theorems to examples of larger scale, further investigate the relation of indistinguishability of traces to other security properties, as well as start the development of tools supporting our approach.

Acknowledgement

We thank Peter Ryan for extensive and constructive feedback on a previous version of the paper. The anonymous reviewers also provided useful comments. The program committee allowed us to considerably rework the paper during the acceptance process. Thanks also go to Sandra Steinbrecher and Elke Franz for asking good questions and finding some typos.

References

[1] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982. 207

[2] J. Graham-Cumming and J. W. Sanders. On the refinement of non-interference. In *9th IEEE Computer Security Foundations Workshop*, pages 35–42. IEEE Computer Society Press, 1991. 208, 209

[3] J. W. Gray. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, 1992. 208

[4] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-preserving refinement. In *14th IEEE Computer Security Foundations Workshop*, pages 295–305. IEEE Computer Society Press, 2001. 195, 198, 203

[5] J. Jacob. On the derivation of secure components. In *IEEE Symposium on Security and Privacy*, pages 242–247. IEEE Press, 1989. 208

[6] J. Jürjens. Secure information flow for concurrent processes. In *CONCUR 2000*, LNCS 1877. Springer-Verlag, 2000. 208

[7] J. Jürjens. Secrecy-preserving refinement. In J. N. Oliveira and P. Zave, editors, *FME 2001: Formal Methods for Increasing Software Productivity*, LNCS 2021, pages 135–152. Springer-Verlag, 2001. 208, 209

[8] G. Lowe. Quantifying information flow. In *15th IEEE Computer Security Foundations Workshop*, pages 18–31. IEEE Computer Society, 2002. 208

[9] H. Mantel. Preserving information flow properties under refinement. In *IEEE Symposium on Security and Privacy*, pages 78–91. IEEE Computer Society Press, 2001. 208, 209

[10] H. Mantel. On the composition of secure systems. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2002. to appear. 201, 209

[11] C. Morgan, A. McIver, K. Seidel, and J. W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996. 197

[12] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998. 195

[13] P. Y. A. Ryan and S. A. Schneider. Process algebra and non-interference. In *12th IEEE Computer Security Foundations Workshop*, pages 214–227. IEEE Computer Society, 1999. 207, 209

[14] T. Santen, M. Heisel, and A. Pfitzmann. Compositionality of confidentiality-preserving refinement. Technical Report 10/2002, Technische Universität Berlin, 2002. 195, 197

[15] J. M. Spivey. *The Z Notation – A Reference Manual*. Prentice Hall, 2nd edition, 1992. 208

[16] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161. IEEE, 1990. 195

Formal Security Analysis with Interacting State Machines

David von Oheimb and Volkmar Lotz

Siemens AG, Corporate Technology, D-81730 Munich
{David.von.Oheimb,Volkmar.Lotz}@siemens.com

Abstract. We introduce the ISM approach, a framework for modeling and verifying reactive systems in a formal, even machine-checked, way. The framework has been developed for applications in security analysis. It is based on the notion of Interacting State Machines (ISMs), sort of high-level Input/Output Automata. System models can be defined and presented graphically using the AutoFocus tool. They may be type-checked and translated to a representation within the theorem prover Isabelle or defined directly as Isabelle theories. The theorem prover may be used to perform any kind of syntactic and semantic checks, in particular semi-automatic verification. We demonstrate that the framework can be fruitfully applied for formal system analysis by two classical application examples: the LKW model of the Infineon SLE66 SmartCard chip and Lowe's fix of the Needham-Schroeder Public-Key Protocol.

1 Introduction

1.1 Motivation

In industrial environments, there is an increased demand for rigorous analysis of security properties of systems. Due to restrictions imposed by the application domain, the system environment, and business needs, new security mechanisms and architectures have to be invented frequently, with time-to-market pressure and intellectual property considerations obstructing the chance to gain confidence by exposing a proposed solution to the security community (which has been shown to be appropriate for cryptographic algorithm assessment). Formal analysis of suitable abstractions of systems has instead turned out to be extremely helpful in reasoning about a system's security, since the mathematical precision of the arguments allows for maximal confidence in the results obtained and, thus, in the security of the system being modeled.

The importance of formal analysis – on top of open review – in security assessment is, for instance, reflected by the requirements stated for high assurance levels of criteria like ITSEC [ITS91] and CC [CC99], which include formal security modeling and formal system development steps, and the achievements of the security protocol verification community, which discovered flaws in protocols that failed to be detected by informal approaches.

However, even in a formal setting it is easy to make – minor and sometimes even major – mistakes: undefined expressions, type mismatches, inconsistent specifications, missing evidence in proofs, false conclusions etc. Therefore, pure pen-and-paper formalizations cannot be considered fully reliable. Machine-checking of formal objects and structures has to be employed in order to significantly reduce the occurrence of such mistakes. Machine support additionally gives the opportunity to represent and deal with formal objects – both specifications and proofs – in an easy-to-comprehend way, which is a prerequisite for introducing formal approaches in an industrial environment characterized by time and cost restrictions.

1.2 Goals

A framework for formal security analysis particularly suited for industrial use should enjoy the following properties:

Expressiveness. It should be possible to describe any typical security sensitive computation, storage, and communication systems in an abstract way. This requires in particular the notions of state transformation, concurrency and message passing.

Flexibility. Since IT systems and their security threats evolve quickly, the models produced within the framework should be easily adaptable and extendable as necessary to reflect the changes.

Simplicity. Modeling a system, stating its properties and proving them should require as little expertise and time as possible while maintaining the rigor of a fully formal approach.

Graphical capabilities. System models should be representable as diagrams that give a good overview of the system structure and a quick intuition about its behavior.

Maturity of the semantics. The semantic foundation of the framework should be well-developed, supporting in particular modular refinement.

Availability of tools. The framework should be built from existing widely available (open-source) software like editors and proof tools and require at most minor modifications or extensions to them.

1.3 Related Work

The *IOA Language and Toolset* [GL98, Kay01] is a framework for analyzing computational processes with aims very similar to ours. It consists of a specification language and tool support for simulation, theorem proving, model checking, and code generation, where by now the simulation aspect is developed most and theorem proving support is limited to PVS. Its semantic foundation is the notion of *I/O Automata (IOAs)* [LT89] modeling asynchronous distributed computation with synchronous communication. Since the notion is based on transition systems augmented by communication primitives (rather than e.g. a process algebra augmented by local computation primitives), it is fairly easy to understand. It is equipped with a well-developed meta theory supporting refinement and compositional reasoning. System properties, both safety and lifeness ones, may be described using temporal logics and proved by model checking and interactive theorem proving.

The only — but severe — drawback of IOAs from our perspective, in particular when modeling system security in an abstract way, is that their interaction scheme is rather low-level: buffered communication has to be modeled explicitly, and transitions involving several related input, internal processing and output activities cannot be expressed atomically. Instead, each high-level transition has to be split into multiple low-level transitions, and between these, any number of further input events may take place due to the input-enabledness of IOAs. The solution to this problem is to add extra structure, essentially by interpreting parts of the local state of an automaton as input/output buffers. Our notion of ISMs, introduced in [Ohe02a], provides for that.

A further related work that provided inspiration for our framework is AutoFocus [HSSS96] – see also §2.2. Even though developed primarily for modeling and verifying functional properties of embedded systems, it is used also for the securiy analysis of general distributed systems [WW01, JW01].

Other related approaches combine state-oriented and message-oriented description methods, for example translating CSP to B [But99] or Z to CSP [Fis00]. The drawback of such hybrids is that the user has to deal with two different nontrivial formalisms. Moreover, theorem proving support respecting the structure of the mixed-style specifications seems not to be available.

2 Preliminaries

In this section, we briefly introduce the two software tools we rely on and comment on their suitability for the ISM approach.

2.1 Isabelle/HOL

Isabelle [Pau94] is a generic theorem prover that has been instantiated to many logics, in particular the very practical *Higher-Order Logic (HOL)*. Isabelle/HOL [PNW$^+$] is a predicate logic based on the simply-typed λ-calculus and thus in a sense combines logical and functional programming. Being quite expressive and supporting automatic type inference, it is the most important and best supported logic of Isabelle. The lack of dependent types introduces a minor nuisance for applications like ours: for each system modeled there is a single type of message contents into which all message data has to be injected, and analogously for the local states of automata.

Proofs are conducted primarily in an interactive fashion where automatic and semi-automatic methods are available to tackle the routine parts. The Isabelle system is well-documented and well-supported, is freely available (including sources) and comes with the excellent user interface ProofGeneral [AGKS99]. We consider it the most flexible and mature verification environment available.

Using Isabelle/HOL, security properties can be expressed easily and adequately and verified with powerful proof methods.

2.2 AutoFocus

AutoFocus [HSSS96] is a freely available specification and simulation tool for distributed systems. Components and their behavior are specified by a combination of *system structure diagrams (SSDs)*, *state transition diagrams (STDs)* and auxiliary *data type definitions (DTDs)*. Their execution can be visualized using *extended event traces (EETs)*. Various back-ends including code generators and interfaces to model checkers may be acquired by purchase from Validas [S^+].

We employ AutoFocus for its strengths concerning graphical design and presentation, which is important when setting up models in collaboration with clients (where strong familiarity with formal notations cannot be assumed), when documenting our work and publishing its results. For abstract security modeling, there are currently two problems. First, expressiveness is limited concerning the type system and the handling of underspecification. These weaknesses are going to be removed in the near future. Second, due to the original emphasis of AutoFocus on embedded systems, the underlying semantics is still clock-synchronous. In contrast, for the most of our applications, in particular communication protocols, an asynchronous (buffered) semantics is more adequate, which is under consideration also for future versions of AutoFocus. Using an alternative semantics implies that we cannot make use of the simulation, code generation and model checking capabilites of current AutoFocus and its back-ends. Yet this is not a real obstacle for us since we are interested mainly in its graphic capabilities and the offered specification syntax is general enough to cover also our deviating semantics.

3 The ISM Approach

We first introduce the core of our modeling and verification framework, viz. ISMs, and then give some general comments how they are handled by AutoFocus and Isabelle.

3.1 Interacting State Machines

An *Interacting State Machine (ISM)* is an automaton whose state transitions may involve multiple input and output simultaneously on any number of ports. As the name suggests, the key concepts of ISMs are states (and in particular the transitions between them) and interaction. By interaction we mean explicit buffered communication via named ports, where on each of them one receiver listens to possibly multiple senders. A *system* consists of the parallel composition of any number of ISM components where the state of the whole system is essentially the Cartesian product of the states of its components.

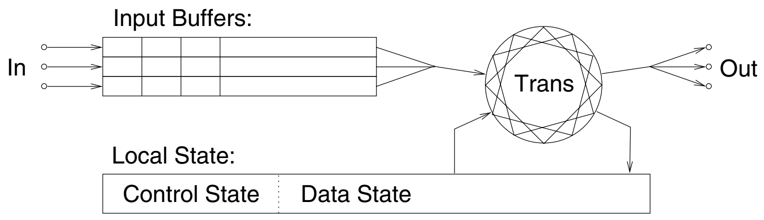


Fig. 1. ISM structure

The *state* of an ISM consists of the local state and its input buffers. The *local state* may have arbitrary structure but typically is the Cartesian product of a *control state* which is of finite type, and a *data state* that typically is a record of named fields. Transitions between states may be nondeterministic and can specified in any relational style. Thus the user has the choice to define them in an operational (i.e., executable) or axiomatic (i.e., property-oriented) fashion or a mixture of the two.

Each ISM declares two sets of port names, one for input and the other for output. The local input buffers are a family of unbounded FIFOs indexed by port names. Input of individual messages is triggered by any ISM and cannot be blocked, i.e. may occur at any time, appending the received value to the corresponding FIFO. Values stored in the input buffers may be processed by the ISM when it is ready to do so. This is done in a transition specified as follows: under a given precondition, the ISM consumes as much input from the buffers as appropriate, makes a transition of the local state, and produces output values at its discretion. These values are forwarded to the input buffers of all ISMs listening to the respective port, which may include feedback to the current component.

Each ISM has a single[1] initial state with empty input buffers. The execution of a system is a finite (but unbounded) sequence of nondeterministically interleaved steps of any of its components. Finiteness implies that we can handle safety, but not liveness properties. Transitions of different ISMs are related only by the causality wrt. the messages interchanged. Execution gets stuck when there is no component that can perform any step. As typical for reactive systems, there is no built-in notion of final or "accepting" states.

The representation of ISMs and their semantics consists of several layers. Its details, as well as a translation to IOAs, may be found in [Ohe02a].

[1] If a non-singleton set of initial states is required, these may be simulated by spontaneous nondeterministic transitions originating from a single dummy initial state.

3.2 AutoFocus Representation

By design, ISMs have almost the same structure as the automata definable with AutoFocus, and thus we can use AutoFocus as a graphical front-end to our Isabelle implementation.

In a typical application of our framework, ISMs are first "painted" using AutoFocus, saved in the so-called *Quest* file format, and then translated into suitable Isabelle theory files by a tool program.

3.3 Isabelle Representation

ISMs can be defined in special sections of Isabelle theories. This abstract (and almost semantics-independent) representation has essentially a one-to-one correspondence to the AutoFocus representation. As the cases studies below show, the structure of this section is almost self-explanatory. The formal definition of both the syntax and the semantics of ISMs in Isabelle/HOL is given in [Ohe02a].

4 LKW Model for the Infineon SLE66

We give an improvement of the LKW model for the Infineon SLE66 SmartCard processor. We demonstrate that, with the ISM approach, transition systems can be adequately modeled and their security properties stated and proven.

The *LKW model* [LKW00] is one of the first formal models for security properties of hardware chips. It has been used successfully within the security evaluation process for the SLE66 on ITSEC level E4 and the corresponding Evaluation Assurance Level 5 (*semiformally designed and tested*, which includes a formal security model) [CC99]. Recently, a slight extension was introduced [OLW02] in order to reflect additional application-oriented security objectives as defined in the SmartCard IC Platform Protection Profile [AETS01].

The LKW model gives an abstract system model for the SLE66 based on an ad-hoc automaton formalism, formalizes the security requirements in terms of properties of automaton runs and proves that the system meets the given requirements. All this is done as a pen-and-paper work, i.e. without tool assistance. Thus it is inevitable that the model contains many (mostly minor) syntactical, typographical and semantical slips as well as type errors, but also omissions like missing assumptions and incomplete proofs. Therefore it was desirable to formalize the model in a machine-checked way, applying a well-developed meta theory. Using ISMs, the LKW model can be represented adequately, including some improvements.

4.1 AutoFocus Diagrams

On the abstract level of the LKW model, the system architecture of the SLE66 is rather trivial: there is one component with one input port named `In` and one output port named `Out`, as depicted by Figure 2. The data state of the

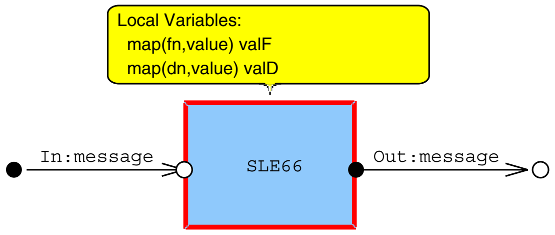


Fig. 2. SLE66 System Structure Diagram

component consists of two stores mapping names of functions and data objects to corresponding values.

Much more involved is the structure of the state transitions. There are four control states corresponding to the *phases* of the SLE66 life cycle:

Phase *0*: construction of the chip
Phase *1*: upload of Smartcard Embedded Software, personalization
Phase *2*: normal usage
Phase *Error*: locked mode from which there is no escape

In order to keep the state transition diagram clear, Figure 3 contains all control states and transitions, but instead of showing the preconditions, inputs, outputs, and changes to the data state, we just label the transitions with the names of the corresponding transition rules. Part of the rules are described in

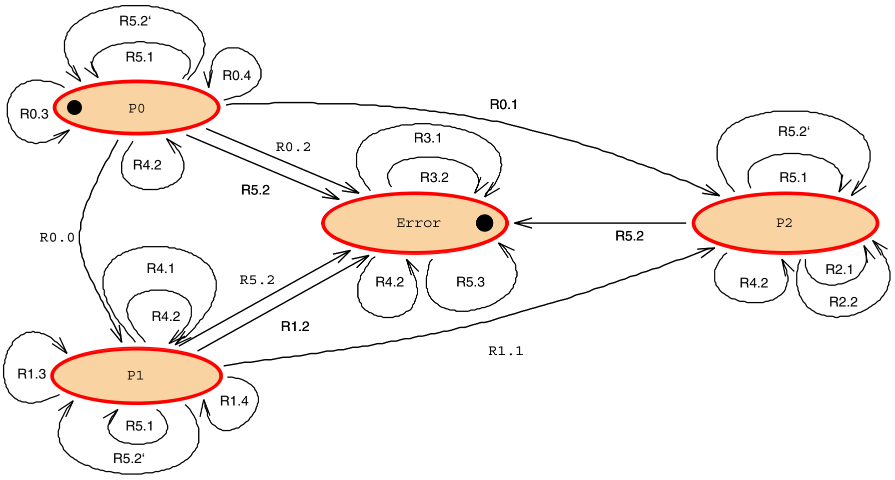


Fig. 3. SLE66 State Transition Diagram

detail in the next subsection. Here we give an informal general description of the transitions.

R0.0 thru R0.4 describe the execution of functions in the initial phase 0. Only the processor manufacturer is allowed to invoke functions in this phase and the required function must be enabled.
R0.0 states that if the function belongs to class `FTest0` and the corresponding test succeeds, phase 1 will be entered, and the test functions of that class are disabled.
R0.1 describes a shortcut leaving out phase 1: if the function belongs to class `FTest1` and the test succeeds, phase 2 will be entered, and all test functions are disabled.
R0.2 states that if a test fails, the system will enter the error state.
R0.3 models the successful execution of any other (enabled) function, in which case the function may change the chip state and yield a value.
R0.4 states that in all remaining cases of function execution the chip responds with `No` and its state is unchanged.
R1.1 thru R1.4 describe the execution of functions in the upload phase 1 analogously to R0.1 thru R0.4.
R2.1 and R2.2 describe the execution of functions in the usage phase 2 analogously to R0.3 and R0.4.
R3.1 and R3.2 describe the execution of functions in the error phase analogously to R0.3 and R0.4, except that the only function allowed to be executed in this phase is chip identification.
R4.1 and R4.2 describe the effects of a specific operation used for uploading new (operating system and application) functionality on the chip. This must be done by subjects trusted by the processor manufacturer and is allowed only in phase 1.
R4.1 describes the admissible situations, and
R4.2 describes all other cases.
R5.1 thru R5.3 describe the effects of attacks. Any attempts to tamper with the chip and to read security-relevant objects via physical probing on side channels (by mechanical, electrical, optical, and/or chemical means), for example differential power analysis or inspecting the silicon with a microscope, are modeled as a special "spy" input. Note that modeling physical attacks in more detail is not feasible because this would require a model of physical hardware. In particular, the conditions (and related mechanisms) under which the processor detects a physical attack is beyond the scope of the model.
R5.1 describes the innocent case of reading non-security-relevant objects in any regular phase, which actually reveals the requested information.
R5.2 describes the attempt to reading security-relevant objects in any regular phase. The chip has to detect this and enters the error phase, while the requested object may be revealed or not. This concept is called "destructive reading": one cannot rule out that attacks may reveal information even about security-relevant objects, but after the first of any such attacks, the processor hardware will be "destroyed", i.e. cannot be used regularly.
R5.3 states that in the error phase no (further) information is revealed.

4.2 Isabelle Theory

Next we give the Isabelle/HOL representation of the SLE66 model. For lack of space, of course we can show only the most essential definitions and a very small selection of the transition rules. Yet we do describe the slight extension mentioned above. For a detailed formal description of the LKW model see [LKW00]. The full Isabelle definitions are contained in [Ohe02b].

Objects stored on the chip may be either functions or data and are referred to by object names:

datatype *on = F fn | D dn*

Objects are classified as security-relevant (demanding secrecy and integrity) by including their names in the sets *F_Sec* or *D_Sec*, whose union is called *Sec*. In order to meet the additional requirements of [AETS01], the domain of security relevant functions *F_Sec* of the original LKW model has been refined to the disjoint union of *F_PSec* and *F_ASec*, which control the protection of the processor and application functionality, respectively.

The four control states of the SLE66 are defined as

datatype *ph = P0 | P1 | P2 | Error*

The data state consists of two fields, a function store and a data store:

```
record data =
  valF :: "fn ⇀ val"
  valD :: "dn ⇀ val"
```

The initial data state is declared but not actually defined. This is a typical example of underspecification, an important modeling technique.

const *s0 :: data*

We need only two port names, one for input and one for output:

datatype *interface = In | Out*

Possible input to the chip consists of either the two kinds of regular input messages (modeling function execution and load commands to the SLE66), or the *Spy* operation. The chip may respond with a value or a status message indicating success or failure.

```
datatype message =
  Exec sb fn | Load sb fn val | Spy on
| Val val | Ok | No
```

The subjects *sb* performing regular operations identify themselves to the chip via physical means. The actual authentication mechanism, as well as many other implementation details, is beyond the scope of this article.

Having defined its various parameters, we can now give the new **ism** theory element that specifies the SLE66 model:

```
ism SLE66 =
    ports interface
    inputs    "{In}"
    outputs   "{Out}"
    messages  message
  state
    control P0 :: ph
    data    s0 :: data
  transitions

R0.0: P0 -> P1
        pre "f ∈ fct s∩FTest0", "test f s"
        in   In  "[Exec Pmf f]"
        out  Out "[Ok]"
        eff  "valF := valF s⌊-FTest0"
```

Rule R0.0 specifies execution of a test function *f* by the processor manufacturer *Pmf* from the initial phase *P0*: if the test is successful then the SLE66 enters the next phase *P1*, disables the test functions *FTest0*, and answers with *Ok*. This rule is typical for interactions with the SLE66 in the sense that a single input triggers a single output. Note that the direct relation of input and output is expressed easily using ISMs, whereas using IOAs, two transitions would be required whose relation would be cumbersome to express and to use during verification.

```
R5.2: ph -> Error
        pre "ph ≠ Error", "oname ∈ Sec",
            "v ∈ {[],[Val (the (val s oname))]}"
        in   In  "[Spy oname]"
        out  Out "v"
        eff  "valF := fs, valD := ds"
```

Rule R5.2 specifies the typical reaction of the SLE66 upon attacks trying to read (the representation of) a secret object: The desired value may be output or not, but in any case the *Error* phase is reached. Note that R5.2 is a generic transition from any regualar phase to the *Error* phase. Furthermore, two sorts of nondeterminism are involved: *v* denotes either the empty output or the singleton output giving the desired value, and the attack may corrupt the function and data stores arbitrarily.

In contrast to the original LKW model, the *Load* operation may upload not only non-security-relevant functions but also functions of the application security domain, as long as they are not overwritten:

```
R4.1: P1 -> P1
        pre "f ∈ F_NSec ∪ (F_ASec - fct s)"
        in   In  "[Load Pmf f v]"
        out  Out "[Ok]"
        eff  "valF := valF s(f↦v)"
```

All remaining transition rules and further details on the system model may be found in [LKW00] and [Ohe02b].

4.3 Properties

The original security objectives for the SLE66 were stated as follows.

SO1. "The hardware must be protected against unauthorised disclosure of security enforcing functionality."
SO2. "The hardware must be protected against unauthorised modification of security enforcing functions."
SO3. "The information stored in the processor's memory components must be protected against unauthorised access."
SO4. "The information stored in the processor's memory components must be protected against unauthorised modification."
SO5. "It may not occur that test functions are executed in an unauthorised way."

Later, an additional requirement concerning the confidentialiy and integrity of Smartcard Embedded Software, which is not part of the security enforcing functionality, has been added [AETS01, §4.1].

Having defined the SLE66 system model, these informal statements can now be expressed formally as predicates on the system behavior, describing unambiguously and in detail which states may be reached under which circumstances, which data may be modified, and which output may appear on the output channel.

After formalizing the security objectives, it is natural to ask if the chip behavior, as specified in the system model, actually fulfills these requirements. The corresponding proofs have been conducted first using pen and paper, as reported in [LKW00]. Within the ISM framework, we meanwhile have verified these properties even mechanically (and thus with maximal reliability) using Isabelle.

Due to the abstract specification style where the semantics of parts of the chip functionality is not fully specified, it turns out that in order to prove the properties, a few general axioms are required. These assert for example that security-relevant functions do not modify security-relevant functions:

```
Axiom1: "f∈fct s∩F_Sec ⟹ valF (change f s)⌊F_Sec = valF s⌊F_Sec"
```

In comparison to the version of this axiom in the original model, the scope of functions `f` has been extended from "initially available" to "security-relevant", reflecting the changes to rule `R4.1`. Part of the lemmas as well as the formalized security objective FSO2.1 change accordingly:

```
FSO21: "⟦((ib,(ph,s)),p,(ib',(ph',s'))) ∈ Trans; ph' ≠ Error;
         g ∈ fct s∩fct s'∩F_Sec⟧ ⟹ valF s' g = valF s g"
```

The proof of this property is — as usual — by induction on the construction of all runs the SLE66 ISM can perform, which boils down to a case distinction over all possible transitions. Most cases are trivial except for those where function execution may change the stored objects, which are described by the rules `R0.3`, `R1.3`, and `R2.1`. Here an argumentation about the invariance of security-relevant functions `g` is needed, which follows easily from `Axiom1` and `Axiom2` stating the analogous property for non-security-relevant functions `f`.

The third (and last) axiom introduced in the LKW model states that in phase 2, a non-security-relevant function may not "guess" or (accidentally) reveal security-relevant information.

When machine-checking the proofs contained in [LKW00] with Isabelle, we noticed that a fourth axiom was missing that makes an implicit but important assumption explicit: if a function object may be referenced in two (different) ways and one of them declares the function to be security-relevant, the other has to do the same. Such experience demonstrates how important machine support is when conducting formal analysis.

Another omission was that in the proof of the security objective FSO5 an argumentation about the accessibility of certain functions was not given in a rigorous way. We fix this by introducing an auxiliary property (where, as typical with invariants, finding the appropriate one is the main challenge) and proving it to be an invariant of the ISM:

```
 no_FTest_invariant :: "state ⇒ bool"
"no_FTest_invariant ≡ λ(ph,s).
 ∀f ∈ fct s. (ph = P1 ⟶ f ∉ FTest0) ∧ (ph = P2 ⟶ f ∉ FTest)"
```

Exploiting the invariant, we can prove the desired property easily:

```
FSO5: "⟦((ib,(_,s)),(p,_,(_,s'))) ∈ Trans; ib In = Exec sb f#r;
       f ∈ FTest⟧ ⟹ sb = Pmf ∨ p Out = [No] ∧ s' = s"
```

The Isabelle proofs of all six theorems formalizing the security objectives and the two lemmas required are well supported by Isabelle: each of them takes just a few steps, about half of which are automatic.

The formalization of the remaining security objectives as well as their proofs wrt. the system model may be found in [LKW00] and [Ohe02b].

5 Needham-Schroeder Public-Key Protocol

As an example of an interaction-oriented system modeled with ISMs, we take Lowe's fix of the *Needham-Schroeder public-key authentication protocol* [Low96], which we call *NSL*. The emphasis here is not to provide new insights to the protocol, but to take a well-known (and thus easy to compare) benchmark system in order to show that, using the ISM approach, not only high-level requirements analysis but also low-level analysis of distributed systems can be done in a both rigorous and elegant way.

5.1 AutoFocus Diagrams

The system consists of an agent called `Alice` aiming to establish an authenticated session with another agent called `Bob` in the presence of an `Intruder` according to the Dolev-Yao attacker model [DY83]. As will be motivated in §5.2, we introduce a server `NGen` generating nonces. The corresponding system structure diagram in Figure 4 shows the four components with their data state (reflecting the expectations of the two agents, the set of messages the intruder

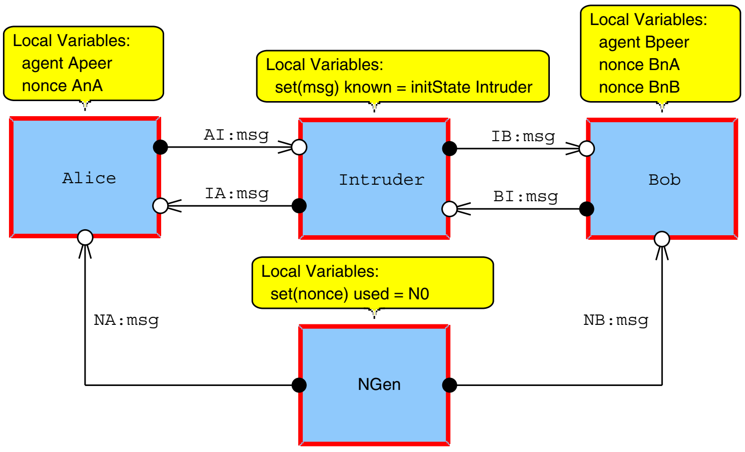


Fig. 4. NSL System Structure Diagram

knows of, and the set of already used nonces, respectively) and the named connections between them. Even if sometimes neglected, agents involved in communication protocols do have state: their current expectations and knowledge. This is made explicit in a convenient way by describing their interaction behavior with state transition diagrams. Figure 5 shows the three states of the agent `Alice` and the transitions between them, which have the general format `guard : inputs : outputs : assignments`.

In the initial state, `Alice` decides which agent she wants to talk to and sends the corresponding request. In the next state she awaits the response from the prospective peer before sending an acknowledgment. The third state represents (hopefully) successful session establishment. From the example of `Alice`'s tran-

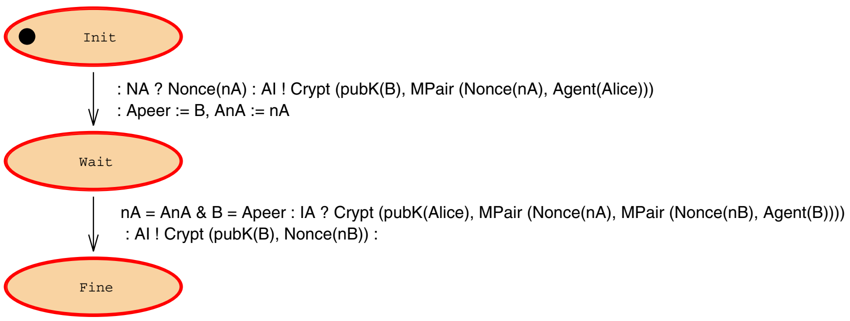

Init
: NA ? Nonce(nA) : AI ! Crypt (pubK(B), MPair (Nonce(nA), Agent(Alice)))
: Apeer := B, AnA := nA
Wait
nA = AnA & B = Apeer : IA ? Crypt (pubK(Alice), MPair (Nonce(nA), MPair (Nonce(nB), Agent(B))))
: AI ! Crypt (pubK(B), Nonce(nB)) :
Fine

Fig. 5. NSL State Transition Diagram: `Alice`

sitions we realize that control state information is the natural way to fix the order of protocol steps.

If the analysis needs to include the possibility that an agent takes part in more than one protocol run simultaneously, this can be modeled by multiple instantiation of the respective agent — under the assumption that from that agent's perspective the protocol runs are independent of each other.

5.2 Isabelle Theory

We base our ISM model on the formalization by Paulson [Pau98]. His so-called "inductive approach" is tailored to semi-automated verification of cryptographic protocols. Its great advantage is a high degree of automation, due to abstraction to the core semantics of the protocols: event traces. On the other hand, this makes both the models and the properties at least cumbersome to express: state information is implicit, yet often it has to be referred to, which is done by repeating suitable parts of the event history and sometimes even by introducing auxiliary events.

For lack of space, we do not show the definitions of the various state and message components since they are straightforward and analogous to the SLE66 model. Moreover, we show only the ISM definition of agent `Bob` as a typical representative.

```
ism Bob =
   ports channel
     inputs    "{NB,IB}"
     outputs   "{BI}"
     messages  msg
   state
     control Idle  :: B_control
     data    B0    :: B_data
   transitions
 Resp: Idle -> Resp
       in   NB "[Nonce nB]",
            IB "[Crypt (pubK Bob) {|Nonce nA, Agent A|}]"
       out  BI "[Crypt (pubK A) {|Nonce nA, Nonce nB, Agent Bob|}]"
       eff  "Bpeer := A, BnA := nA, BnB := nB"
 Ack': Resp -> Conn
       pre  "nB' = BnB s"
       in    IB "[Crypt (pubK Bob) (Nonce nB')]"
```

Note that `Bob`'s first transition `Resp` takes two inputs, from the nonce generator and the intruder, and produces one output. If we modeled this transition using IOAs, we would have needed three transitions with intermediate states. The precondition of transition `Ack'` could have been made implicit by moving the comparison as a pattern to the **in** part, yet we make it explicit in order to emphasize its importance. The local variable `BnB` serves to remember the value of the nonce expected, while the other two variables express Bob's view to whom he is connected in which session. In Paulson's approach, this state information is implicit in the event trace.

Modeling the freshness of nonces is intricate. In Paulson's model [Pau98], nonces are generated under the side condition that they do not already appear in the current event history. This criterion refers to the semantic and system-global notion of event traces — something not available from the (local) perspective of an ISM. We solve the problem by introducing a component called NGen that performs the generation of nonces for all agents in a centralized fashion. In this way we can ensure global freshness with a local criterion. Note that this component is just a modeling aid and thus its correct interplay with the agents does not need to be analyzed. We could alternatively express global freshness by adding an axiom restricting system runs in the desired way, yet we prefer the more constructive approach and derive the required freshness property as a lemma.

5.3 Properties

Properties of protocols specified with ISMs may be expressed with reference to both the state of agents and the messages exchanged. In the case of NSL, the most interesting property is authentication of Alice to Bob (actually, even session agreement [Low97] from Bob's view), which we formulate as

```
⟦Alice ∉ bad;  Bob ∉ bad;  (b,s)#cs ∈ Runs;
 Bob_state s = (Conn, ⦇Bpeer = Alice, BnA = nA, BnB = _⦈)⟧ ⟹
∃(_,s') ∈ set cs.
 Alice_state s' = (Wait, ⦇Apeer = Bob, AnA = nA⦈)
```

This can be quite intuitively read as: if in the current state *s* of the system Bob believes to be connected to Alice within a session characterized by the nonce nA then there is an earlier state *s'* where Alice was in the waiting state after initiating a connection to Bob using the same nonce nA.

It is interesting to compare the above formulation with the one given by Paulson:[2]

```
⟦A ∉ bad;  B ∉ bad;  evs ∈ ns_public;
 Crypt (pubK B) (Nonce NB) ∈ parts (spies evs);
 Says B A (Crypt (pubK A) ⦃Nonce NA,Nonce NB,Agent B⦄) ∈ set evs
⟧ ⟹
 Says A B (Crypt (pubK B) ⦃Nonce NA,Agent A⦄) ∈ set evs
```

This statement is necessarily more indirect since the beliefs of the agents have to be coded by elements of the event history. At least in this case, all messages of the protocol run have to be referred to. Note that this formulation makes stronger assumptions than ours because the value of the nonce NB is involved.

On the other hand, due to the extra detail concerning agent state and the input buffers (which are not actually required here), the inductive proofs within

[2] http://isabelle.in.tum.de/library/HOL/Auth/NS_Public.html

the ISM approach are more painful and require more lemmas on intermediate states of protocol runs than Paulson's inductive proofs.

6 Conclusion

The Interacting State Machines approach turns out to offer good support for formal security analysis in the way required within an industrial environment. ISMs are designed as high-level I/O automata, with additional structure and communication facilities. Like IOAs, ISMs are suitable for describing typical state-based communication systems relevant for security analysis, where ISM provide increased simplicity wrt. specifying component interaction via buffered communication and means to relate input and output actions directly.

The ISM approach offers graphical representation by means of AutoFocus System Structure Diagrams and State Transitions Diagrams. The graphical views are closely related to the formal system specification and verification via a tool translating the AutoFocus representation to an Isabelle theory.

We have shown that the ISM approach is equally applicable to a variety of security analysis tasks, ranging from high-level security modeling and requirements analysis, typically showing less system structure but increased complexity of state transitions, to security analysis of distributed systems including cryptographic protocols, likely to exhibit advanced system structuring. The examples explicate the importance of a fully formalized strategy, in particular, the LKW model has been significantly improved by identifying hidden assumptions and completing sloppy argumentation.

Further work on ISMs includes the extension of the proof support in the ISM level and the provision of a specification language based on temporal logic. Additional AutoFocus capabilities may be made available, including further systems views like event traces and simulation, as well as test case generation.

Acknowledgments We thank Guido Wimmel, Thomas Kuhn and some anonymous referees for their comments on earlier versions of this article.

References

[AETS01] Atmel, Hitachi Europe, Infineon Technologies, and Philips Semiconductors. Smartcard IC Platform Protection Profile, Version 1.0, July 2001. 217, 220, 222

[AGKS99] David Aspinall, Healfdene Goguen, Thomas Kleymann, and Dilip Sequeira. *Proof General*, 1999. 214

[But99] Michael Butler. csp2B : A practical approach to combining CSP and B. In *Proceedings of FM'99: World Congress on Formal Methods*, pages 490–508, 1999. `http://www.dsse.ecs.soton.ac.uk/techreports/99-2.html`. 214

[CC99] Common Criteria for Information Technology Security Evaluation (CC), Version 2.1, 1999. ISO/IEC 15408. 212, 217

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983. 223

[Fis00] Clemens Fischer. *Combination and implementation of processes and data: from CSP-OZ to Java*. PhD thesis, University of Oldenburg, 2000. 214

[GL98] Stephen J. Garland and Nancy A. Lynch. The IOA language and toolset: Support for designing, analyzing, and building distributed systems. Technical Report MIT/LCS/TR-762, Laboratory for Computer Science, MIT, August 1998. 213

[HSSS96] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. Autofocus - a tool for distributed systems specification. In *Proceedings FTRTFT'96 - Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 467–470. Springer-Verlag, 1996. See also `http://autofocus.in.tum.de/index-e.html`. 214, 215

[ITS91] Information Technology Security Evaluation Criteria (ITSEC), June 1991. 212

[JW01] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *Automated Software Engineering*. IEEE Computer Society, 2001. 214

[Kay01] Dilsun Kirli Kaynar. IOA language and toolset, 2001. `http://theory.lcs.mit.edu/tds/ioa.html`. 213

[LKW00] Volkmar Lotz, Volker Kessler, and Georg Walter. A Formal Security Model for Microprocessor Hardware. In *IEEE Transactions on Software Engineering*, volume 26, pages 702–712, August 2000. `http://www.computer.org/tse/ts2000/e8toc.htm`. 217, 220, 221, 222, 223

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer-Verlag, 1996. 223

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997. 226

[LT89] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. `http://theory.lcs.mit.edu/tds/papers/Lynch/CWI89.html`. 213

[Ohe02a] David von Oheimb. Interacting State Machines, 2002. Submitted for publication. 214, 216, 217

[Ohe02b] David von Oheimb. The Isabelle/HOL implementation of Interacting State Machines, 2002. Technical documentation, available on request. 220, 221, 223

[OLW02] David von Oheimb, Volkmar Lotz, and Gerog Walter. An interpretation of the LKW model according to the SLE66CX322P security target. Unpublished, January 2002. 217

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994. For an up-to-date description, see `http://isabelle.in.tum.de/`. 214

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998. 225, 226

[PNW$^+$] Lawrence C. Paulson, Tobias Nipkow, Markus Wenzel, et al. The Isabelle/HOL library. `http://isabelle.in.tum.de/library/HOL/`. 214

[S$^+$] Oscar Slotosch et al. Validas Model Validation AG. `http://www.validas.de/`. 215

[WW01] Guido Wimmel and Alexander Wisspeintner. Extended description techniques for security engineering. In M. Dupuy and P. Paradinas, editors, *International Conference on Information Security (IFIP/SEC)*. Kluwer Academic Publishers, 2001. 214

Decidability of Safety in Graph-Based Models for Access Control

Manuel Koch[1], Luigi V. Mancini[2], and Francesco Parisi-Presicce[2,3]

[1] Freie Universität Berlin, Berlin (DE)
`mkoch@inf.fu-berlin.de`
[2] Univ. di Roma La Sapienza, Rome (IT)
`{lv.mancini,parisi}@dsi.uniroma1.it`
[3] George Mason University, Fairfax VA (USA)
`fparisi@ise.gmu.edu`

Abstract. Models of Access Control Policies specified with graphs and graph transformation rules combine an intuitive visual representation with solid semantical foundations. While the expressive power of graph transformations leads in general to undecidable models, we prove that it is possible, with reasonable restrictions on the form of the rules, to obtain access control models where safety is decidable. The restrictions introduced are minimal in that no deletion and addition of a graph structure are allowed in the same modification step. We then illustrate our result with two examples: a graph based DAC model and a simplified decentralized RBAC model.

1 Introduction

Safety analysis determines, for a given set of policy rules and an initial state, whether or not it is possible to reach a state in which a particular access right is acquired by a subject that did not previously possess it. A system state is said to be *safe* with respect to an access right r if no sequence of commands can transform the state into a state that leaks r. Safety analysis was first formalized by Harrison, Ruzzo and Ullman [HRU76] in a model commonly known as HRU model. The HRU model captures security policies in which access rights may change and subjects as well as objects may be created and deleted [HRU76]. In general, safety of a system state with respect to an access right r is an undecidable problem, but decidability can be gained by restricting the models. For example, an authorization system is decidable if one considers mono-operational commands only [HRU76] or if the number of subjects is finite [LS78]. Sandhu and Suri have shown in [SS92] that their model of non-monotonic transformations (i.e., no deletion of access rights from the system), is subsumed by the model of [LS78] if object creation is possible but no subject creation. The approach of defining limited access control models with sufficient expressive power for practical use has not been followed recently because of complexity (both conceptual and algorithmic).

More recent research has focused on the alternative of defining constraint languages for the treatment of safety requirements. An example of a limited logical language is RSL99 [AS00]. Unfortunately, the languages proposed so far seem to lack either in simplicity or in expressive power.

We proposed a graph-based security framework to specify access control models [KMPP00, KMPP01a, KMPP01b] and we investigate in the present paper the safety issue of this graph-based framework. Compared to the HRU model, safety in our framework is decidable if each graph rule either deletes or adds graph structure but does not do both.

Graph rules do not have to be mono-operational and object creation is possible. Subject nodes chosen from a predefined finite set can be added to the system graph using the graph rules. The graph transformations do not have to be non-monotonical in the sense of [SS92].

There are other approaches [NO99, JT01] to the graphical representation of constraints in access control. The first defines a graphical model to combine role inheritance and separation of duty constraints. The second one uses a graphical model to express constraints based on set identification and set comparison. Neither paper deals with decidability of safety. In the context of the Take-Grant model, decidability is discussed in [Sny77]. The model presented is graph based, in the sense that the state is represented by a graph and state changes by graph transformations as rewriting rules. Safety is described as the (in)ability to derive, using the rewrite rules, a graph containing a specified edge.

In this paper, we illustrate our result with two simple examples: a graph based DAC model and a decentralized RBAC model. These examples are oversimplified in order to focus on the methodology to carry on the safety analysis. Note that the graph-based security framework proposed can specify also more expressive (and realistic) access control models for which the safety property remains decidable.

The result can be seen also as a new mechanism to compute safety in the presence of propositional constraints.

In section 2 of this paper, we review the basic notion of graph transformations; section 3 presents the decidability results and section 4 and section 5 show examples of decidability in a graph-based DAC model and a decentralized RBAC model, respectively. Section 6 contains a summary and points to future work.

2 Graph-Based Models

This section recalls some basic definitions and notation for graph transformations [Roz97]. A *graph* $G = (G_V, G_E(e)_{e \in ETyp}, t_V^G, l^G)$ consists of a set of nodes G_V, a relation $G_E(e) \subseteq G_V \times G_V$ for each edge type e in the set of edge types $ETyp$, a typing mapping $t_V^G : G_V \to VTyp$ and a label mapping $l^G : G_V \to VLabel$ for nodes. Node labels are elements of a disjoint union $VLabel = X \cup C$, where X is a set of variables and C is a set of constants. A reflexive binary relation $< \subseteq VLabel \times VLabel$ is defined as $\alpha < \beta$ if and only if $\alpha \in X$ or $\alpha = \beta$.

Figure 1 on the left-hand side depicts a graph with three nodes and three edges. The node types are U (for user) and O (for objects). There is an edge type for edges without any label and an edge type for edges with label r (for a read access right). The labels for nodes are the user names *Jackie* and *Thomas*, the object label is the object name *newProject.pdf*. We omit the explicit presentation of the typing and labeling morphisms in the figures and attach types and labels at the nodes and edges.

A *graph morphism* $f = (f_V, f_E(e)_{e \in ETyp}) : G \to H$ between graphs G and H is given by partial mappings $f_V : G_V \to H_V$ between nodes and $f_E(e) : G_E(e) \to H_E(e)$ for $e \in ETyp$ between edges so that

– f preserves the graph structure, i.e., $f_E(e)((v, v')) = (f_V(v), f_V(v'))$ for all $(v, v') \in dom(f_E(e))$ and $e \in ETyp$,
– f preserves types, i.e., $t^G_V(v) = t^H_V(f_V(v))$ for each $v \in dom(f_V)$ and
– f respects the label order, i.e. $l^G(v) < l^H(f_V(v))$ for each $v \in dom(f_V)$.

A graph morphism f is total (injective) if all underlying mappings f_V and $f_E(e)$ ($e \in ETyp$) are total (injective). Nodes in the domain of a graph morphism need not have variable labels. But if they do, they can be replaced by other variables or constants; if they are not variables, the labels must be preserved.

A *graph rule* $p(V) : (r, A(p))$ consists of a rule name p, a tuple of variables $V = (x_1, ..., x_n), x_i \in X$, a label preserving injective graph morphism $r : L \to R$ and a set $A(p)$ of *negative application conditions*. The graph L, *left-hand side*, describes the elements a graph must contain for p to be applicable. The morphism r is undefined on nodes/edges that are intended to be deleted, defined on nodes/edges that are intended to be preserved. Nodes and edges of R, *right-hand side*, without a pre-image are newly created. The actual deletions/additions are performed on the graphs to which the rule is applied. A Negative Application Condition (NAC) consists of a set $A(p)$ of pairs (L, N), where the graph L is a subgraph of N. The part $N \setminus L$ represents a structure that must not occur in a graph G for the rule to be applicable. In the figures, we depict (L, N) by the graph N, where the subgraph L is drawn with solid and $N \setminus L$ with dashed lines

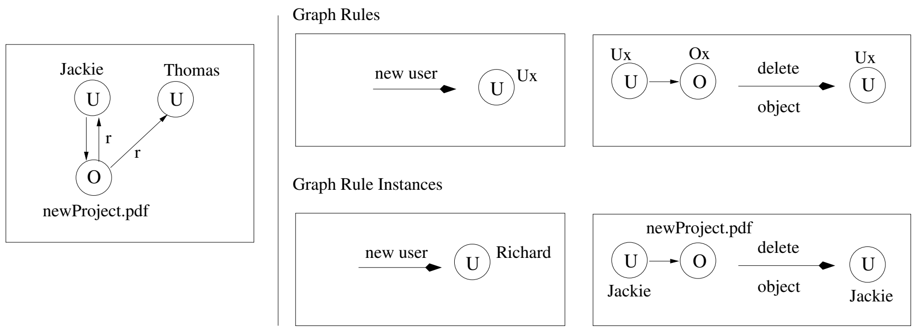


Fig. 1. An example of a graph (left), graph rules and graph rule instances (right)

(e.g., rule *remove user* in figure 4). A rule p with NAC $A(p)$ is applicable to G if L occurs in G and it is not possible to extend L to N for each (L, N) in $A(p)$.

A *rule instance* $p(c_1, ..., c_n) : (r_I, A(p))$ with $c_i \in C$ for a rule $p(x_1, ..., x_n) : (r, A(p))$ instantiates the rule variables x_i by constants c_i $(i = 1...n)$, so that the constants in the left-hand side and the right-hand side are unique.

Figure 1 on the right-hand side shows examples for rules and rule instances. The rule *new user* creates a new user, the rule *delete object* deletes an object which belongs to a user. The rules contain the variables Ux and Ox which can be instantiated by constants. The bottom of the figure shows two possible rule instances where, on the one hand, Ux is instantiated by *Richard* (rule *new user*), on the other hand Ux is instantiated by *Jackie* and Ox by *newProject.pdf*.

A *match* for a rule instance $p(C) : (r, A(p))$ in a graph G is a label preserving total graph morphism $m : L \to G$. The application of a rule instance $p(C) : (r, A(p))$ to a graph G is given by a match for p in G which satisfies the NAC $A(p)$. The direct derivation $G \overset{p,m}{\Rightarrow} H$ is given by the pushout of r and m in the category of graphs [Roz97]. The pushout is constructed by deleting all elements $m(L\backslash dom(r))$ and adding afterwards all elements $R\backslash r(L)$ to the part $m(dom(r))$. A *derivation sequence* $\rho = (G_0 \overset{p_0,m_0}{\Rightarrow} G_1 \overset{p_1,m_2}{\Rightarrow} G_2 \overset{p_2,m_2}{\Rightarrow} ...)$ is a sequence of direct derivations $G_i \overset{p_i,m_i}{\Rightarrow} G_{i+1}$. The length of a derivation sequence is the number of direct derivations it consists of. We denote a derivation sequence (possibly of length 0) from G_0 to G_n using rule instances of $\mathcal{R}$ by $G_0 \overset{\mathcal{R}}{\Rightarrow}_* G_n$.

Figure 2 depicts an example of a derivation sequence of length 2. First, a rule instance for the rule *delete object* is applied. The rule *delete object* specifies the deletion of an object which belongs to a user, in this rule instance, the deletion of the object *newProject.pdf* of user *Jackie*. We can find these graphical elements into graph G and delete the *newProject.pdf* node. Since no dangling edges must

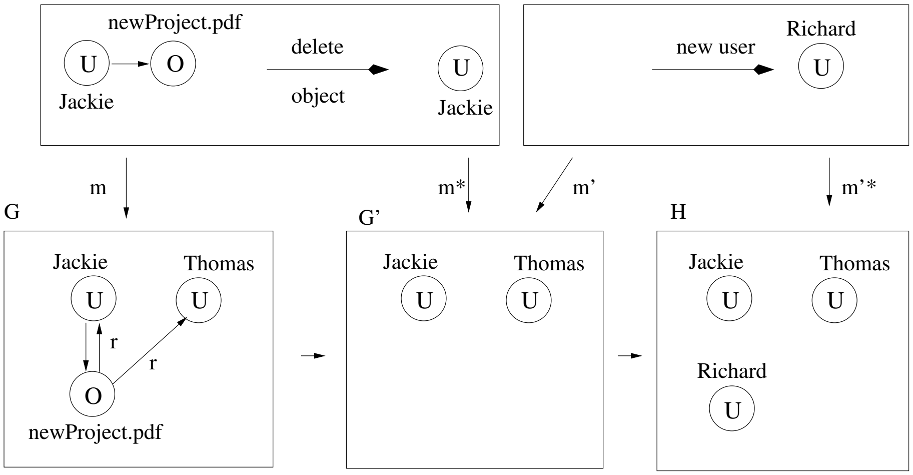


Fig. 2. A derivation sequence

occur in graphs, dangling edges are removed as well. In the second derivation step, a rule instance for the rule *new user* is applied to the result graph of the first derivation step. The rule *new user* adds a new user, in this instance the user *Richard*.

Graph transformations are the basic component for security policy frameworks used to model access control policies (a role-based model in [KMPP00], a lattice-based model and a discretionary model in [KMPP01a]).

3 Decidability in Graph Transformations

Definition 1 (safety). *Let Q_0 and A be graphs and $\mathcal{R}_I$ be a set of graph rule instances. The graph Q_0 is* safe *w.r.t. $\mathcal{R}_I$ and A if and only if there does not exist a graph Q so that $A \subseteq Q$ and $Q_0 \overset{\mathcal{R}_I}{\Rightarrow}_* Q$.*

In other words, an initial system state represented by Q_0 is safe, with respect to the access right represented by the graph A and a set of policy rule instances represented by $\mathcal{R}_I$, if and only if no derivation sequence exists from Q_0 to a graph Q such that $A \subseteq Q$, that is no sequence of graph rule instances can transform Q_0 in a state Q that leaks A.

Proposition 1. *Safety for graph transformations in general is undecidable.*

This is not surprising since graph transformation rules can model the rules of a type 0 grammar, but safety is decidable if each rule is either *expanding* or *deleting*. Deleting rules delete nodes or edges, but add nothing (i.e., $r(L) = R$) and expanding rules may add nodes and edges, but do not delete anything (i.e., $dom(r) = L \subset R$) and do not have a negative application condition.

Proposition 2 (upper bound). *Let Q_0 and A be graphs, $\mathcal{R}_I^+$ be a finite set of expanding graph rule instances and $Der = \{Q_0 \overset{\mathcal{R}_I^+}{\Rightarrow}_* Q | A \subseteq Q\}$ be the set of all derivation sequences starting at Q_0 that use the rule instances in $\mathcal{R}_I^+$ and ending in a graph Q which contains A. Then, the minimal derivation ρ^{min} in Der (i.e., for all $\rho \in Der$, the length of ρ is greater or equal the length of ρ^{min}) has an upper bound that depends only on $\mathcal{R}_I^+, Q_0$ and A.*

Proof. Let R_0 be the set of rule instances in $\mathcal{R}_I^+$ that are applicable to Q_0 and $M(p) = \{m : L \to Q_0 | m \text{ total}\}$ the set of all matches of rule instance $p \in R_0$ in Q_0. Furthermore, let R_f be the set of rule instances in $\mathcal{R}_I^+$ that construct parts of A, i.e., $(p : L \xrightarrow{r} R) \in R_f$ if and only if there is a nonempty injective partial graph morphism $h : R \to A$ so that $h(R \setminus r(L)) \cap A \neq \emptyset$. The set $R_f(p)$ contains all these partial morphisms h in A for the rule instance $p \in R_f$.

If all rule instances in R_f are applicable to a graph Q_f, the number of rule applications to construct graph A is at most $\sum_{p \in R_f} card(R_f(p))$[1]. Then, all

[1] $card(A)$ of a set A denotes the number of elements in A.

elements of A are constructed that can be constructed by the rule instances in $\mathcal{R}_I^+$ and one can decide whether A occurs or not.

We show now, that there is an upper bound for the number of rule applications to construct a graph Q_f (i.e., a graph to which all rule instances are applicable) from Q_0 by the rule instances in $\mathcal{R}_I^+$. We consider all paths $\sigma = (p_0 \overset{m_0}{\to} p_1 \to p_2 \to ...p_{n-2} \to p_{n-1})$, where $p_0 \in R_0$, $m_0 \in M(p_0)$, $p_i \in \mathcal{R}_I^+$ for each $0 \leq i < n$, so that each p_i occurs at most once. The paths have a length not greater than $card(\mathcal{R}_I^+)$. Let $\mathcal{P}$ be the set that contains all the paths σ.

A derivation $\rho = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \overset{p_{n-1},m_{n-1}}{\Rightarrow} Q_n)$ is a transformation of a path $\sigma = (p_0 \overset{m_0}{\to} p_1 \to p_2... \to p_{n-1})$, if it applies the rule instances in the order given in the path, starting at match m_0. Let Ω be the set of derivations transformed from the paths in $\mathcal{P}$. All derivations $\rho \in \Omega$ have the maximal length of $maxDerivation = card(\mathcal{R}_I^+)$.

$$\begin{tikzcd}
L_0 \arrow[r, "p_0:r_0"] \arrow[d, "m_0"'] & R_0 \arrow[dr, "m_0^*"'] & & L_1 \arrow[r, "p_1:r_1"] \arrow[dl, "m_1"] & R_1 \arrow[dr, "m_1^*"'] & & L_2 \arrow[r, "p_2:r_2"] \arrow[dl, "m_2"] & R_2 \arrow[d, "m_2^*"] & \cdots \\
Q_0 \arrow[rr, "r_0^*"] & & Q_1 \arrow[rrr, "r_1^*"] & & & Q_2 \arrow[rr, "r_2^*"] & & Q_3 & \cdots
\end{tikzcd}$$

We claim now: if a graph Q_f can be constructed by the rule instances in $\mathcal{R}_I^+$, there is a derivation $\rho = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \Rightarrow Q_f)$ in Ω. If we assume the opposite, there is a derivation $\rho' = (Q_0 \overset{p_0,m_0}{\Rightarrow} Q_1 \overset{p_1,m_1}{\Rightarrow} ... \overset{p_j,m_j}{\Rightarrow} Q'_f) \notin \Omega$, which constructs a graph Q'_f to which all rule instances are applicable. Then, the path $\sigma' = (p_0 \overset{m_0}{\to} p_1 \to p_2... \to p_j)$ can not be in $\mathcal{P}$, i.e., there must be at least one rule instance p_i that occurs more than once in the path σ'. In the diagram below, we assumed that the rule instance p_i is applied at position i and k.

$$\begin{tikzcd}
L_0 \arrow[r, "p_0:r_0"] \arrow[d, "m_0"'] & R_0 \arrow[d, "m_0^*"] \arrow[ddd, bend right] & L_1 \arrow[r, "p_1:r_1"] \arrow[dl, "m_1"] \arrow[dddl, bend left, "\tilde{m}_1"] & R_1 & \cdots & L_i \arrow[r, "p_i:r_i"] \arrow[d, "m_i"] & R_i \arrow[d, "m_i^*"] & \cdots & L_i \arrow[r, "p_i:r_i"] \arrow[d, "m_k"] & R_i \arrow[d, "m_k^*"] & L_{k+1} \arrow[r, "p_{k+1}:r_{k+1}"] \arrow[dl, "m_{k+1}"] \arrow[dddll, bend left, "\tilde{m}_{k+1}"] & R_{k+1} \arrow[d] & \cdots & L_j \arrow[r, "p_j:r_j"] \arrow[d, "m_j"] & R_j \arrow[d, "m_j^*"] \\
Q_0 \arrow[r, "r_0^*"] \arrow[d, "q_0"'] & Q_1 \arrow[d, "q_1"] \arrow[dd, bend left, "u_1"] & & & \cdots & Q_i \arrow[r, "r_i^*"] \arrow[d, "q_i"] & Q_{i+1} \arrow[d, "q_{i+1}"] \arrow[dd, bend left, "u_{i+1}"] & \cdots & Q_k \arrow[r, "r_k^*"] \arrow[d, "q_k"] \arrow[dd, bend left, "u_k"] & Q_{k+1} \arrow[rr] \arrow[d, "q_{k+1}"] & & Q_{k+2} \arrow[d] \arrow[dd, bend left, "u_{k+2}"] & \cdots & Q_j \arrow[r, "r_j^*"] \arrow[d, "q_{j-1}"] & Q'_f \arrow[d, "q_f"] \arrow[dd, bend left, "u_f"] \\
\bar{Q}_0 \arrow[r, "\bar{r}_0"] \arrow[d, "id"'] & \bar{Q}_1 & & & \cdots & \bar{Q}_i \arrow[r, "\bar{r}_i"] & \bar{Q}_{i+1} & \cdots & \bar{Q}_k \arrow[r, "\bar{r}_k"] & \bar{Q}_{k+1} \arrow[rr] & & \bar{Q}_{k+2} & \cdots & \bar{Q}_j \arrow[r, "\bar{r}_j"] & \bar{Q}'_f \\
Q_0 \arrow[r, "\tilde{r}_0"] & \tilde{Q}_1 & & & \cdots & \tilde{Q}_i \arrow[r, "\tilde{r}_i"] & \tilde{Q}_{i+1} & \cdots & \tilde{Q}_k \arrow[rrr] & & & \tilde{Q}_{k+2} & \cdots & & \tilde{Q}_f
\end{tikzcd}$$

In the derivation ρ', we identify in each graph Q_i $(0 \leq i \leq j)$ all nodes with the same label and get total surjective morphisms $q_i : Q_i \to \bar{Q}_i$. The sequence $(\bar{Q}_0 \to \bar{Q}_1... \to \bar{Q}'_f)$ is generally not a derivation sequence, since the diagrams $\bar{r}_i \circ$

$q_i \circ m_i = q_{i+1} \circ m_i^* \circ r_i$ are generally not pushout diagrams. But the identification ensures that $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i)$, so that in particular $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i) \subseteq \bar{Q}_{i+1} \subseteq \bar{Q}_k \subseteq \bar{Q}_{k+1}$.

We construct now the derivation sequence $\tilde{\rho} = (\tilde{Q}_0 \overset{p_0,\tilde{m}_0}{\Rightarrow} \tilde{Q}_1 \overset{p_1,\tilde{m}_1}{\Rightarrow} ...\tilde{Q}_{k-1} \overset{p_{k-1},\tilde{m}_{k-1}}{\Rightarrow} \tilde{Q}_k \overset{p_{k+1}}{\Rightarrow} \tilde{Q}_{k+2} \Rightarrow .... \Rightarrow \tilde{Q}_f)$, where $\tilde{Q}_0 = Q_0$, $\tilde{m}_0 = q_0 \circ m_0$ and $\tilde{m}_i = u_i \circ m_i$ where u_i is the universal pushout morphism w.r.t. the pushout diagram $m_i^* \circ r_i = r_i^* \circ m_i$ $(0 < i < j,\ i \neq k+1)$. We define $\tilde{m}_{k+1} = u_k \circ q_k^{-1} \circ q_{k+1} \circ m_{k+1}$, what is possible since q_k is surjective and $\bar{r}_k$ is the identity on Q_k, since $q_{i+1} \circ m_i^*(R_i) = q_{k+1} \circ m_k^*(R_i)$.

By assumption, there is a match $m_i : L_i \rightarrow Q_f$ for each rule instance $p_i : r_i \in \mathcal{R}_I^+$. Then, we have also a match $u_f \circ m_i : L_i \rightarrow \tilde{Q}_f$, i.e., all rule instances in $\mathcal{R}_I^+$ are applicable to $\tilde{Q}_f$. In such a way, we can remove each repeated rule instance application from ρ' and get a path $\tilde{\sigma} \in \Omega$, what is a contradiction to our assumption.

To conclude, the upper bound for constructing a graph Q_f from Q_0 using the rule instances in $\mathcal{R}_I^+$ in which all rule instances of R_f are applicable is $card(\mathcal{R}_I^+)$. The necessary rule applications for constructing A from Q_f are at most $\sum_{p \in R_f} card(R_f(p))$. Together, we get an upper bound of $card(\mathcal{R}_I^+) + \sum_{p \in R_f} card(R_f(p))$.

Theorem 1 (safety). *Safety of a graph Q_0 with respect to a graph A and a finite set of graph rule instances $\mathcal{R}_\mathcal{I}$ is decidable if $\mathcal{R}_\mathcal{I}$ contains only expanding and deleting graph rules.*

Proof. We show first that the minimal length derivation has only expanding rules. Suppose $Q_0 \overset{p_1}{\Rightarrow} Q_1 \overset{p_2}{\Rightarrow} ... \overset{p_n}{\Rightarrow} Q$ is a minimal length derivation reaching graph Q that contains A. Then, each rule instance p_i for $1 \leq i \leq n$ is an expanding rule. If we assume not, let p_j be a deleting rule. The removal of the rule p_j from the derivation does not affect the construction of the graph A in Q: If we remove p_j, we get the derivation sequence $Q_0 \overset{p_1}{\Rightarrow} Q_1 \overset{p_2}{\Rightarrow} ... \overset{p_{j-1}}{\Rightarrow} Q_{j-1} \overset{p_{j+1}}{\Rightarrow} Q'_{j+1} ... \overset{p_n}{\Rightarrow} Q'$. Since the graph Q could be constructed also without the structure $Q_{j-1} \setminus Q_j$ and expanding rules do not have application conditions forbidding some structure in a graph, the rule p_{j+1} can be applied to Q_{j-1} as well. Analog, the rules p_k for $k > j$ can be applied constructing a graph Q' containing A. Hence, we got a shorter derivation what is a contradiction to the assumption of a minimal length derivation sequence.

Thus, the minimal sequence for constructing a graph containing A starting at Q_0 contains only expanding rules. Proposition 2 shows that the minimal sequence to construct A has an upper bound m. A decision procedure would try all derivation sequences of expanding rules of length up to m.

Proposition 3. *Let A be a graph and $p_I(C) : L_I \xrightarrow{r_I} R_I$ be an instance rule of rule $p(V) : L \xrightarrow{r} R$. Furthermore, let*

- *$R_f(p)$ be the set of nonempty partial injective morphisms $g : R \rightarrow A$ so that $g(R \setminus r(L)) \cap A \neq \emptyset$ and*

– *$R_f(p_I)$ the set of nonempty partial injective morphisms $g : R_I \rightarrow A$ so that $g(R_I \setminus r_I(L_I)) \cap A \neq \emptyset$.*

Then, $card(R_f(p_I)) \leq card(R_f(p))$.

Proof. Bases on the fact that the variables in rules can be mapped to any appropriate constant, but constants in rule instances can be mapped only to the same constants: For each morphism in $R_f(p_I)$ there is a morphism in $R_f(p)$. A morphism in $R_f(p)$, however, does not need to have a counterpart on the instance level, since nodes labeled with a constant can be mapped only to nodes labeled with the same constant.

Corollary 1 (upper bound). *Let Q_0 and A be graphs, $\mathcal{R}^+$ a set of expanding rules, $\mathcal{R}_I^+(p)$ a finite set of rule instances for $p \in \mathcal{R}^+$ and $Der = \{Q_0 \overset{\mathcal{R}^+ I}{\Rightarrow}_* Q | A \subseteq Q\}$ be the set of all derivation sequences starting at Q_0 and ending in a graph Q which contains A. Then,*

$$UB = card(\mathcal{R}_I^+) + R_F$$

is an upper bound for the minimal derivation in Der, where $R_F = \sum_{p \in \mathcal{R}^+} card(R_f^+(p))$.

Proof. In the proof of proposition 2, the upper bound was given by

$$card(\mathcal{R}_I^+) + \sum_{p_I \in R_f} card(R_f(p_I)).$$

By proposition 3, we have

$$\sum_{p_I \in R_f} card(R_f(p_I)) \leq \sum_{p_I \in R_I^+} card(R_f(p_I)) \leq \sum_{p \in R^+} card(R_f(p)).$$

Figure 3 depicts the decidability algorithm.The method *setOfOverlaps(rule,graph)* returns the set of partial injective mappings from the right-hand side of the *rule* to the *graph*, where the domain of the mapping contains at least an element constructed by *rule* (i.e. in $R \setminus r(L)$). The method *setOfDerivationSequences(ruleSet,startGraph, length)* constructs all derivation sequences up to *length* starting from *startGraph* using rule instances of *ruleSet*. The method *card(set)* returns the number of elements in *set*. The method *subgraph(graph1,graph2)* returns *true* if *graph*1 is a subgraph of *graph*2, otherwise it returns *false*.

The decidability algorithm presented above can probably be optimized. The apparent complexity is exponential because it requires generating a set of overlaps and checking the subgraph property. The effective complexity of the algorithm is probably much lower than the computed worst case since labels (or unique names) on nodes reduce considerably the number of possible matchings. Graph transformation tools [EEKR99] can be used to automate the process of generating all the mappings and all the derivation sequences to be checked against the unwanted configuration.

input: sets $\mathcal{R}_I(p)$ ($p \in \mathcal{R}$) of expanding rule instances of rules in $\mathcal{R}$, graph Q_0 and graph A
output: true, if there is a derivation sequence ($Q_0 \Rightarrow ... \Rightarrow Q$) where $A \subseteq Q$, **false** otherwise

```
begin
  //get the rules which construct parts of $A$
  $R_F = 0$;
  for each $p \in \mathcal{R}$ {
    $R_f(p) = \text{setOfOverlaps}(p, A)$;
    $R_F = R_F + card(R_f(p))$;
  }
  //construct the upper bound for derivation sequences
  $length = \text{card}(R_I) + R_F$;
  $Der = \text{setOfDerivationSequences}(\mathcal{R}_I, Q_0, length)$;
  for each $(Q_0 \Rightarrow ... \Rightarrow Q) \in Der$
    if subgraph($A$,$Q$) then return true;
  return false;
end
```

Fig. 3. The decidability algorithm

4 Example: Decidability in a Graph-Based Model for Discretionary Access Control

We describe a graph-based DAC model for which the safety property is decidable. This simplified DAC model provides subject and object creation/removal and the granting of rights to access objects from one subject to another subject. The terms subject and user are synonymies in this section. For simplicity, the access rights considered are *read* and *write* only.

4.1 Graph Model for the DAC Policy

Users are represented by nodes of type U, objects to which users may have access by nodes of type O. Labels attached to user nodes specify the user name, labels attached to objects the object name. The variables used in rules are denoted by $Ux, Uy, Ox, Oy, ...$.

The rule *new user* in figure 4 introduces a user Ux. The creation of new objects (e.g., files and directories) on behalf of a user Ux is specified in the rule *new object*. Every object has an owner, namely the user who has created the object. The ownership of an object to a user is modeled by an edge from the user node to the object node. The owner of the new object has read and write access to the object. A permission on an object o for the owner is modeled by an edge from o to the owner with a label specifying the permitted access right (r for read and w for write). The owner of an object can simply remove the object (modeled in rule *delete object*). Users can be removed with rule *remove user*.

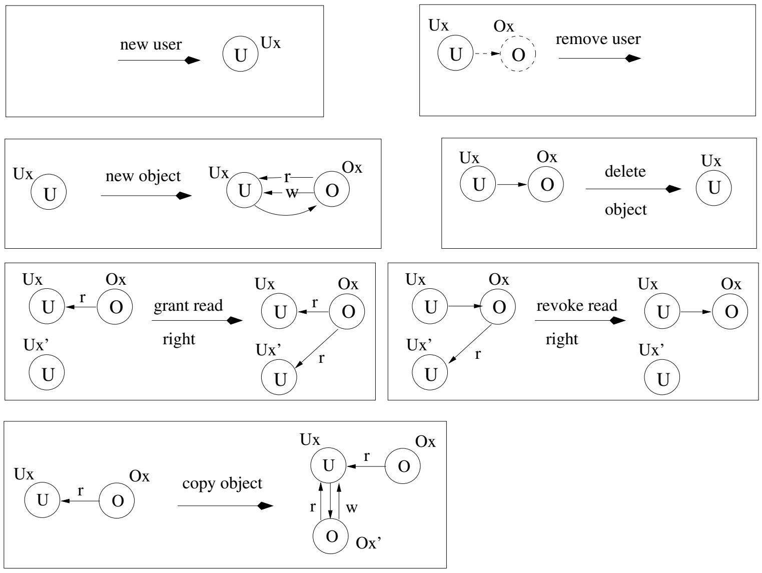


Fig. 4. The graph rules for the DAC model

To prevent objects without owner, the deletion of users is possible only if they do not own objects (specified in the NAC of rule *remove user* which forbids the application of *remove user* to a user who is connected to an object).

A user may grant its right on an object to other users. The rule *grant read right* specifies how a user Ux grants a read right on an object Ox (the right is modeled by an r-labeled edge from Ox to Ux) to another user Ux'. The r-labeled edge from Ox to Ux' specifies the read access right for Ux' on Ox. The rule for granting the write permission is analogously. Whereas an access right on an object can be granted by anybody who has the right, the revocation of a right on an object can be only executed by the object's owner. This is specified in rule *revoke read right*. The revocation of the write right is similar to rule *revoke read right*.

If a user Ux has read access to an object (i.e., there exists an edge with label r from the object to the user), and Ux copies the object, then Ux becomes the owner of the copy with read and write permission. The rule *copy object* specifies the copying of objects.

4.2 Safety of the DAC Model

This section carries on a safety analysis of the initial system state Q_0 shown in figure 5 with users *Richard*, *Jackie* and *Thomas*. *Jackie* is the owner of the object *newProject.pdf*. *Thomas* has read access to the object *newProject.pdf* and *Richard* has no right to access the object at all. *Jackie* has read and write access to the object because she is the object owner.

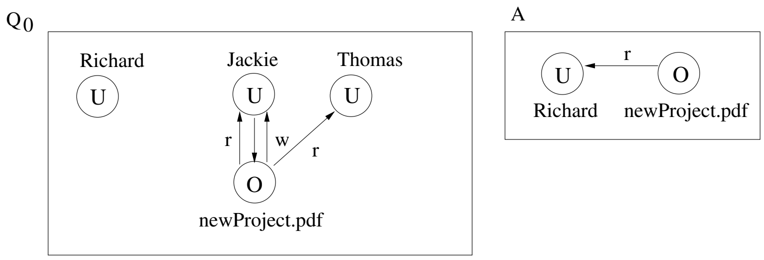


Fig. 5. An example for safety

Suppose that we would like to know now if it is possible for Richard to read *newProject.pdf* unbeknown to Jackie. That means, is it possible to construct a graph Q starting from Q_0 using the rules in figure 4 so that graph A of figure 5 is a subgraph of Q? Theorem 1 and corollary 1 state that it is sufficient to check all the derivation sequences of rule applications using rule instances of expanding rules up to a length of the upper bound $UB = card(\mathcal{R}_I^+) + R_F$. The expanding rules in figure 4 are: *new user*, *new object*, *grant read right* and *copy object*.

The number of rule instances $card(\mathcal{R}_I^+)$ depends on the (finite) set of lables for user names, where UN denotes the number of elements, and of the (finite) set of labels for object names, where we denote by ON the number of elements. For example, the number of rule instances for *new user* is equal to UN since *new user* has one variable for a user. The number of rule instances for rule *new object* is $UN \cdot ON$ since both a user and an object variable must be instantiated. The number of rule instances for rule *grant read right* is $UN \cdot (UN - 1) \cdot ON$, since two user and one object node must be instantiated. Since node labels for nodes of the same type must be unique in the left-hand side of the rule (as well as in the right-hand side), we have $UN \cdot (UN - 1)$ possibilities to instantiate the two user nodes with different labels. For the rule *copy object*, we get $UN \cdot ON \cdot (ON - 1)$ possible rule instances. Table 1 shows the number of possible instance rules.

The last column of table 1 depicts the number of partial nonempty injective mappings from the right-hand side R of rule p into the graph A, so that at least some newly created elements in R are mapped to A. By the information of this last column, the value of R_F can be calculated (cf. corollary 1).

Table 1.

	number of instances	$card(R_f(p))$
new user(Ux)	UN	1
new object(Ux,Ox)	$UN \cdot ON$	3
grant read right(Ux,Ox)	$UN \cdot (UN - 1) \cdot$ ON	1
copy object(Ux,Ox)	$UN \cdot ON \cdot (ON - 1)$	3

In this example, the upper bound is: $UN + UN \cdot ON + UN \cdot (UN - 1) \cdot ON + UN \cdot ON \cdot (ON - 1) + 8$. After having calculated the upper bound, only a finite number of derivation sequences with length less or equal than the upper bound are checked to decide the safety of the model. In particular, in this DAC example the derivation sequence which applies first the rule instance *copy object(Thomas,newProject.pdf)* to the initial state Q_0 and then the rule instance *grant read right(Thomas,Richard,newProject.pdf)* to the resulting graph constructs a graph which contains A. Therefore, the algorithm in figure 3 returns *true*, meaning that Q_0 leaks A.

5 Example: Decidability in a Graph-Based Model for Role-Based Access Control

The example in this section considers a simple graph model for decentralized RBAC (i.e., with more than one administrator for roles). In this model, where the safety property is decidable, a user is assigned to at most one role at a time and the assignment of a user to a role is static in the sense that it can only be changed by deleting the assignment and inserting a new one. The deletion of user-role assignment implies the loss of authorization for roles that were granted by the deleted assignment edge. This simplified RBAC graph model was originally presented in [KMPP00], and is reported in figure 6.

Users are created by the rule *add user* and are deleted by the rule *remove user*. A user can be assigned to a role if (s)he is not yet a member of a role. Membership is indicated by the color of the u node. A white u node indicates that the user is not yet in a role; a black one indicates that (s)he is a member of a role. The rule *add to role* turns a white user node to black when an administrator sets the assignment edge from the user to the role node. Since the roles authorization for each user is known, namely all junior roles reachable from the unique assignment edge, the deletion of a user-role assignment is simpler than in the general decentralized RBAC model. In particular, the deletion of the unique assignment edge ensures that the user does not have authorization for any role. In this case, all sessions are deactivated since all active roles for a session are authorized by the removed assignment edge, and the user node is changed to white.

A session is graphically presented by a node of type s and has always a connection to one user. The rules for the creation and deletion of sessions are *new session* and *remove session*. A session can be deleted at any time regardless of the presence of active roles of the session. The session is deleted by deleting the session node. This implies that all session-to-role assignments are deleted as well.

A user may activate any role r for which she/he is authorized. A user is authorized for r if there is a path starting from an assignment edge and ending in r. The corresponding graph rule is *activate role*. Rule *deactivate role* specifies the deactivation of a role from a session by deleting the edge between the session and the role node.

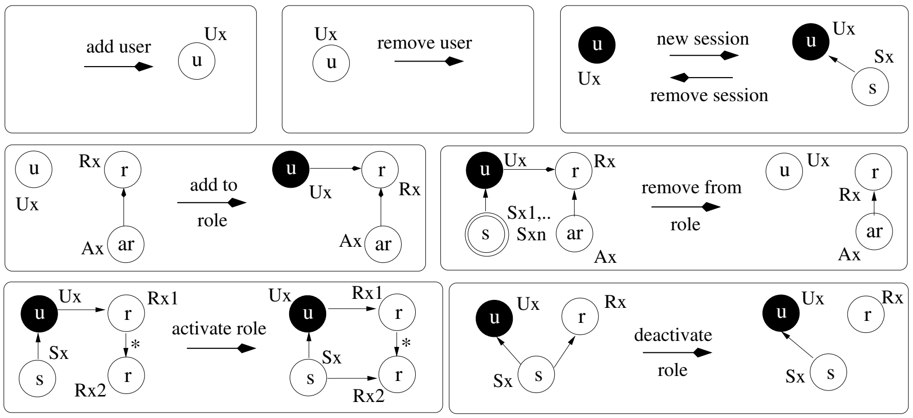


Fig. 6. Graph rules for the RBAC model

5.1 Safety of the RBAC Model

Suppose that we want to know if the initial state Q_0 in figure 7 can leak A, that is we would like to know whether *Elena* may play the role *President*. Table 2 shows the number of possible rule instances for the expanding rules of the RBAC example and the number of mappings from the right-hand side of the rules into the graph A. The number of rule instances depends on the finite label sets for user, session, role and administrator role variables. The label set for user includes the label *Elena*, the set of role labels is $\{President, ChiefManager, Manager\}$, the set of administrator role labels is $\{Bart, Anna\}$. We denote by UN the number of elements in the user label set, by SN the number of elements in the session label set, by $RN = 3$ the number of elements in the role label set and by $AN = 2$ the number of elments in the administrator role set.

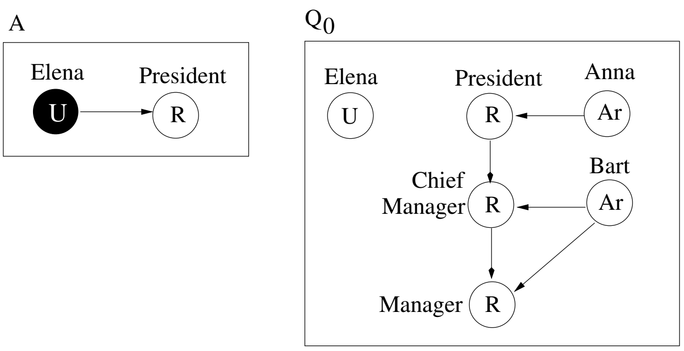


Fig. 7. A safety example for the RBAC model

Table 2.

	number of instances	$card(R_f(p))$
add user(Ux)	UN	1
new session(Ux,Sx)	$UN \cdot SN$	0
activate role(Ux,Rx1,Rx2,Sx)	$UN \cdot SN \cdot RN \cdot (RN-1)$	0
add to role	$UN \cdot RN \cdot AN$	1

The last column of table 2 depicts the number of nonempty partial injective mappings from the right-hand side R of the rule into graph A so that at least some newly created elements of R are mapped to A. Here, only the rules *add user* and *add to role* construct parts of A. After having calculated the upper bound, all derivation sequences with the given rule instances are checked up to the length given by the upper bound. In this example, the decidability algorithm returns *true* since the application of the rule instance *add to role(Elena, President, Anna)* constructs graph A.

Note that if we remove *Anna* from the administrator set AN (e.g., because we have confidence in *Anna*, but *Bart* must be checked explicitly), this RBAC example turns to be safe. Indeed, in the modified example, it will result that A cannot be constructed from Q_0. We will not list here all derivation sequences up to the upper bound, this can be automated with the help of a graph transformation tool [EEKR99]. However, we would like to informally convince the reader, that none of these derivation sequences constructs the graph A. The reason is that the rule *add to role(Elena,President,Anna)* cannot be instantiated, since the administrator label *Anna* is not in the label set AN for administrators. Indeed, only *Anna* can assign *Elena* to the president role, and *Elena* will never become president. Therefore, the decidability algorithm returns *false*.

6 Conclusion

The safety problem (can a state be reached in which a subject is granted a permission not previously possessed) in an arbitrary access control model based on graph transformations is in general undecidable.

This is not surprising since graph transformation rules can easily model the rules of a type 0 grammar. By imposing reasonable restrictions on the form of the rules, we have shown that the problem becomes decidable. The restrictions imposed still allow for an expressive framework as illustrated by the two examples of a graph-based DAC model and a graph-based single assignment decentralized RBAC. Moreover, it is possible in our framework to specify the Take-Grant model so that the rewrite rules *take, grant, create, remove* in [Sny77] satisfy the restrictions of our Theorem 1. Our notion of safety is more general than the one in [Sny77], since we test the presence of a general subgraph instead of a simple edge.

The safety problem "there exists a subject with a permission not previously possessed" can be handled, with a finite number of subjects, by defining a graph A for each specific subject. Alternatively, the problem can be handled by allowing the nodes in the graph A to be labelled with variables and checking if there exist an instantiation that is a subgraph of a state Q generated by the graph rules (similar to resolution in logic programming, shown in [CRPP91]).

The effective complexity of the decidability algorithm is probably much lower than the computed worst case since labels (or unique names) on nodes reduce considerably the number of possible matchings.

References

[AS00] G. Ahn and R. Sandhu. Role-based Authorization Constraint Specification. *ACM Trans. of Info. and System Security*, 3(4), 2000. 230

[CRPP91] A. Corradini, F. Rossi, and F. Parisi-Presicce. Logic programming as hypergraph rewriting. In *Proc. of CAAP91*, volume 493 of *LNCS*, pages 275–295. Springer, 1991. 243

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. II: Applications, Languages, and Tools.* World Scientific, 1999. 236, 242

[HRU76] M. A. Harrison, M. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976. 229

[JT01] Trent Jaeger and Jonathan E. Tidswell. Practical safety in flexible access control models. *ACM Trans. of Info. and System Security*, 4(2), 2001. 230

[KMPP00] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A Formal Model for Role-Based Access Control using Graph Transformation. In F.Cuppens, Y.Deswarte, D.Gollmann, and M.Waidner, editors, *Proc. of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, number 1895 in Lect. Notes in Comp. Sci., pages 122–139. Springer, 2000. 230, 233, 240

[KMPP01a] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the Specification and Evolution of Access Control Policies. In S. Osborne, editor, *Proc. 6th ACM Symp. on Access Control Models and Technologies*, pages 121–130. ACM, May 2001. 230, 233

[KMPP01b] M. Koch, L. V. Mancini, and F. Parisi-Presicce. Foundations for a graph-based approach to the Specification of Access Control Policies. In F.Honsell and M.Miculan, editors, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, Lect. Notes in Comp. Sci. Springer, March 2001. 230

[LS78] R. J. Lipton and L. Snyder. On synchronization and security. In Demillo et al., editor, *Foundations of Secure Computation*. Academic Press, 1978. 229

[NO99] M. Nyanchama and S. L. Osborne. The Role Graph Model and Conflict of Interest. *ACM Trans. of Info. and System Security*, 1(2):3–33, 1999. 230

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. I: Foundations.* World Scientific, 1997. 230, 232

[Sny77] L. Snyder. On the Synthesis and Analysis of Protection Systems. In *Proc. of 6th Symposium on Operating System Principles*, volume 11 of *Operating System Review*, pages 141–150. ACM, 1977. 230, 242

[SS92] Ravi S. Sandhu and Gurpreet S. Suri. Non-Monotonic Transformation of Access Rights. In *Proc. IEEE Symposium on Research and Privacy*, pages 148–161, 1992. 229, 230

Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones

Xinyuan Wang[1], Douglas S. Reeves[1,2], and S. Felix Wu[3] *

[1] Department of Computer Science
[2] Department of Electrical and Computer Engineering
North Carolina State University
{xwang5,reeves}@eos.ncsu.edu
[3] Department of Computer Science, University of California at Davis
wu@cs.ucdavis.edu

Abstract. Network based intrusions have become a serious threat to the users of the Internet. Intruders who wish to attack computers attached to the Internet frequently conceal their identity by staging their attacks through intermediate "stepping stones". This makes tracing the source of the attack substantially more difficult, particularly if the attack traffic is encrypted. In this paper, we address the problem of tracing encrypted connections through stepping stones. The incoming and outgoing connections through a stepping stone must be correlated to accomplish this. We propose a novel correlation scheme based on inter-packet timing characteristics of both encrypted and unencrypted connections. We show that (after some filtering) inter-packet delays (IPDs) of both encrypted and unencrypted, interactive connections are preserved across many router hops and stepping stones. The effectiveness of this method for correlation purposes also requires that timing characteristics be distinctive enough to identify connections. We have found that normal interactive connections such as telnet, SSH and rlogin are almost always distinctive enough to provide correct correlation across stepping stones. The number of packets needed to correctly correlate two connections is also an important metric, and is shown to be quite modest for this method.

1 Introduction

Network-based intrusions have become a serious threat to users of the Internet. Today, perpetrators can attack networked information systems from virtually anywhere in the world.

One major problem in apprehending and stopping network-based intrusion is that the intruders can easily hide their identity and point of origin through

* This work was supported by AFOSR contract F49620-99-1-0264 and by DARPA contract F30602-99-1-0540. The views and conclusions contained herein are those of the authors.

readily available means. One of the techniques most commonly used by intruders is to hide their origin by connecting across multiple stepping stones [4, 9, 13] before attacking the final targets. For example, an attacker logged into host A may telnet into host B, and from there launch an attack on host C. An analysis of the traffic at C will only reveal it is being attacked from B, but will not identify the actual source of the attack. A careful inspection of the contents of the traffic coming into and going out of B may reveal that A is the source of the attack. However, if the traffic arriving at B is encrypted (using SSH [6, 11] or IPSEC [3]) before being transmitted to C, it will not be possible to use the traffic contents for correlation purposes. Network-based intruders thus have an easy way to launch attacks without revealing their identity. Without a means of effectively and quickly tracing the source of an attack back to its source, it will not be possible to stop further attacks or punish those who are responsible.

In this paper, we address the problem of correlating the incoming and outgoing connections of a stepping stone. The goal is to identify which connections are part of an attack path, so that the attack can be traced back to its source. We assume that attack traffic may be encrypted at any stepping stone in an attempt to interfere with correlation. We propose a novel scheme based on the inter-packet timing characteristics of both encrypted and unencrypted connections. While, as with most intrusion tracing and detection systems, out correlation scheme could be evaded by highly sophisticated intruders, it is our goal to make it difficult to do so and thus deter network-based intrusions.

The remainder of the paper is organized as follows. In section 2, we give a summary of related works. In section 3, we formulate the correlation problem of our focus and give our correlation problem solution model. In section 4, we discuss IPD (Inter-Packet Delay) based correlation in detail. In section 5, we evaluate correlation effectiveness of our proposed correlation metrics through experiments. In section 6, we conclude with summary of our findings.

2 Related Work

Most of the existing work on correlating connections across stepping stones assumes the traffic is unencrypted. In general, attack tracing approaches can be categorized as either being host-based or network-based. In a host-based approach, the stepping stone itself participates in the tracing, while in the network-based approaches, the stepping stones are not used for tracing purposes. Based on how the traffic is traced, tracing approaches can further be classified as either active or passive. Passive approaches monitor and compare all traffic, allowing any traffic to be traced at any time. On the other hand, active approaches dynamically control when, where, what and how the traffic is to be correlated, through customized packet processing. They only trace the traffic of interest when needed. Table 1 provides a classification of existing tracing approaches, as well as our proposed tracing mechanism.

The Distributed Intrusion Detection System (DIDS) [8] developed at UC Davis is a host-based tracing mechanism that attempts to keep track of all the

Table 1. Classification of Correlation and Tracing Approaches

	Passive	Active
Host-Based	DIDS [8] CIS [2]	
Network-Based	Thumb printing [9] ON/OFF-Based [13] Deviation-Based [12]	IDIP [7] SWT [11] IPD-Based (proposed method)

users in the network and report all activities to network-wide intrusion detection systems. The Caller Identification System (CIS) [2] is another host-based tracing mechanism. It eliminates centralized control by utilizing a truly distributed model. Each host along the connection chain keeps a record about its view of the connection chain so far.

A fundamental problem with the host-based tracing approach is its trust model. Host-based tracing places its trust upon the monitored hosts themselves. In specific, it depends on the correlation of connections at every host in the connection chain. If one host is compromised and is providing misleading correlation information, the whole tracing system is fooled. Because host-based tracing requires the participation and trust of every host involved in the network-based intrusion, it is very difficult to be applied in the context of the public Internet.

Network-based tracing is the other category of tracing approaches. It does not require the participation of monitored hosts, nor does it place its trust on the monitored hosts. Rather, it is based on the assumption that one or more properties of a connection chain is maintained throughout the chain. In particular, the thumbprint [9] is a pioneering correlation technique that utilizes a small quantity of information to summarize connections. Ideally it can uniquely distinguish a connection from unrelated connections and correlate successfully those connections in the same connection chain.

Sleepy Watermark Tracing (SWT) [11] applies principles of steganography and active networking in tracing and correlating unencrypted connections through stepping stones. By injecting watermarks in the traffic echoed back to the attacker, SWT is able to trace and correlate even a single keystroke by the intruder. By actively generating tracing traffic, it can trace and correlate even when an intrusion connection is idle.

IDIP (the Intrusion Identification and Isolation Protocol) [7] is a part of Boeing's Dynamic Cooperating Boundary Controllers Program that uses an active approach to trace the incoming path and source of the intrusion. In this method, boundary controllers collaboratively locate and block the intruder by exchanging intrusion detection information, namely, attack descriptions. The ON/OFF-based scheme [13] by Zhang and Paxson is the first correlation intended to correlate traffic across stepping stones even if the traffic is encrypted by the stepping stone. The method is based on correlation of the ends of OFF periods (or equivalently the beginnings of ON periods) of interactive traffic, rather than the connection contents. While it is robust against payload padding, ON/OFF-based correlation requires that the packets of connections have precise, synchronized timestamps in order to be able to correlate them. This makes correlations of measurements taken at different points in the network difficult or impractical.

The deviation-based approach [12] by Yoda and Etoh is another network-based correlation scheme. It defines the minimum average delay gap between the packet streams of two TCP connections as the deviation. The deviation-based approach considers both the packet timing characteristics and their TCP sequence numbers. It does not require clock synchronization and is able to correlate connections observed at different points of network. However, it can only correlate TCP connections that have one-to-one correspondences in their TCP sequence numbers, and thus is not able to correlate connections where padding is added to the payload, e.g., when certain types of encryption are used.

Both ON/OFF-based and deviation-based approaches define their correlation metrics over the entire duration of the connections to be correlated. This makes the correlation applicable to post-attack traces only.

The IPD-based approach we discuss in the remainder of the paper defines its correlation metric over the sliding window of packets of the connections to be correlated. This enables it to correlate both live traffic (at real-time) and collected traces (post-attack). It supports distributed correlation of traffic measure at different points of network and is robust against payload padding.

3 Problem Formulation

A *connection* c_i (also called a flow) is a single connection from computer host H_i (the source) to host H_{i+1} (the destination). A user may log into a sequence of hosts H_1, H_2, … H_{n+1} through a connection chain c_1, c_2, … c_n, where connection c_i is a remote login from host H_i to host H_{i+1}.[1] The *tracing problem* is, given connection c_n, to determine the other connections c_1, c_2, … c_{n-1} in the chain, and only those connections. From these connections the identities of all the hosts in the chain, including H_1, may be directly determined.[2]

3.1 Correlation Problem Solution Model

Let $\hat{C}$ represent the set of all connections being examined. We can define an *ideal correlation function* $\mathrm{CF} : \hat{C} \times \hat{C} \rightarrow \{0, 1\}$ such that $\mathrm{CF}(c_i, c_j) = 1$ iff c_i

[1] The same host may appear in a connection chain more than once, in which case the chain contains a loop. Due to space limitations we do not consider this case here.

[2] If IP spoofing is used, of course, the packets of a connection will incorrectly identify the source of the connection. We consider this problem to be orthogonal to our problem and do not address it here. It is, in any event, unlikely that IP spoofing will be used for interactive traffic, where the response to the interactive traffic must be correctly echoed back to the source.

and c_j are in the same connection chain, otherwise $\mathrm{CF}(c_i, c_j) = 0$. To solve the tracing problem we must find such a function CF.

In practice, connection correlation is based on the characteristics of the connections, which may include packet contents, header information (such as packet size) and packet arrival and/or departure times. The connection characteristics can be modeled by a metric function of the connection

$$M : \hat{C} \times P \rightarrow Z \tag{1}$$

where $\hat{C}$ is the set of connections to be correlated, P is some domain of parameters and Z is the correlation metric domain. Based on the connection metric, a *correlation value function* (CVF) can be defined as

$$\mathrm{CVF} : Z \times Z \rightarrow [0, 1] \tag{2}$$

where the result of CVF is a real number between 0 and 1. To approximate CF through CVF, we introduce a threshold $0 \leq \delta \leq 1$ such that c_i and c_j are considered correlated iff

$$\mathrm{CVF}(M(c_i, p), M(c_j, p)) \geq \delta \tag{3}$$

Therefore the tracing problem is now replaced by the following: *find or construct M, p, CVF and δ such that*

$$\begin{gathered} \forall c_i, c_j \in \hat{C}, \mathrm{CVF}(M(c_i, p), M(c_j, p)) \geq \delta \\ \text{iff } c_i \text{ and } c_j \text{ are in the same connection chain} \end{gathered} \tag{4}$$

In finding M, p, CVF and δ, the key is to identify those unique characteristics of connections that are invariant across routers and stepping-stones. If those identified invariant characteristics of connections are distinctive enough to exclude other unrelated connections, reliable correlation of connections can be constructed from these metrics.

4 IPD Based Correlation of Encrypted Connections

In principle, correlation of connections is based on inherent characteristics of connections. To correlate potentially encrypted connections, the key is to identify a correlation metric from the connection characteristics that is: 1) invariant across routers and stepping stones; 2) not affected by encryption and decryption; 3) unique to each connection chain. Potential candidates for the correlation metric of a flow of packets include header information, packet size, inter-packet timing etc. In particular, inter-packet timing should not be affected by encryption and decryption. We now present an original correlation method based on inter-packet delays or IPDs.

4.1 General IPD Based Correlation Model

The overall IPD correlation of two connections is a two-step process. First, the two connections to be correlated are processed to generate a number of *correlation points* between the two connections. Second, these generated correlation points are evaluated to obtain the *correlation value* of the two connections.

The rationale behind this two-step process is to support the true real-time correlation, which is the capability to correlate "live" traffic when they come and go. This means that the approach must be able to correlate connections before their ends are reached. Therefore, the correlation metric for true real-time correlation cannot be defined over the entire duration of a connection; we choose instead to compute it over a window of packets in the connection. A correlation point generated from IPDs within the window reflects some localized similarity between the two flows; the correlation value obtained from all the correlation points will indicate the overall similarity of the two flows.

Basic IPD Correlation Concepts and Definitions Given a bi-directional connection, we can split it into two unidirectional flows. We define our correlation metric over the unidirectional flow of connections.

Given a unidirectional flow of $n > 1$ packets, we use t_i to represent the timestamp of the i^{th} packet observed at some point of the network. We assume all the t_i's of a flow are measured at the same observation point with the same clock. We define the i^{th} *inter-packet delay* (IPD) as

$$d_i = t_{i+1} - t_i \tag{5}$$

Therefore, for any flow consisting of $n > 1$ packets, we can measure the inter-packet delay (IPD) vector $\langle d_1, \ldots, d_{n-1} \rangle$. Ideally, the IPD vector would uniquely identify each flow and we could construct our correlation metric from the IPD vectors. To support real-time correlation based on the IPD vector, we define the *IPD correlation window* $W_{j,s}$ on $\langle d_1, \ldots, d_n \rangle$ as

$$W_{j,s}(\langle d_1, \ldots, d_n \rangle) = \langle d_j, \ldots, d_{j+s-1} \rangle \tag{6}$$

where $1 \leq j \leq n$ represents the starting point of the window, and $1 \leq s \leq n-j+1$ is the size of the window.

Given any two flows X and Y, whose IPD vectors are $\langle x_1, \ldots, x_m \rangle$ and $\langle y_1, \ldots, y_n \rangle$ respectively, we define a *Correlation Point Function* CPF over IPD correlation windows of $X : W_{j,s}(X)$ and of $Y : W_{j+k,s}(Y)$ as

$$\mathrm{CPF}(X, Y, j, k, s) = \phi(W_{j,s}(X), W_{j+k,s}(Y)) \tag{7}$$

where ϕ is a function of two vectors: $R^s \times R^s \to [0, 1]$, $1 \leq j \leq \min(m-s+1, n-k-s+1)$ is the start of the IPD correlation window, $-j+1 \leq k \leq n-j-s+1$ is the offset between the two IPD correlation windows, and $1 \leq s \leq \min(m, n)$ is the size of the two IPD correlation windows. $\mathrm{CPF}(X, Y, j, k, s)$ quantitatively

Flow X: $x_1, \ldots, \boxed{x_j, \ldots, x_{j+s-1},} \ldots, x_m$

Flow Y: $y_1, \ldots, \boxed{y_{j+k}, \ldots, y_{j+k+s-1},} \ldots, y_n$[!t]

Fig. 1. CPF in PD Correlation Windows $W_{j,s}(X)$ and $W_{j+k,s}(Y)$

expresses the correlation between $W_{j,s}(\langle x_1, \ldots, x_m \rangle)$ and $W_{j+k,s}(\langle y_1, \ldots, y_n \rangle)$ as shown in Figure 1.

Because the value of $\mathrm{CPF}(X, Y, j, k, s)$ changes as j and k changes, we can think of $\mathrm{CPF}(X, Y, j, k, s)$ as a function of j and k. Given any particular value of j, $\mathrm{CPF}(X, Y, j, k, s)$ may have a different value for each different value of k. We are interested in the maximum value of $\mathrm{CPF}(X, Y, j, k, s)$ for any particular value of j. We define $(j, j+k)$ as a correlation point if

$$\max_{-j+1 \leq k \leq n-j-s+1} \mathrm{CPF}(X, Y, j, k, s) \geq \delta_{cp} \tag{8}$$

where δ_{cp} is the correlation point threshold with value between 0 and 1. The δ_{cp} here is for detecting correlation point and is different from δ in inequality (3).

We further define k for this correlation point $(j, j+k)$ as the *correlation-offset* of $\mathrm{CPF}(X, Y, j, k, s)$ and the correlation point.

Given flow X, Y, correlation window size s and threshold δ_{cp}, by applying formula (8), we can obtain a series of correlation points: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ where $n \geq 0$.

Assuming one packet of flow X corresponds to one packet of flow Y[3], if flow X and Y are really part of the same connection chain, the IPDs of flow X should have a one-to-one correspondence with the IPDs of flow Y. In this case, all the correlation points should have same correlation-offset. This can be formally represented with CPF as

$$\exists k' \forall j \left[\mathrm{CPF}(X, Y, j, k', s) = \max_{-j+1 \leq k \leq n-j-s+1} \mathrm{CPF}(X, Y, j, k, s) \right] \tag{9}$$

That is there exists an offset k' such that $\mathrm{CPF}(X, Y, j, k', s)$ is closest to 1 for all possible j. In this case, all the correlation points (x, y) will be covered by a linear function $y = x + k'$.

After obtaining $n > 0$ correlation points: $(j_1, j_1+k_1), (j_2, j_2+k_2), \ldots, (j_n, j_n+k_n)$, we represent those n correlation points with two n-dimensional vectors $C_x = \langle j_1, \ldots, j_n \rangle$ and $C_y = \langle j_1 + k_1, \ldots, j_n + k_n \rangle$. The Correlation Value Function formula (3) is now transformed to

$$\mathrm{CVF}(C_x, C_y) \geq \delta \tag{10}$$

In summary, the metric function M in formula (1) is now mapped to CPF, the parameter domain P in formula (1) is mapped to s and δ_{cp}, and Z in formula (1) is mapped to n-dimensional vectors $C_x = \langle j_1, \ldots j_n \rangle$ and $C_y = \langle j_1 + k_1, \ldots, j_n + k_n \rangle$.

[3] We have found this is true for most packets in correlated flows.

Correlation Method Assessment Criteria A critical issue in this method is the choice of the function ϕ for computing the correlation point function CPF. Criteria by which the method may be judged include:

- Uniqueness of perfect correlation: for a flow X, no flow Y not in the same connection chain as X should have $\mathrm{CPF}(X, Y, j, k, s) = 1$.
- Correlation Point (CP) *true positives (hits)*: this is the number of packets that should be correlated, and are found to be correlated according to equation 8. The true positive rate is the number of hits divided by the number of packets that should be correlated.
- Correlation Point (CP) *false positives (misses)*: this is the number of packets which are not in fact correlated, but which are found to be correlated according to equation 8.

Ideally, we would expect a perfect correlation method 1) has unique perfect correlation; 2) has a 100% CP true positive rate; and 3) has 0 misses or false positives.

4.2 Correlation Point Functions

We now propose four correlation point functions, each of which enjoys certain advantages or applicability, as discussed below.

Min/Max Sum Ratio (MMS) One simple metric to quantitatively express the ôsimilarityö between two vectors is the ratio between the summation of the minimum elements and the summation of the maximum elements.

$$\mathrm{CPF}(X, Y, j, k, s)_{MMS} = \frac{\sum_{i=j}^{j+s+1} \min(x_i, y_{i+k})}{\sum_{i=j}^{j+s+1} \max(x_i, y_{i+k})} \tag{11}$$

$\mathrm{CPF}(X, Y, j, k, s)_{MMS}$ has the range $[0, 1]$ and takes the value 1 only when $x_i = y_{i+k}$ for $i = j, \ldots, j+k-1$. Therefore, $\mathrm{CPF}(X, Y, j, k, s)_{MMS}$ is likely to exhibit unique perfect correlation.

Statistical Correlation (STAT) Based on the concept of the coefficient of correlation from statistics [1], we can define

$$\mathrm{CPF}(X, Y, j, k, s)_{Stat} = \begin{cases} \rho(X, Y, j, k, s) & , \rho(X, Y, j, k, s) \geq 0 \\ 0 & , \rho(X, Y, j, k, s) < 0 \end{cases} \tag{12}$$

where

$$\rho(X, Y, j, k, s) = \frac{\sum_{i=j}^{j+s+1} (x_i - E(X)) \times (y_{i+k} - E(Y))}{\sqrt{\left[\sum_{i=j}^{j+s+1} (x_i - E(X))^2\right] \times \left[\sum_{i=j}^{j+s+1} (y_{i+k} - E(Y))^2\right]}}$$

The range of $\mathrm{CPF}(X, Y, j, k, s)_{Stat}$ is also $[0, 1]$. Unlike $\mathrm{CPF}(X, Y, j, k, s)_{MMS}$, for a given $W_{j,s}(X)$, there may be more than one value of $W_{j+k,s}(Y)$ for which $\mathrm{CPF}(X, Y, j, k, s)_{Stat}$ has the value 1. For example, for a particular $W_{j,s}(X)$, any linear transform of $W_{j,s}(X) : W_{j+k,s}(Y) = a \times W_{j,s}(X) + b$ will result in $\mathrm{CPF}(X, Y, j, k, s)_{Stat}$ being equal to 1 ($a > 0$) or -1 ($a < 0$). $\mathrm{CPF}(X, Y, j, k, s)_{Stat}$ is therefore less likely to exhibit unique perfect correlation, and is more likely to result in false positives.

Normalized Dot Product 1 (NDP1) In digital signal processing, linear correlation (or matched filtering) of two discrete signals will reach a maximum at the point where the signals have the most similarity. It is well known that linear correlation is optimal in detecting the similarity between a discrete signal and the corresponding signal distorted by additive, white Gaussian noise. However the range of linear correlation is not necessarily between 0 and 1.

If the discrete signals are replaced by two vectors, the corresponding operation to linear correlation of signals is the *inner-product* or *dot-product* of two vectors in n-dimensional space. From linear algebra, the inner-product of two n-dimensional vectors is equal to the cosine of the angle between the two vectors, multiplied by the lengths of the two vectors. That is:

$$W(X) \bullet W(Y) = \cos(\theta) \times |W(X)| \times |W(Y)| \tag{13}$$

where θ is the angle between vector $W(X)$ and $W(Y)$, and $|W(X)|$ and $|W(Y)|$ are the lengths of vector $W(X)$ and $W(Y)$ respectively[4].

$\cos(\theta)$ in (13) can be used as a correlation point function. The range of $\cos(\theta)$ is $[-1, 1]$ and it provides a measure of the similarity of two vectors. For any vector $W_{j,s}(X)$, $\cos(\theta)$ will be 1 for any vector $W_{j+k,s}(Y) = a \times W_{j,s}(X) + b$.

To make the correlation point function more likely to exhibit unique perfect correlation, we can define it as follows:

$$\begin{aligned}
\mathrm{CPF}(X, Y, j, k, s)_{NDP1} &= \frac{\min(|W(X)|, |W(Y)|}{\max(|W(X)|, |W(Y)|} \times \cos(\theta) \\
&= \frac{\min(|W(X)|, |W(Y)|)}{\max(|W(X)|, |W(Y)|} \times \frac{\sum_{i=j}^{j+s-1} x_i \times y_{i+k}}{|W(X)| \times |W(Y)|} \\
&= \frac{\sum_{i=j}^{j+s-1} x_i \times y_{i+k}}{[\max(|W(X)|, |W(Y)|)]^2} \\
&= \frac{\sum_{i=j}^{j+s-1} x_i \times y_{i+k}}{\max\left(\sum_{i=j}^{j+s-1} x_i^2, \sum_{i=j}^{j+s-1} y_{i+k}^2\right)}
\end{aligned} \tag{14}$$

As x_i and y_i are non negative, the range of $\mathrm{CPF}(X, Y, j, k, s)_{NDP1}$ is [0, 1]. It can be shown that $\mathrm{CPF}(X, Y, j, k, s)_{NDP1}$ will be 1 only when $W_{j,s}(X) = W_{j+k,s}(Y)$. Therefore, $\mathrm{CPF}(X, Y, j, k, s)_{NDP1}$ is likely to exhibit unique perfect correlation.

[4] We have dropped the subscripts of W(X) and W(Y) for clarity purposes in this section.

Normalized Dot Product 2 (NDP2) Another way to normalize the dot-product of two vectors is

$$\mathrm{CPF}(X,Y,j,k,s)_{NDP2} = \frac{\sum_{i=j}^{j+s-1} x_i \times y_{i+k}}{\sum_{i=j}^{j+s-1} [\max(x_i, y_{i+k})]^2} \tag{15}$$

Because x_i and y_i are non negative, the range of $\mathrm{CPF}(X,Y,j,k,s)_{NDP2}$ is $[0,1]$. It is obvious that $\mathrm{CPF}(X,Y,j,k,s)_{NDP2}$ equals 1 only when $W_{j,s}(X) = W_{j+k,s}(Y)$.

Among these four proposed correlation point functions, Mini/MaxSum Ratio (MMS) is likely to be the most sensitive to local details of the IPD vectors to be correlated. This is because it does not average any differences, and it accumulates all the IPD differences. As a result, MMS may potentially have a lower true positive rate due to its emphasis on local details. While the STAT CPF is much more robust to noise, we expect it to have substantially more false positives. The normalized dot product functions (NDP1 and NDP2) are likely to be in between MMS and STAT in terms of sensitivity to local detail and robustness to noise.

4.3 Correlation Value Function

Given flows X, Y, correlation window size s and threshold δ, by applying formula (8), we can potentially obtain a set of correlation points: $(j_1, j_1 + k_1), (j_2, j_2 + k_2), \ldots, (j_n, j_n + k_n)$. We represent this sequence of correlation points through two n-dimensional vectors $C_x = \langle j_1, \ldots j_n \rangle$ and $C_y = \langle j_1 + k_1, \ldots, j_n + k_n \rangle$.

We define the overall *Correlation Value Function* CVF of flows X and Y from this sequence of correlation points, as follows:

$$\mathrm{CVF}(C_x, C_y) = \begin{cases} 0 & n = 0 \\ \rho(C_x, C_y) & n = 1 \\ 1 & n = 1 \end{cases} \tag{16}$$

where

$$\rho(C_x, C_y) = \frac{\sum_{i=1}^{n} (j_i - E(C_x)) \times (j_i + k_i - E(C_y))}{\sqrt{[\sum_{i=1}^{n} (j_i - E(C_x))^2] \times [\sum_{i=1}^{n} (j_i + k_i - E(C_y))^2]}}$$

$\mathrm{CVF}(C_x, C_y)$ quantitatively expresses the overall correlation between flows X and Y, and its value range is $[-1, 1]$. When there exists more than one correlation point and all the correlation points have same correlation offset (i.e., $k_1 = k_2 = \ldots k_n$), $\mathrm{CVF}(C_x, C_y) = 1$. When flow X and Y has only one correlation point, $\mathrm{CVF}(C_x, C_y) = 1$. When flow X and Y have no correlation point, $\mathrm{CVF}(C_x, C_y)$ is defined to be 0.

4.4 Limitations and Countermeasures

The effectiveness of IPD-based correlation relies on the uniqueness of IPDs of connections. It may be ineffective at differentiating uncorrelated connections that exhibit very similar IPD patterns, such as file transfers accomplished via FTP. Interactive traffic, as we show later in the experiments section, usually has IPD patterns that are distinctive enough for our purposes.

Intruders could thwart IPD-based correlation by deliberately changing the inter-packet delays of a connection in a chain. Such delays may be designed to reduce the true positive rate, or increase the number of false positives. There are relatively simple means of accomplishing such traffic shaping, although they may require kernel-level manipulations. The amount of delay that can be added by the intruder is limited by the maximum delay that is tolerable for interactive traffic. Another countermeasure against IPD-based correlation is to set some connection into line mode while keeping other connection in character mode. This could potentially merge several packets into one bigger packet. However, the server side shell could always dynamically turn off the line mode [10]. Other countermeasures include:

- Injecting "padding" packets that can be removed by the application
- Segmenting one flow into multiple flows and reassembling them, again at the application level

It is an area of future work to address such countermeasures.

5 Experiments

The goal of the experiments is to answer the following questions about IPD based correlation:

1. Are inter-packet delays preserved through routers and stepping stone, and to what extent?
2. Are inter-packet delays preserved across encryption/decryption and various network applications (such as telnet/rlogin/SSH)?
3. How effective is the IPD-based correlation metric in detecting connections which are part of the same chain?
4. How well does the IPD-based correlation metric differentiate a connection from connections that are not part of the same chain?

5.1 Correlation Point Experiment

To answer the first two questions, we have conducted the following experiment. We first telnet'ed from a Windows 2000 PC behind a cable modem connected to an ISP in North Carolina to a Sun workstation via VPN. From the workstation, we used SSH to login to another workstation at N. C. State University. We then telnet'ed to a PC running Linux at UC Davis, and from there we SSH'ed back to a PC running FreeBSD at NC State. As shown in Figure 2, the connection chain has a total of 3 stepping-stones and 59 hops and covers a distance on the order of 10,000 *km*. The connection chain consists of links of different speeds – including residential Internet access, typical campus LAN and public Internet

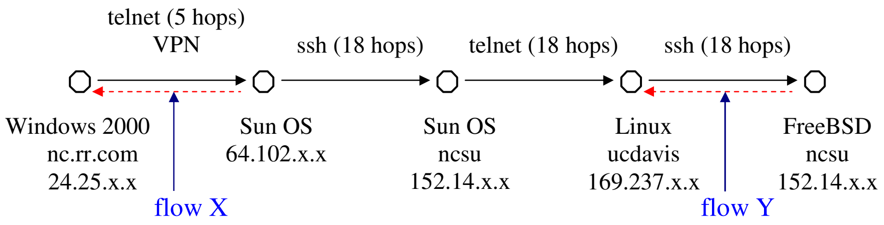


Fig. 2. Correlation Experiment on telnet and ssh

backbone. We have captured the packet traces at the Windows 2000 node and the FreeBSD node; both traces have a timestamp resolution of 1 microsecond. We label the telnet return path[5] flow from the Sun workstation to the Windows 2000 PC flow X, and the SSH backwards flow from the FreeBSD PC to the Linux PC flow Y. Therefore, flow X consists of telnet packets and flow Y consists of SSH packets.

We have calculated the IPD vectors after filtering out the following sources of errors:

- Duplicated packets
- Retransmitted packets
- ACK only packets

We then calculated correlation points (j, k) by applying (8) using each of the four correlation point functions, with different correlation window sizes s and correlation point thresholds δ_{cp}.

Figure 3 shows the correlation points between flow X and Y obtained by the MMS CPF with different correlation window sizes s and thresholds δ_{cp}. In these plots, a point at position (j, k) indicates inequality (8) was true for that value of j, k, s and δ_{cp}. True positives are points located along the major diagonal. False positives are points located off the major diagonal.

With correlation window size of 5 and δ_{cp} threshold of 0.70, there are a large number of falsely detected correlation points (false positives) in addition to the true correlation points (true positives). The overall CVF value (equation (4.13)) for this case is 0.1707. With larger correlation window size, or a higher threshold δ_{cp}, MMS results in fewer false positives and has a higher CVF value, as would be expected. At correlation window size 15, and a threshold δ_{cp} of 0.70, MMS detects most of the true correlation points between flow X and Y, finds no false positives, and has an overall CVF value of 0.9999. When the threshold δ_{cp} is increased to 0.95 with the same correlation window size of 15, MMS detects substantially fewer correlation points between flow X and Y, with no false positives, and has an overall CVF value of 1.0. This suggests that with correlation window

[5] The "return path" is the echoed traffic generated on the destination host and sent to the origination host.

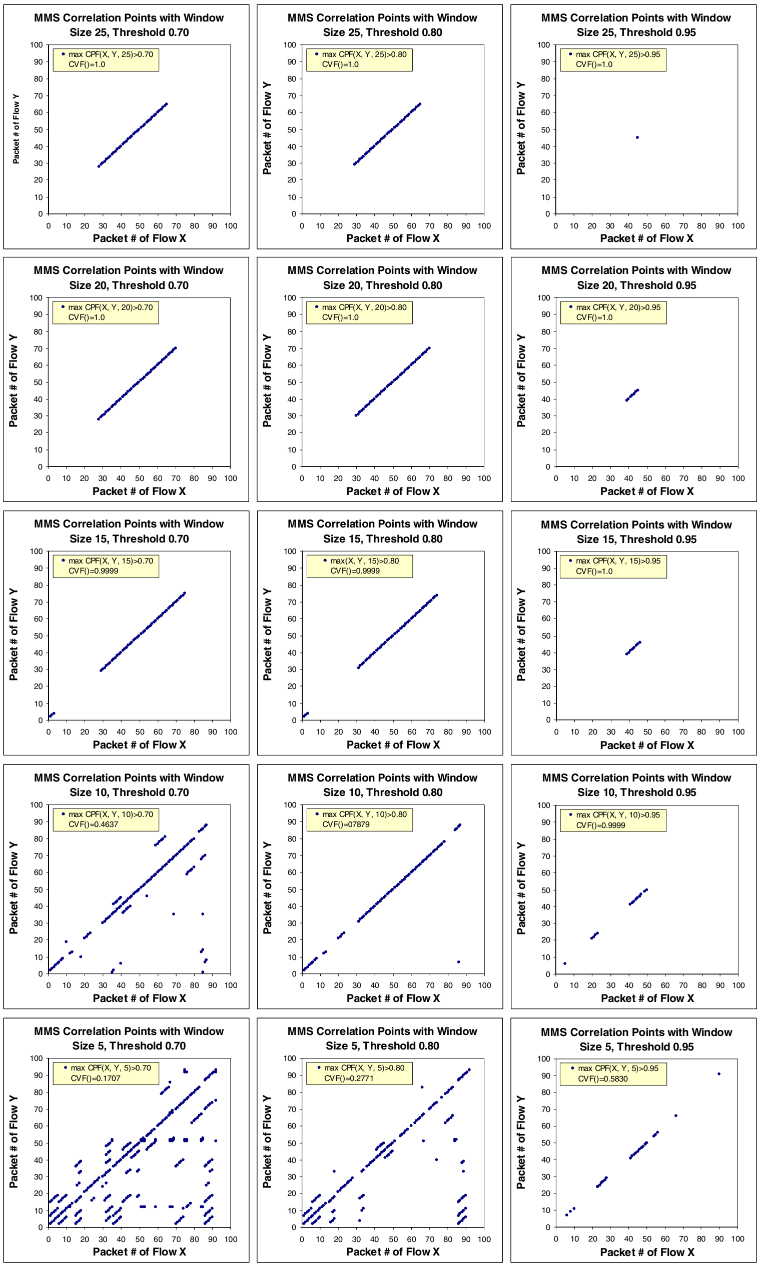


Fig. 3. Correlation Point between Two Correlated Flows Detected by MMS with Different Correlation Window Sizes and Thresholds

Table 2. Traces of Flows Used in Correlation Experiments

Flow Set	Date	Flow Type	Flow #	Packet #
FS1	03/02/02	SSH	16	12372
FS2	03/02/02	Telnet	15	5111
FS3	02/20/01	Telnet/SSH	144	34344
FS4	02/26/01	Telnet/SSH	135	38196
FS5	05/xx/02	SSH	400	364158

size 15, the threshold δ_{cp} of 0.95 is probably too high for MMS. The correlation points missed by correlation windows size 15 and threshold δ_{cp} of 0.95 are actually due to correlation-offset shifts. The correlation-offset between our sample flows X and Y has shifted 3 times between the start and finish. This indicates that a telnet packet may trigger more than one SSH packet, or vice versa. Fortunately, such correlation-offset shifts are infrequent between correlated flows. Generally, a larger correlation window size is very effective in filtering out false positives, and a higher threshold δ_{cp} tends to filter out both true positive and false positive correlation points. An excessively large correlation window size with a high threshold δ_{cp} tends to have a low true positive rate, due to both correlation-offset shifts and IPD variations introduced by the network.

Figure 4 compares the detected correlation points between flow X and Y by different CPFs: MMS, STAT, NDP1 and NDP2, with identical correlation window sizes of 10 and threshold δ_{cp} of 0.80. As expected, the statistical CPF results in substantially more false positives than the other three CPFs. While NDP2 has slightly fewer false positives than NDP1, they both have somewhat more false positives than MMS. Generally, MMS is very sensitive to localized details of IPDs and is able to accurately correlate the flows using a smaller correlation window (i.e. 5). NDP1 and NDP2 are less effective with a small correlation window, but they are able to correlate effectively with a moderate window size (15 or 20). The statistical CPF appears to fail to consider enough localized details to correlate accurately

5.2 Aggregated Flow Correlation Experiment

To evaluate more generally the performance of the different correlation point functions, we have used five sets of flows (Table 2). FS1 and FS2 were collected at two ends of connection chains similar to the scenario shown in Figure 2. FS1 and FS2 contain 16 SSH flows and 15 Telnet flows, respectively; for each flow in FS2, there is one flow in FS1 which was in the same connection chain. FS3 and FS4 are derived from 5 million packet headers and 12 million packet headers of the Auckland-IV traces of NLANR [5]. FS5 is derived from over 49 million packet headers of the Bell Lab-I traces of NLANR.

We have conducted four sets of aggregated correlation experiments. For all of these experiments, two flows were regarded as being correlated if the CVF

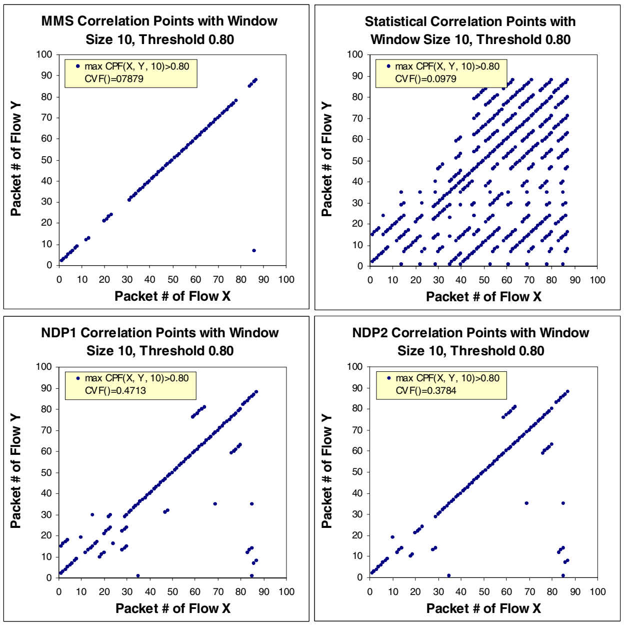


Fig. 4. Correlation Points Detected by MMS, Stat, NDP1 and NDP2 with Same Window Size and Threshold

of their correlation points (equation (16)) was greater than $\delta = 0.6$. The first set tests how well different correlation metrics (CPF) detect correlation between sets FS1 and FS2. Figure 5 shows both the number of true positives (out of a total of 15) and the number of false positives (out of $15\times15=225$) of flow correlation detection with different correlation window sizes and correlation point thresholds δ_{cp}.

With a δ_{cp} threshold of 0.70, MMS reaches its TP peak of 93% at a correlation window size of 20, and NDP2 reaches its TP peak of 80% with a correlation window size of 20 or 25. However NDP2 has a significantly higher number of false positives at the window size corresponding to its peak true positive rate than does than MMS. Both STAT and NDP1 have very low (<7%) TP rates with all correlation window size. This indicates that STAT and NDP1 are ineffective with a low δ_{cp} threshold.

For all δ_{cp} threshold values, MMS attains its peak TP rate with 0 false positives. NDP1 and NDP2 show a similar success rate, with a somewhat higher failure (false positive) rate. STAT is generally not successful at correlating the flows in the same chain. The best results are obtained for the highest δ_{cp} threshold setting. MMS is able to achieve 100% TP rate with 0 false positives with correlation window size 15, δ_{cp} threshold 0.90 and window size 10, δ_{cp} threshold 0.95. NDP2 is also able to have 100% TP rate with 0 FP at correlation window size 15, δ_{cp} threshold 0.95. NDP1's overall TP peak is 93% with 7% FP at correlation window size 20, δ_{cp} threshold 0.90.

The second set of experiments shows the correlation detection effectiveness by different correlation metrics. We use combined flow set of FS3 and FS4 (279 flows) and flow set FS5 (400 flows) to correlate themselves respectively. Figure 6 shows the true positive rate of different correlation metric with different correlation window size and δ_{cp} threshold. Again the STAT correlation point function consistently performs poorly. MMS and NDP1 almost have identical correlation detection rates across all the correlation window size and δ_{cp} threshold combinations in both data sets, where NDP1 has little lower detection rate. For flow set FS5, the detection rates of both MMS and NDP2 reach 92% and higher with correlation window size 25 or bigger. At correlation window size 35, MMS's and NDP2's detection rate achieve over 97%. For the combined flow set FS3 and FS4, at a correlation window size of 15, for δ_{cp} threshold 0.95, MMS, NDP1 and NDP2 all have the highest correlation detection rate of 76.7%. This lower detection rate is due to the nature of the flows in FS3 and FS4. We have found a number of SSH flows in FS3 and FS4 show very similar periodicity, with constant very short IPDs. We suspect they are bulk data transfers within the SSH connections. This shows a potential limitation of the use of IPD-based tracing.

The third experiment is intended to evaluate the ability of the different correlation point functions to successfully discriminate flows not part of the same chain. Figure 7 shows the number of false positives (out of $16 \times 279 = 4464$) when correlating FS1 and the combined flow set of FS3 and FS4. Because no flow from FS1 correlates with any flow from FS3 and FS4, any detected correlation by the correlation metric is a false positive. MMS consistently has 0 false positives; and NDP1 and NDP2 false positives decrease as the correlation window size increases. The STAT correlation point function reports an increasing number of FPs with larger correlation sizes.

The fourth experiment similarly investigated the false positive rate, this time between sets FS3 and FS4. Figure 8 shows the results. The number of false positives (out of $144 \times 133 = 19152$) for MMS, NDP1 and NDP2 decreases dramatically when the correlation window size increases; that of MMS decreases faster than NDP1 and NDP2. Again, the statistical correlation metric has a consistently higher FP rate with increasing correlation window size. For the MMS method, a window size of 20 0r 25 packets is sufficient to reduce the false positive rate to a fraction of a percent.

In summary, we have found that MMS is very effective in both detecting interactive, correlated flows and differentiating uncorrelated flows with even relatively small correlation window sizes (10, 15). NDP1 and NDP2 are not as sensitive as MMS with small correlation windows; however, they both perform well with larger correlation windows. We have confirmed that the statistical correlation metric is not effective in detecting correlation and differentiating uncorrelated flows.

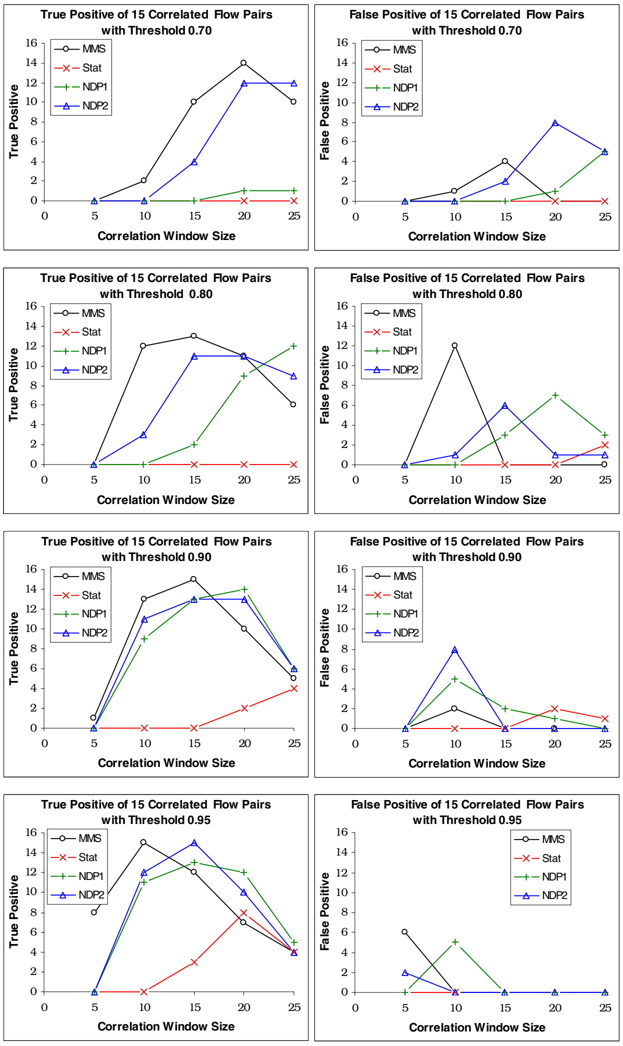


Fig. 5. True Positive and False Positive of Correlation between 16 and 15 Correlated Flows

5.3 Correlation Performance

We have measured the number of calculations of correlation points per second achieved by our unoptimized code. Table 3 shows the average number of millions of correlation point calculation per second of various correlation point functions under different load. Despite dynamic overheads of disk operation, the overall throughput remains largely constant at various loads.

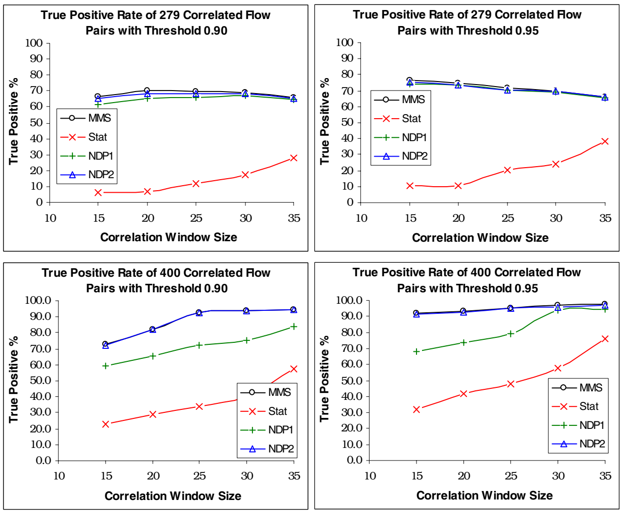


Fig. 6. True Positive Rate of Correlation between 279 and 279, 400 and 400 Correlated Flows

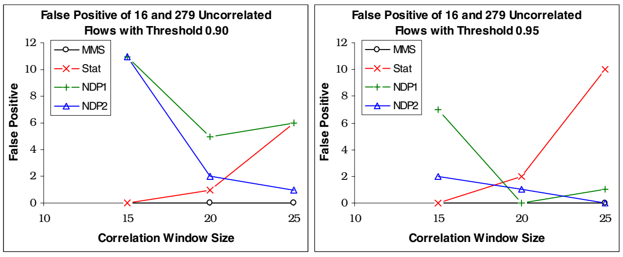


Fig. 7. False Positive of Correlation between 16 and 279 Uncorrelated Flows

Table 3. Throughput (Millions Per Second) of Correlation Point Calculation with Correlation Window Size 15

	40k	379k	937k	2309k	5704k	54210k
MMS	2.00	1.65	1.74	1.75	1.43	1.35
STAT	0.90	0.76	0.69	1.14	1.22	1.83
NDP1	3.99	3.16	2.23	3.25	2.29	3.24
NDP2	1.33	1.31	1.16	1.37	1.17	1.13

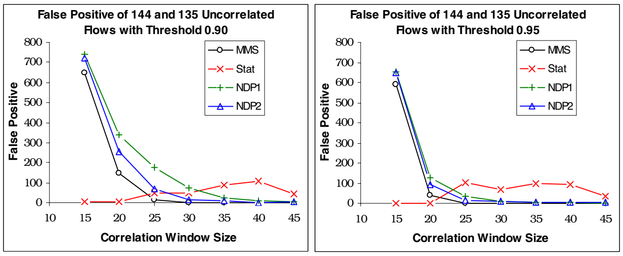


Fig. 8. False Positive of Correlation between 144 and 135 Uncorrelated Flows

6 Conclusions

Tracing intrusion connections through stepping-stones at real-time is a challenging problem, and encryption of some of the connections within the connection chain makes tracing even harder. We have addressed the tracing problem of encrypted connections based on the inter-packet delays of the connections. We proposed and investigated four correlation point functions. Our correlation metric does not require clock synchronization, and allows correlation of measurements taken at widely scattered points. Our method also requires only small packet sequences (on the order of a few dozen packets) for correlation. We have found that after some filtering, IPDs (Inter-Packet Delay) of both encrypted and unencrypted, interactive connections are largely preserved across many hops stepping-stones. We have demonstrated that both encrypted and unencrypted, interactive connections can be effectively correlated and differentiated based on IPD characteristics.

Our experiments also indicate that correlation detection is significantly dependent on the uniqueness of flows. We have found that normal interactive connections such as telnet, SSH and rlogin are almost always unique enough to be differentiated from connections not in the same chain. While bulk data transfer with SSH connection introduces an additional challenge in correlation detection, its impact on correlation differentiation may simply be offset by larger correlation windows and higher correlation point thresholds.

A natural area of future work is to extend the correlation to non-interactive traffic. How to address countermeasures with "bogus packets" and packet splitting and merging remains an open problem.

References

[1] M. H. DeGroot. *Probability and Statistics*. Addison-Wesley, 1989. 251
[2] H. Jung, et al. Caller Identification System in the Internet Environment. In *Proceedings of 4th USENIX Security Symposium*, 1993. 246
[3] S. Kent, R. Atkinson. Security Architecture for the Internet Protocol. *IETF RFC 2401*, September 1998. 245

[4] S. C. Lee and C. Shields. Tracing the Source of Network Attack: A Technical, Legal and Social Problem. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, June 2000. 245

[5] NLANR Trace Archive. `http://pma.nlanr.net/Traces/long/`. 257

[6] OpenSSH. `http://www.openssh.com`. 245

[7] D. Schnackenberg. Dynamic Cooperating Boundary Controllers. `http://www.darpa.mil/ito/Summaries97/E295_0.html`, Boeing Defense and Space Group, March 1998. 246

[8] S. Snapp, et all. DIDS (Distributed Intrusion Detection System) – Motivation, Architecture and Early Prototype. In *Proceedings of 14th National Computer Security Conference*, 1991. 245, 246

[9] S. Staniford-Chen, L. T. Heberlein. Holding Intruders Accountable on the Internet. In *Proceedings of IEEE Symposium on Security and Privacy*, 1995. 245, 246

[10] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocol*. Addison-Wesley, 1994. 254

[11] X. Y. Wang, D. S. Reeves, S. F. Wu and J. Yuill. Sleepy Watermark Tracing: An Active Network-Based Intrusion Response Framework. In *Proceedings of 16th International Conference on Information Security (IFIP/Sec'01)*, June, 2001. 245, 246

[12] K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *F. Guppens, Y. Deswarte, D. Gollmann and M. Waidner, editors, 6th European Symposium on Research in Computer Security - ESORICS 2000 LNCS-1895*, Toulouse, France, October 2000. 246, 247

[13] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of 9th USENIX Security Symposium*, 2000. 245, 246

Learning Fingerprints for a Database Intrusion Detection System

Sin Yeung Lee, Wai Lup Low, and Pei Yuen Wong

Computer Security Laboratory, DSO National Laboratories, Singapore
{lsinyeun,lwailup,wpeiyuen}@dso.org.sg

Abstract. There is a growing security concern on the increasing number of databases that are accessible through the Internet. Such databases may contain sensitive information like credit card numbers and personal medical histories. Many e-service providers are reported to be leaking customers' information through their websites. The hackers exploited poorly coded programs that interface with backend databases using SQL injection techniques. We developed an architectural framework, DIDAFIT (Detecting Intrusions in DAtabases through FIngerprinting Transactions) [1], that can efficiently detect illegitimate database accesses. The system works by matching SQL statements against a known set of legitimate database transaction fingerprints. In this paper, we explore the various issues that arise in the collation, representation and summarization of this potentially huge set of legitimate transaction fingerprints. We describe an algorithm that summarizes the raw transactional SQL queries into compact regular expressions. This representation can be used to match against incoming database transactions efficiently. A set of heuristics is used during the summarization process to ensure that the level of false negatives remains low. This algorithm also takes into consideration incomplete logs and heuristically identifies "*high risk*" transactions.

1 Introduction

Nowadays, most e-commerce sites offering some kind of online services have a database at its backend. Applications accessing these databases support a large variety of activities. Data found in these databases ranges from personal information and banking transactions to medical records and commercial secrets. Any breach of security to these databases can result in tarnished reputation for the organization, loss of customers' confidence and might even result in lawsuits. Unfortunately, recent reports indicate that there is a large increase in the number of security breaches, which resulted in theft of transaction information and financial fraud [2, 3, 4]. Clearly, it is important that the data in these databases be protected from unauthorized access and modification.

One mechanism to safeguard the information in these databases is to use intrusion detection systems (IDS). These systems aim to detect intrusions as early as possible, so that any damage caused by the intrusions is minimized.

They function as sentinels and ensure that any compromise on the integrity and confidentiality of the data is detected and reported as soon as possible. Intrusion detection research is not new and has been on going for many years. However, previous efforts were focused on *network-based intrusion detection* and *host-based intrusion detection*. Network-based intrusion detection typically works by monitoring network traffic and host-based intrusion detection works by monitoring log files in the hosts. Both network- and host-based intrusion detection systems look for attack signatures, which are specific patterns that usually indicate malicious or suspicious intent, to identify intrusions. There are numerous commercial network- and host-based intrusion detection systems in the market today, and the market leaders include RealSecure [5], NFR [6], Dragon [7], Cisco [8] and Symantec [9]. There are also IDS that are free and highly acclaimed (e.g. Snort [10]).

However, these intrusion detection systems do not work at the application layer, which can potentially offer more accurate and precise detection for the targeted application. The distinctive characteristics of database management systems (DBMSes), together with their widespread use and the invaluable data they hold, make it vital to detect any intrusion attempts made at the databases. Therefore, intrusion detection models and techniques specially designed for databases are becoming imperative needs. There are recent reports [4] in which SQL injection techniques, which refer to the use of carefully crafted and malicious SQL statements, were used by the intruders to pilfer sensitive information. SQL injection will be further discussed in a later section. This reinforces the point that intrusion detection systems should not only be employed at the network and hosts, but also at the database systems where the critical information assets lie.

DIDAFIT (Detecting Intrusions in DAtabases through FIngerprinting Transactions) is a system developed to perform database intrusion detection at the application level. It works by fingerprinting access patterns of legitimate database transactions, and using them to identify database intrusions. The framework for DIDAFIT has been described in [1]. This paper describes how the fingerprints for database transactions can be represented and presents an algorithm to learn and summarize SQL statements into fingerprints. The main contribution of this work is a technique to efficiently summarize SQL statements queries into compact and effective regular expression fingerprints. The technique can handle incomplete training data sets and uses heuristics to maintain false negatives (missed attacks) at low levels.

The rest of the paper is as follows. Section 2 discusses two basic concepts necessary for the understanding of subsequent sections. A brief introduction to the DIDAFIT framework is given in Section 3. The issues that arise during fingerprint learning and derivation are discussed in Section 4. In Section 5, our algorithm for fingerprint learning is detailed together with a full example. We survey the related works in Section 6, and conclude in Section 7.

2 Preliminaries

In this section, we introduce *SQL injection* and *SQL fingerprints* which are necessary for understanding the rest of the paper.

2.1 SQL Injection

Application users do not usually communicate with the database server directly, but through the application server. Although there is no direct interaction with the database server, it is possible that unauthorized users can access the database in ways unintended by the developer. This is made possible by carelessly designed applications, database server holes, as well as application server exploits. One technique of exploiting carelessly written database applications is *SQL injection* [11, 12, 13]. SQL injection refers to crafting SQL statements using "string building" techniques to trick the application server into executing the intruder's (often malicious) code. Possible results of actions by the injected code include information disclosure, unauthorised data modification, deletion of database or even escalation of the intruder's database privileges to that of the administrator's.

As an illustration, consider the following Perl script:

```
...
my $passwd = $cgi->param('passwd');
my $name= $cgi->param('name');
$sql = "select * from cust where name='$name'".
       " and passwd='$passwd'";
$sth = $dbh->prepare($sql);
$sth->execute;
if (!($sth->fetch)) {
   report_illegal_user();
} ...
```

The script shown is a typical procedure for login checking. It is supposed to verify if the user has supplied the user name and his/her password correctly. However, this script is vulnerable to SQL injection attacks. A malicious user can enter the following text into the password field of the submitted form:

```
x' OR 'x'='x
```

Assuming the user name is "alice" and the password is as above, the prepared SQL statement becomes

```
select * from customer
where name='alice' and passwd='x'
      OR 'x'='x'
```

The `where` clause of this statement will always be true since the intruder has carefully injected a " `OR 'x'='x'` " clause into it. This makes the result set of the query non-empty no matter what password is supplied. The malicious user can now log in as the user "alice" without knowing the password. The main reason for this vulnerability is the carelessly written procedure. DIDAFIT can detect such attacks on the database and other anomalous data accesses. Conventional solutions to this vulnerability include the use of stored procedures and checking user parameters before using them (e.g. dangerous characters such as ' should not be allowed in the input). While stored procedures tend to be non-portable and require longer development time, methods to enforce good programming practices are beyond the scope of this work.

2.2 SQL Fingerprints

For most applications with database services, the SQL statements submitted to the database are typically generated by some server-side scripts/programs. These statements are generated in a predictable manner and this regularity gives rise to opportunities to characterize valid transactions with some sort of signatures.

For example, assume we have a `delete order` transaction that deletes from the `order` table. Further assume that an order can only be deleted by specifying its `orderID`. A typical **valid** SQL statement can be

```
delete from order where orderID='12573';
```

Now, suppose the database server receives the following SQL statement:

```
delete from order where custid='eric';
```

This SQL statement does not conform to the pattern of the SQL statements for `delete order` transaction (i.e. it uses `custid` as the criteria for filtering instead of `orderID`). This statement could be the result of an intruder.

We have presented the idea behind fingerprinting database transactions. DIDAFIT uses a fingerprinting technique that derives signatures (which we call fingerprints) to help identify illegitimate SQL statements. In DIDAFIT, each fingerprint characterizes one set of SQL statements. Our fingerprints (as described in [1]) uses regular expressions. A single fingerprint can precisely represent various variants of SQL statements that can result from a transaction.

For example, we have a transaction that can query the order table given a customer ID (`custid`) and a range of values for the order amount (`amt`). The code below shows how this service may be coded in Perl.

```
$sqlstm = "SELECT orderID, amt from order where ";
$sqlstm .= "custid='$custid' and ";
$sqlstm .= "amt>$min_amt and amt<$max_amt";
```

We list below some possible SQL statements generated from this code:

```
SELECT orderID, amt from order where custid='3822' and amt>20 and amt<100
SELECT orderID, amt from order where custid='7312' and amt>10 and amt<200
```

All variants of queries generated by the code above can be represented using this regular expression:

```
^SELECT orderID, amt from order where custid='[^']*'
        and amt>[[:digit:]]+ and amt<[[:digit:]]+$
```

Notably,

1. The literal containing the value of `custid` has been replaced with the regular expression `'[^']*'`, which represents a quoted string.
2. The numerical value of `amt` is represented by the regular expression `[[:digit:]]+`, which represents an integer.
3. Finally, the whole expression begins with a `^` and ends with a `$` to prevent any additional clauses from being injected into the statement (such as using the `union` clause to append another query).

3 DIDAFIT: An Overview

DIDAFIT is designed as a misuse detection system. It can be broadly classified as a signature-based IDS with enhanced capabilities for learning and deducing new signatures. The overall architecture for DIDAFIT is shown in Figure 1. The meaning of the numbered flows in Figure 1 are as follows:

1. The application user issues a service request to the application server. This transaction may or may not be legitimate.
2. The application server formulates the necessary SQL statements and issues them to the database server through the database user.
3. The database user logs into the database. The database session is traced and the SQL statements received from the application are channeled to the misuse detection module.
4. In the misuse detection module, the received SQL statements are matched with the set of fingerprints of legitimate database transactions.
5. Anomalies or intrusions are then channeled to the reaction modules for the appropriate action to be taken. Actions that can be taken include alerting the administrators, sounding the alarm on the console and paging the duty personnel.
6. Output is returned to the application user (if applicable).

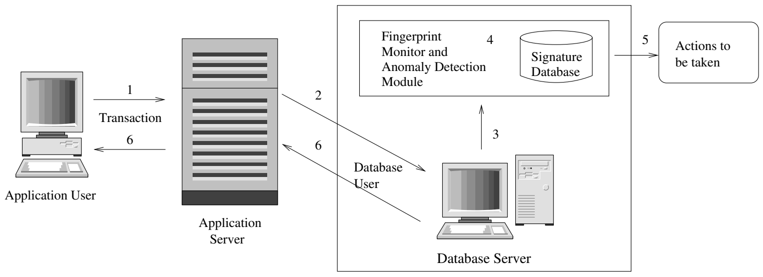


Fig. 1. Architecture for DIDAFIT

DIDAFIT involves three components:

1. An utility to log all the SQL statements submitted to the database server.
   We need to capture the submitted SQL statements for the database transactions in order to compare them with those in the "legitimate" fingerprint database. Different database systems offer different ways to capture the SQL statements. For example, Oracle provides the `sql_trace` [14] utility that can be used to trace all database operations in a database session of an user. The trace results are logged to a file, but we can channel the data to a monitoring and misuse detection module. Note that the `sql_trace` utility is not designed for this purpose. Rather, its intended use was to aid in the process of optimizing database performance. However, we make use of its capability to log SQL statements executed by the database engine. A concern with using the trace facility of database systems is its impact on the performance of the databases. Our experiments in [1] show that the impact on performance is extremely small.
2. A process to derive the fingerprints for SQL statements of legitimate database transactions.
   We propose the use of regular expressions to represent the derived fingerprints as described in Section 2.2.
3. A database of "legitimate" fingerprints to be used for database intrusion detection.
   To make use of the fingerprints, every submitted SQL statement is matched with the set of legitimate fingerprints. If the SQL statement cannot match any of the fingerprints, then an intrusion may have occurred.
   As an illustration, consider the example of `order` table queries (Section 2.2). Suppose our database server receives the following SQL statement.
   ```
   select orderID, amt from order;
   ```
   This statement does not match our legitimate fingerprint shown previously. This is caused by the missing `custid` input, which is mandatory by our signature. Hence, this anomalous statement is detected.
   There are many software packages and utilities that can process regular expressions. The standard Unix tool `egrep` is one such software that can handle the matching of our fingerprints efficiently.

4 Issues in Automated Fingerprints Learning and Derivation

One obvious method of generating the complete set of fingerprints for all database transactions is to do a code-walkthrough. However, this may not be feasible for several reasons. The code may contain sensitive business logic and not available for the walkthrough for some applications. The code base for many applications are also large and changes frequently. Maintaining the consistency and keeping the set of finerprints updated will require a lot of effort. Hence, it is important that the fingerprints be automatically learnt and deduced as much as possible.

In this section, we first discuss the problems and challenges that arise when we automate the learning and derivation of the fingerprints for the database transactions. We propose possible solutions after that.

4.1 Problems and Challenges

An obvious method to "learn" fingerprints is to use each legitimate SQL statement in the training log as a fingerprint. However, it suffers from the following problems:

1. *The set of SQL statements collected is a very large set.* If each unprocessed SQL statement forms a fingerprint, the size of the fingerprint database becomes unmanageable. We need a way to automatically summarize the SQL statements. Consider the following SQL statements:
   ```
   delete from order where orderID='13245';
   delete from order where orderID='45718';
   ```
   Although the `orderID`s are different, they serve the same function, which is to remove a particular order based on the `orderID` and hence, can be summarized to a single fingerprint:
   ```
   delete from order where orderID=#$ORDERID;
   ```
   where `#$ORDERID` is a meta-constant to denote a string literal in the domain of `orderID`. However, we need to avoid over-summarization. For example,
   ```
   delete from order where orderType='UNDELIVERABLE';
   delete from order where orderType='NORMAL';
   ```
   should not be summarized to a single fingerprint. The first SQL statement may be a typical statement used during maintenance. However, it is possible that the second statement originated from a malicious attack (why should anyone delete all "normal" orders?). Over-summarization may cause DIDAFIT to miss such illegitimate statements. Thus, our learning algorithm must be able to group similar SQL statements into fingerprints, but yet does not increase the probability of missing attacks.
2. *The SQL transactions in the training trace log may contain illegitimate statements from past intrusion activities.* We do not assume that all the SQL statements in the training log to be legitimate. Our learning algorithm must be able to detect potentially invalid SQL statements, even within the training set of SQL statements.
3. *The trace log may not be complete.* Some legitimate transactions may not be executed during the period of monitoring. With incomplete input for training, DIDAFIT may treat unseen, but legitimate statements as illegitimate. This may give rise to a large number of false positives. Thus, our learning algorithm should be able to deduce a set of possibly legitimate fingerprints with an associated level of confidence, that are missing from the training data. The legitimacy of these deduced fingerprints can be ascertained by the DBA.

```
L1   select orderID, Amt from order where custID='2317';
L2   select orderID, Amt from order where custID='1920';
L3   select orderID from order where custID='3571' and amt>100 and amt<300;
L4   select prod_desc from product where prodID='X3175';
L5   select orderID from order where custID='' or TRUE or custID='';
L6   select comment from prod_comment where prodID='X3175' and conf='PUBLIC';
L7   select orderID, Amt from order where custID='8256';
L8   select orderID from order where custID='1354' and odate='1-Jan-1999' and amt>500;
L9   select comment from prod_comment where prodID='X0507' and conf='PRIVATE';
L10  select prod_desc from product where prodID='X1754';
L11  select orderID from order where custID='1028' and odate='1-Jan-1999' and amt<1000;
L12  select orderID, Amt from order where custID='1754';
L13  select comment from prod_comment where prodID='X1754' and conf='PUBLIC';
L14  select comment from prod_comment where prodID='X0507' and conf='PUBLIC';
L15  select comment from prod_comment where prodID='X3075' and conf='PUBLIC';
L16  select prod_desc from product where prodID='X0675';
L17  select orderID, Amt from order where custID='8317';
```

Fig. 2. A snapshot of the SQL trace log for an E-mall

Before we present our solutions to these three challenges, we consider an online E-mall application with the following business rules:

– Users can create and delete their own orders.
– Users can perform searches on their own orders and limit the search by specifying the order ID (`orderID`), product ID (`prodID`), the range of the order's value (`amt`), and/or the date of order (`odate`).
– Users can see the description of the products (`prod_desc`).
– Users can only see the public comments on the product (`conf='PUBLIC'`), but not internal comments, which are reserved only for the staff of the E-mall.

A snapshot of the SQL trace log is shown in Figure 2. This example will be used in the examples for the rest of this work.

4.2 Selective Summarization of Literals

Observe from the logs in Figure 2 that L1 differs from L2 only by the value of the `custID` parameter. They query the same attributes for a given customer ID. Our learning algorithm can group these SQL statements under the same fingerprint. This can be done by first replacing each literal by a meta-constant. In this case, each string literal that represents the value of a `custID` attribute is replaced by the token `#$CUSTID`. The summarized fingerprint for L1 and L2 is

```
F1   select orderID, Amt from order where custID=#$CUSTID;
```

Likewise, the next three log records (L3,L4,L5) give the following three fingerprints:

```
F2   select orderID from order where custID=#$CUSTID and amt>#%AMT and amt<#%AMT;
F3   select prod_desc from product where prodID=#$PRODID;
F4   select orderID from order where custID=#$CUSTID or TRUE or custID=#$CUSTID;
```

However, as mentioned previously, replacing all literals to wild-card tokens indiscriminately may cause some malicious SQL statements to go undetected. Consider L6, L9, L13, L14 and L15 , the first attribute in the WHERE-clause (product ID `prodID`), can take any value, but the second attribute, (confidentiality `conf`), takes only the value “`PUBLIC`”. This corresponds well to the “business rule” that customers can see only the public comments but not otherwise. Thus, we should only replace the literals of the `prodID` with the wild-card token and let the literals of the `conf` attribute remain unchanged, i.e.

```
select comment from prod_comment
  where prodID=#$PRODID and conf='PUBLIC';
```

The difference between the two treatments lies on the fact that `prodID` is not from a small set of pre-specified values. This value carries no other implication, except to identify a product. On the other hand, the valid values of `conf` is restricted to a **small** list of pre-determined values. Values such as “PUBLIC”, “PRIVATE” and “SECRET” are not only used to identify tuples in the database, but also carry implications for the operations and sensitivity of the tuples. Hence, it is not advisable to summarize these literals into one token.

In view of this, our algorithm will replace a literal with a token only when the literal corresponds to a domain that carries no implicit meaning for operations and data sensitivity. Such domains can be determined by consulting the DBA. However, in the event that this information is unavailable, we can assume that if the range of values in the domain is very large (or unbounded), then it is unlikely that such value has an implicit meaning. To test if the domain is unbounded, we test the growth rate of the number of unique values for the attribute as the size of the sample increases. If the domain is unbounded, the growth rate will be constant. That is to say:

$$\text{Number of distinct literals found} \propto s$$

where s is the number of the samples in the observation. On the other hand, if the domain contains only a few fixed constants, then the number of distinct values will be similar to the standard diminishing growth curve,

$$\text{Number of distinct literals} = N_0(1 - e^{-\lambda s})$$

This behavior can be easily tested with many statistical methods such as the non-parametric one-sample Komoglov-Smirnov’s D Statistics test (KS-test). For example, consider the set of trace L6, L9, L13, L14, L15 from Figure 2. The `prodID` attribute employs four literals: `'X3175'`, `'X0507'`, `'X1754'` and `'X3075'` in the first conjunction in the WHERE-clause.

Using the one-sample KS-test, the D-statistic for testing the linear growth can be computed as follows:

$$D = \sqrt{5}\max\{|\frac{1}{4} - \frac{1}{5}|, |\frac{2}{4} - \frac{2}{5}|, |\frac{3}{4} - \frac{3}{5}|, |\frac{3}{4} - \frac{4}{5}|\} = 0.15$$

Table 1. Distribution of the `prodID` Literal

Number of statements sampled	1 (L6)	2 (L6,L9)	3 (L6,L9,L13)	4 (L6,L9,L13,L14)	5 (L6,L9,L13,L14,L15)
Total no. of unique literals observed	1	2	3	3	4
Cumulative distribution	1/4	2/4	3/4	3/4	4/4
Expected uniform distribution	1/5	2/5	3/5	4/5	5/5

At α=90% significance level, this value should be at most 0.563. Thus, we accept the hypothesis that the domain of `prodID` is unbounded.

On the other hand, the `conf` attribute has 2 unique literals in the 5-statement sample: `'PUBLIC'` and `'PRIVATE'`. The D-statistic is computed likewise as:

$$D = \sqrt{5}\max\{|\frac{1}{2}-\frac{1}{5}|,|\frac{2}{2}-\frac{2}{5}|,|\frac{2}{2}-\frac{3}{5}|,|\frac{2}{2}-\frac{4}{5}|\} = 0.6$$

It exceeds 0.563. Thus, we reject the hypothesis that the domain of `conf` is unbounded. Hence, we should retain all the constants related with `conf` when generating the fingerprints.

4.3 Detection of High-Risk Transactions

It is obvious that L5 is an instance of SQL injection. It is caused by injecting the string "`' or TRUE or custID='`" into the `custID` parameter. We consider this fingerprint to be *rare*. A fingerprint is *rare* if its frequency is statistically below a small threshold. The occurrence of a rare fingerprint warns of a possible malicious SQL statement.

In our algorithm, we tabulate the frequency of each fingerprint learnt. If a fingerprint occurs frequently, then it is safe to include it in the fingerprint database. For instance, consider the four fingerprints, F1, F2, F3 and F4 (in Section 4.2).

Fingerprint F1 matches 5 statements in the trace of size 17 (Figure 2). A z-test can conclude that the frequency is not close to 0. Similarly, fingerprint F3 can be considered safe based on its frequency. On the other hand, fingerprints F2 and F4 matches only one statement each in the trace. Statistically, at 95% confidence level, we cannot deny that their frequency is 0. In other words, they may correspond to potentially malicious statements and require further investigation.

To further refine our decision to ascertain the legitimacy of F2, we observe that F2 differs from F1 as follows:

1. F2 has a more restrictive WHERE-clause compared to F1. It specifies more conditions using the "AND" operator in the WHERE-clause than F1.
2. F2 selects fewer attributes than F1.

Thus, we can consider F2 to be safe because for any SQL statement that matches F2, the set of returned tuples is a subset of the tuples returned by some SQL statement matching F1 (which is a legitimate fingerprint). Thus, even if some SQL statements that match F2 is malicious, it cannot reveal more information than some legitimate query that is allowed by F1.

On the other hand, F4 cannot be considered safe. F4's WHERE-conditions are joined using the "OR" operator. Thus, it is possible that a query that matches F4 can reveal more tuples than what is allowed by the legitimate fingerprints.

Thus, our learning algorithm decides that an rare fingerprint $\mathcal{F}$ is legitimate if there exists another legitimate fingerprint $\mathcal{F}'$, such that $\mathcal{F}$ differs from $\mathcal{F}'$ only by

1. any extra conditions in the WHERE-clause of $\mathcal{F}$ that is missing from $\mathcal{F}'$ is joined with the "AND" operator; and
2. $\mathcal{F}$ selects an equal number of or fewer columns than $\mathcal{F}'$.

4.4 Deduction of Missing Fingerprints.

As mentioned previously, some legitimate fingerprints might not appear in the trace log due to incomplete training data. This incompleteness can cause many false alarms during the actual operation. To reduce this problem, we propose an algorithm to deduce a set of possible missing fingerprints. This set of missing fingerprints will then be presented to the DBA for confirmation to be included in the legitimate fingerprint set.

Consider the transaction set L3, L8, L11 of the log in Figure 2. These logs can be captured by the following three fingerprints:

```
P1   select orderID from order where custID=#$CUSTID and amt>#%AMT and amt<#%AMT;
P2   select orderID from order where custID=#$CUSTID and odate=#$ODATE and amt>#%AMT;
P3   select orderID from order where custID=#$CUSTID and odate=#$ODATE and amt<#%AMT;
```

The three fingerprints share the following properties:

1. Except for the conditions at the WHERE-clause, they are identical.
2. The WHERE-clause of each pattern uses three out of the following four conjuncts:
   "`custID=#$CUSTID`", "`odate=#$ODATE`", "`amt>#%AMT`" and "`amt<#%AMT`".

It is possible that there is another fingerprint, that uses all four conjuncts:

```
P4   select orderID from order where custID=#$CUSTID and odate=$#ODATE
        and amt>$%AMT and amt<$%AMT;
```

P4 compacts all the nine conjuncts used in P1, P2 and P3 into only four conjuncts. Indeed, this scenario can be generated by the following Perl script:

```
$sqlstm = "select orderID from order where custID='$custID'";
if !($odate eq "") {
  $sqlstm .= " and odate='$odate'";
}
if ($min_amt > 0) {
  $sqlstm .= " and amt>$min_amt";
}
if ($max_amt > 0) {
  $sqlstm .= " and amt<$max_amt";
}
```

In general, a fingerprint $\mathcal{F}$ is a *derived fingerprint* from fingerprints $\mathcal{F}_1, \ldots, \mathcal{F}_m$ with compactness η if

1. Each pair of fingerprints $\mathcal{F}_i, \mathcal{F}_j$ $(1 \leq i, j \leq m)$, as well as $\mathcal{F}, \mathcal{F}_i$ $(1 \leq i \leq m)$, differs only by some missing conjuncts in their WHERE-clauses.
2. The condition in the WHERE-clause of $\mathcal{F}$ subsumes every condition in the WHERE-clause of each $\mathcal{F}_j$ $(1 \leq j \leq m)$.
3. Each conjunct in the WHERE-clause of $\mathcal{F}$ appears in at least the WHERE-clause of at least one $\mathcal{F}_j$ $(1 \leq j \leq m)$.
4. Furthermore, η $(\eta \geq 1)$ is the ratio of the total number of possibly identical conjuncts in $\mathcal{F}_1, \ldots, \mathcal{F}_m$ over the number of conjuncts in $\mathcal{F}$.

In our derived fingerprint P4, the number of conjuncts used in the WHERE-clause is 4. The total number of conjuncts used in the three fingerprints (P1,P2,P3) is 9. Hence, P4's compactness is $\frac{9}{4}$. High compactness gives higher confidence that the fingerprints $\mathcal{F}_1, \ldots, \mathcal{F}_m$ are closely related. For example, consider the following case,

```
P1'   select orderID from order where custID=#$CUSTID;
P2'   select orderID from order where odate=#$ODATE;
P3'   select orderID from order where itemid=#$ITEMID;
```

These three fingerprints can derive the following:

```
P4'   select orderID from order where custID=#$CUSTID and odate=#$ODATE and itemid=#$ITEMID;
```

of compactness 1. The three fingerprints (P1',P2',P3') are likely generated by different functions. If this is the case, the derived fingerprint (P4') will never appear in the trace log.

To search for derived fingerprints with a high degree of compactness, we perform the following steps,

1. Group all the legitimate fingerprints into equivalent classes such that each pair of fingerprint F_i, F_j in the same class differs only by some missing conjuncts in their WHERE-clauses.
2. For each equivalent class, conjunct all the conditions in the WHERE-clauses of all fingerprints, removing duplicate conjuncts if necessary, into a condition C. Construct F_0 such that it is identical as F_1 except that the WHERE-clause is replaced with C.

Note that the legitimacy of this derived fingerprint should be confirmed with the DBA, before including it in the set of legitimate fingerprints.

5 Algorithm for Fingerprint Learning and Derivation

In this section, we summarize the discussion in the previous sections by presenting the algorithm to automatically generate the set of fingerprints. We illustrate the algorithm with a complete example.

The algorithm is shown in Algorithm 1. After that is a detailed walkthrough of the algorithm using the trace log shown in Figure 2.

Algorithm 1: Algorithm for Fingerprint Learning and Derivation

Given trace log LOG, and a set of attributes $\mathcal{A}$ that carries some implicit meaning optionally specified by the DBA, we perform the following steps:

1. For each attribute in the WHERE-clauses of LOG that is not found in $\mathcal{A}$, scan LOG to compute the Kolmogorov-Smirnov's D statistic to test for linear growth of the number of unique literals.
2. If the number of unique literals of any attribute does not statistically support a linear growth, put it into the set $\mathcal{A}'$.
3. Generate a set of unique fingerprints by replacing all the literals of attributes in the WHERE-clause that are neither in $\mathcal{A}$ nor $\mathcal{A}'$, with meta-constants.
   (a) replace all the string literals of the expression `attr='literal'` in the WHERE-clause with `attr=#$attr`.
   (b) replace all the numeric literals of the expression `attr=numeric_literal` in the WHERE-clause with `attr=#%attr`.
   Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be the fingerprints generated.
4. Count the number of occurrences for each fingerprint. Label the fingerprint "safe" if the number of occurrences is not statistically 0.
5. For each unclassified fingerprint $\mathcal{F}$, label it "safe" if there exists another "safe" fingerprint $\mathcal{F}'$, such that $\mathcal{F}$ differs from $\mathcal{F}'$ by only
   (a) any extra condition in the WHERE-clause of $\mathcal{F}$ that is missing from $\mathcal{F}'$ is joined with the "AND" operator; and
   (b) $\mathcal{F}$ selects an equal number of or fewer columns than $\mathcal{F}'$.
   The remaining fingerprints are labelled "unsafe".
6. Compute the set of derived fingerprints from the set of "safe" fingerprints. Mark these derived fingerprints as "derived".
7. Present all the "unsafe" and "derived" fingerprints to the DBA. Each "unsafe" or "derived" fingerprint that is approved by the DBA will be marked as "safe". If there remains any "unsafe" fingerprints, then an intrusion has occurred in LOG.
8. For each "safe" fingerprint, replace the meta-constant by the regular expressions. This legitimate fingerprint database can now be used to detect intrusions for the database application.

1. Assuming $\mathcal{A}$ is an empty set, we first scan the log to compute the Kolmogorov-Smirnov's D statistic for discrete data against linear growth

Table 2. Occurrence count of the fingerprints

Fingerprint	F1	F2	F3	F4	F5	F6	F7	F8
Frequency	5	1	3	1	4	1	1	1

of the number of unique values for each attribute in the WHERE-clause (i.e. `custid`, `amt`, `prodid`, `conf`, `odate`). For example, (as calculated in the previous section) the `prodid` has a D statistic of 0.15, while the `conf` has a D statistic of 0.6.

2. After step 2, we have $\mathcal{A}' = \{\texttt{conf}\}$.
3. The next step is to generate the initial fingerprint set by replacing the literals of each attribute not in $\mathcal{A}$ nor $\mathcal{A}'$ with its corresponding meta-constant. We obtain the following fingerprints:

```
F1   select orderID, Amt from order where custid=#$CUSTID;
F2   select orderID from order where custid=#$CUSTID and amt>#%AMT and amt<#%AMT;
F3   select prod_desc from product where prodid=#$PRODID;
F4   select orderID from order where custid=#$CUSTID or TRUE or custid=#$CUSTID;
F5   select comment from prod_comment where prodid=#$PRODID and conf='PUBLIC';
F6   select comment from prod_comment where prodid=#$PRODID and conf='PRIVATE';
F7   select orderID from order where custid=#$CUSTID and odate=#$ODATE and amt>#%AMT;
F8   select orderID from order where custid=#$CUSTID and odate=#$ODATE and amt<#%AMT;
```

4. We count the number of occurrences for each fingerprint next. The results are tabulated in Table 2.
   F1, F3 and F5 occurs frequently enough for us to infer that they are "safe". On the other hand, F2, F4, F6, F7 and F8, occurs only once each in the trace log. These fingerprints need to be looked into.
5. Upon inspection, F2, F7 and F8 can be classified as "safe" as they can be subsumed by F1. F4 and F6, however, are marked "unsafe". They should be highlighted to the DBA.
6. The next step involves finding derived fingerprints of high compactness. In this example, we set the degree of compactness to be at least 2. The following new fingerprint marked "derived" is found,

```
F9   select orderID from order where custid=#$CUSTID and odate=#$ODATE
        and amt>#%AMT and amt<#%AMT;
```

7. Assume the DBA approves the "derived" fingerprint F9, but does not approve all the other "unsafe" fingerprints. The legitimate fingerprints are F1, F2, F3, F5, F7, F8 and F9. The meta-constants are replaced by the appropriate regular expressions. The final fingerprints set of legitimate fingerprints for this database application consists of:

```
F1   select orderID, Amt from order where custid='[^']*';
F2   select orderID from order where custid='[^']*' and amt>[0-9]+ and amt<[0-9]+;
F3   select prod_desc from product where prodid='[^']*';
F5   select comment from prod_comment where prodid='[^']*' and conf='PUBLIC';
F7   select orderID from order where custid='[^']*' and odate='[^']*' and amt>[0-9]+;
F8   select orderID from order where custid='[^']*' and odate='[^']*' and amt<[0-9]+;
F9   select orderID from order where custid='[^']*' and odate='[^']*' and amt>[0-9]+
        and amt<[0-9]+;
```

6 Related Work

As far as the authors know, this is the only work using SQL transaction fingerprints or signatures to detect database intrusions. Closest to our work is [15] which profiles users and roles in a relational database system. It generates the profiles from the "working scopes" of the users which are defined as the sets of attributes that are usually referenced together with some values. This profile describes typical user behaviour and is used to detect misuse. This method assumes that the legitimate users show some level of consistency in using the database system. If this assumption does not hold, or if the threshold for inconsistency is not set properly, the result will be a high level of false positives. This method also faces the attribute selection problem when it chooses the "features" to consider when building the working scopes.

Classification algorithms (e.g. C4.5 [16]) seem to be applicable to our problem of identifying the instances of SQL statements that constitute an intrusion. However, this will lead to the "feature selection" problem in which we have to decide on the feature set of the SQL statements to be used in the classifier. The entire training set will also have to be tagged as either legitimate or illegitimate which may be infeasible for large training sets. Text summarization techniques [17, 18] cannot be applied to our problem of SQL transaction summarization as unlike text collections, our logs consist of largely independent SQL transactions and lack the dependency (style, theme) among linguistic units (phrase, clause, etc) in text collections.

7 Conclusion

DIDAFIT is a database intrusion detection system that identifies anomalous database accesses by matching database transactions with a set of legitimate transaction fingerprints. This work addresses the problem of learning the set of legitimate fingerprints from the database trace logs that contain the SQL statements. We developed an algorithm that can:

1. selectively and effectively summarize SQL statements into fingerprints.
2. detect "high-risk" (possibly malicious) SQL statements even in the training set of SQL statements.
3. derive possibly legitimate fingerprints that are missing from the SQL statements in the training set.

Currently, the algorithm works mostly at the syntactic-level. We are looking into how domain knowledge can contribute to form a semantically-richer intrusion detection mechanism. Further research is also done to study how the algorithm can be extended to support incremental learning.

Acknowledgment

The authors would like to thank the anonymous reviewers for their useful comments.

References

[1] Low, W. L., Lee, S. Y., Teoh, P.: DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions. In: Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS). (2002) 264, 265, 267, 269

[2] Atanasov, M.: The truth about internet fraud. In: Ziff Davis Smart Business, Available at URL http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2688776-11,00.html (2001) 264

[3] Hatcher, T.: Survey: Costs of computer security breaches soar. In: CNN.com, Available at URL http://www.cnn.com/2001/TECH/internet/03/12/ csi.fbi.hacking.report/ (2001) 264

[4] Poulsen, K.: Guesswork Plagues Web Hole Reporting. In: SecurityFocus, Available at URL http://online.securityfocus.com/news/346 (2002) 264, 265

[5] Internet Security Systems: RealSecure Intrusion Detection Solution, Available at URL http://www.iss.net (2001) 265

[6] NFR Security: NFR network intrusion detection, Available at URL http://www.nfr.com/products/NID/ (2001) 265

[7] Enterasys Networks, Inc.: The Dragon IDS, Available at URL http://www.enterasys.com/ids/dragonids.html (2001) 265

[8] Cisco Systems, Inc.: Cisco Intrusion Detection, Available at URL http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/ (2001) 265

[9] Symantec Corporation: Enterprise Solutions, Available at URL http://enterprisesecurity.symantec.com/ (2001) 265

[10] Roesch, M.: Snort: Lighweight intrusion detection for networks. In: Proceedings of the 13th Conference on Systems Administration (LISA-99), USENIX Association (1999) 229–238 265

[11] Andrews, C.: SQL injection FAQ, Available at URL http://www.sqlsecurity.com (2001) 266

[12] Anley, C.: Advanced SQL Injection In SQL Server Applications, Next Generation Security Software Ltd, Available at URL http://www.nextgenss.com/papers/advanced_sql_injection.pdf (2002) 266

[13] Anley, C.: (more) Advanced SQL Injection, Next Generation Security Software Ltd, Available at URL http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf (2002) 266

[14] Oracle: Oracle, 2001, Available at URL http://www.oracle.com (2001) 269

[15] Chung, C. Y., Gertz, M., Levitt, K.: Misuse detection in database systems through user profiling. In: Web Proceedings of the 2nd International Workshop on the Recent Advances in Intrusion Detection (RAID). (1999) 278

[16] Quinlan, J. R.: Induction of decision trees. In Shavlik, J. W., Dietterich, T. G., eds.: Readings in Machine Learning. Morgan Kaufmann (1990) Originally published in *Machine Learning* 1:81–106, 1986. 278

[17] Hovy, E., Lin, C. Y.: Automated Text Summarization in SUMMARIST. In: Proceedings of ACL/EACL Workshop on Intelligent Scalable Text Summarization. (1997) Madrid, Spain. 278

[18] Boguraev, B., Bellamy, R.: Dynamic Presentation of Phrasally-Based Document Abstractions. In: Proceedings of Thirty-second Annual Hawaii International Conference on System Sciences (HICSS). (1998) 278

Author Index