Pierpaolo Degano
Luca Viganò (Eds.)

LNCS 5511

Foundations and Applications of Security Analysis

Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security, ARSPA-WITS 2009
York, UK, March 2009, Revised Selected Papers

Springer

Pierpaolo Degano Luca Viganò (Eds.)

Foundations and Applications of Security Analysis

Joint Workshop on Automated Reasoning
for Security Protocol Analysis
and Issues in the Theory of Security, ARSPA-WITS 2009
York, UK, March 28-29, 2009
Revised Selected Papers

Springer

Volume Editors

Pierpaolo Degano
Università di Pisa, Dipartimento di Informatica
Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy
E-mail: degano@di.unipi.it

Luca Viganò
Università di Verona, Dipartimento di Informatica
Strada Le Grazie 15, 37134 Verona, Italy
E-mail: luca.vigano@univr.it

Library of Congress Control Number: 2009933043

CR Subject Classification (1998): D.4.6, K.6.5, C.2, H.2.7, K.4.4

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-642-03458-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-03458-9 Springer Berlin Heidelberg New York

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12725595 06/3180 5 4 3 2 1 0

Preface

The Joint Workshop on "Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security" (ARSPA-WITS 2009) was held in York, UK, March 28–29, 2009, in association with ETAPS 2009.

ARSPA is a series of workshops on "Automated Reasoning for Security Protocol Analysis," bringing together researchers and practitioners from both the security and the formal methods communities, from academia and industry, who are working on developing and applying automated reasoning techniques and tools for the formal specification and analysis of security protocols. The first two ARSPA workshops were held as satellite events of the Second International Joint Conference on Automated Reasoning (IJCAR 2004) and of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), respectively. ARSPA then joined forces with the workshop FCS (Foundations of Computer Security): FCS-ARSPA 2006 was affiliated with LICS 2006, in the context of FLoC 2006, and FCS-ARSPA 2007 was affiliated with LICS 2007 and ICALP 2007.

WITS is the official annual workshop organized by the IFIP WG 1.7 on "Theoretical Foundations of Security Analysis and Design," established to promote the investigation on the theoretical foundations of security, discovering and promoting new areas of application of theoretical techniques in computer security and supporting the systematic use of formal techniques in the development of security-related applications. This is the ninth meeting in the series. In 2008, ARSPA and WITS joined with the workshop on Foundations of Computer Security FCS for a joint workshop, FCS-ARSPA-WITS 2008, associated with LICS 2008 and CSF 21.

In 2009, ARSPA and WITS again joined forces with the aim to provide a forum for continued activity in different areas of computer security, bringing computer security researchers in closer contact with the ETAPS community and giving ETAPS attendees an opportunity to talk to experts in computer security, on the one hand, and to contribute to bridging the gap between logical methods and computer security foundations, on the other.

There were 27 submissions of high quality, from countries in Asia, Europe, and North America. All the submissions were evaluated by at least three referees and the Program Committee then selected the 15 research contributions that were presented at the workshop. Out of these, 12 were further revised by their authors and are included in this volume. The workshop program was enriched by invited talks by Peter Ryan and David Sands, whose contributions are also included here.

We would like to thank all the people who contributed to the organization of the ARSPA-WITS 2009 Workshop, and acknowledge the support from the IFIP WG 1.7, the AVANTSSAR Project (FP7-ICT-2007-1, Project No. 216471), and the SENSORIA Project (EU-FETPI Global Computing Project IST-2005-16004). In particular, we are deeply indebted to the other members of the Program Committee and the additional referees, who allowed us to review the papers in a very short time, while maintaining a very high standard: their help has been invaluable. We are also grateful to Andrei Voronkov, who allowed us to use the free conference software system EasyChair, which greatly simplified the work of the Program Committee. Last but not least, warm thanks to the organizers of ETAPS 2009.

June 2009

Pierpaolo Degano
Luca Viganò

Organization

Program Committee

Lujo Bauer	CMU, USA
Luca Compagna	SAP Research, France
Veronique Cortier	LORIA INRIA-Lorraine, France
Pierpaolo Degano (Chair)	Università di Pisa, Italy
Sandro Etalle	Technical University of Eindhoven, The Netherlands
Riccardo Focardi	Università di Venezia, Italy
Dieter Gollman	Technische Universität Hamburg-Harburg, Germany
Roberto Gorrieri	Università di Bologna, Italy
Joshua Guttman	MITRE, USA
Jerry den Hartog	Technical University of Eindhoven, The Netherlands
Jan Jürjens	The Open University, UK
Gavin Lowe	Oxford University, UK
Catherine Meadows	Naval Research Laboratory, USA
Jonathan Millen	MITRE, USA
Sebastian Mödersheim	IBM Zurich Research Lab, Switzerland
Mark Ryan	University of Birmingham, UK
Luca Viganò (Chair)	Università di Verona, Italy

Additional Reviewers

Misha Aizatulin
Alessandro Aldini
Andreas Bauer
Giampaolo Bella
Mario Bravetti
Roberto Carbone
Kostas Chatzikokolakis
Stéphanie Delaune
Alessandra Di Pierro
Francois Dupressoir
Deepak Garg
Claudio Guidi
Volkmar Lotz
Ilaria Matteucci
Toby Murray
Arnab Roy
Theoodor Scholte
Boris Skoric
Ben Smyth
Bruno Pontes Soares Rocha
Fred Spiessens
Angelo Troina
Mathieu Turuani
Tjark Weber

Table of Contents

A Policy Model for Secure Information Flow

Adedayo O. Adetoye and Atta Badii

School of Systems Engineering, University of Reading, Whiteknights, Berkshire, RG6 6AY, UK
a.o.adetoye@reading.ac.uk, atta.badii@reading.ac.uk

Abstract. When a computer program requires legitimate access to confidential data, the question arises whether such a program may illegally reveal sensitive information. This paper proposes a policy model to specify what information flow is permitted in a computational system. The security definition, which is based on a general notion of information lattices, allows various representations of information to be used in the enforcement of secure information flow in deterministic or nondeterministic systems. A flexible semantics-based analysis technique is presented, which uses the input-output relational model induced by an attacker's observational power, to compute the information released by the computational system. An illustrative attacker model demonstrates the use of the technique to develop a termination-sensitive analysis. The technique allows the development of various information flow analyses, parametrised by the attacker's observational power, which can be used to enforce *what* declassification policies.

1 Introduction

The problem of *secure information flow* arises when a computer program must be granted legitimate access to confidential data. When such a program, which might have access to a network or that might otherwise be able to transmit confidential information to unauthorised observers, is executed, we want assurances that only the information that we wish to reveal is released. An *information flow policy* expresses our security concern about the information release that we consider as safe. This leads to the question of how to specify *what* information release is safe. The traditional approach to the specification of information release, or rather, the lack of it, is through the *noninterference* requirement [7]. Noninterference prevents *any* flow of secret information to public areas in a multi-level security system, where information must not flow from *high* to *low*. Thus, noninterference is very restrictive and its usefulness in general practice has been argued [13]. In practice, for example, during encryption, authentication, or statistical analysis, we often want to release *some level* of information. This requires a more general policy model by which we can specify what is the safe level of information to be released. This paper proposes a lattice model to capture this property.

In [16], a taxonomy of declassification mechanisms is introduced based on *what*, *where*, *when* and by *whom* information is released. This paper is concerned about the *what* dimension of information flow, where we want to express the property that the information released by a system does not exceed certain allowed limits. Based on this observation, a *definition of security* is given, which captures the idea that a given information flow is safe.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 1–17, 2009.

1.1 Contributions

This paper contributes to the theory of secure information flow through a systematic study of lattices of information as a tool for the enforcement of *what* declassification policies. Although lattice-based approaches are often used in language-based security [14], these lattices are usually of security classes in a multi-level security system, rather than lattices of information. We demonstrate that various *representations* of information such as partial equivalence relations, families of sets, information-theoretic characterisation, and closure operators fit into the lattice model of information, unifying the various definitions under the lattice model. This means that the same partial order-based enforcement technique can be applied to all the representations.

Another contribution to the theory is an input-output *relation model*, presented as a primitive for the semantic analysis of information flow. A systematic approach to deriving the relational model from the operational semantics, which is parametric to a chosen attacker's observational power, is presented. The relational model accounts well for information flow due to nontermination, and the specific termination-sensitive analysis presented demonstrates the correct analysis of diverging programs by using the relational model.

1.2 Plan of the Paper

In Section 2 the lattice model of information is motivated, and a security definition is given which uses the lattice model to enforce *what* declassification policies. Section 3 introduces the relational model primitive as a tool for studying information flow in models of deterministic or nondeterministic systems. Section 4 uses the relational model to develop a representation of information, based on PERs, for the analysis of deterministic system models. A language-based analysis technique is presented for *While* programs with outputs to illustrate how to derive the relational model under a given attacker model in a language-based setting. Similarly to Section 4, Section 5 applies the relational model technique to develop a representation of information, based on families of sets, which captures the information that the attacker may gain when the system can be run repeatedly under fixed inputs. An extension of the *While* language with a nondeterministic construct shows the use of this information representation for information flow analysis in a nondeterministic language setting. We compare our approach with related works in Section 6. Section 7 concludes the paper.

2 Secure Information Flow

The concept of secure information flow suggests an understanding of the notions of information and information flow. A fundamental property of information is the intuitive notion of *information levels*, where we say that one piece of information is *greater* or *more informative* than another. This suggests an ordering of information, which we shall exploit in our information model and security definition. For this reason we shall model information as *lattices*, where the associated *partial order* captures the notion of information levels.

Definition 1 (Information and Information Flow). *We define* information *as elements of a complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$, where the associated partial order $\sqsubseteq$ models the relative degree of informativeness of the elements of $\mathcal{I}$, and the join operation $\sqcup$ models the combination of information.*

Information flow *with respect to the lattice $\mathcal{I}$ is defined as an* extensive *and* monotone *self map on $\mathcal{I}$. Define $\mathcal{F}lows \triangleq \{f : \mathcal{I} \rightarrow \mathcal{I} \,|\, f$ is extensive and monotone$\}$ to be the set of all information flows on $\mathcal{I}$.*

The lattice join operation, which models information combination, is idempotent, commutative, and associative. These properties agree with natural intuitions about information, since idempotency says that the combination of a piece of information with itself should yield the same information [10]. Similarly, the commutativity and associativity properties respectively agree with the intuitions that the order and grouping of information combination should not matter to the end result. Furthermore, for any $s, s' \in \mathcal{I}$, the lattice property, $s \sqsubseteq s'$ iff $s \sqcup s' = s'$, agrees with the idea that the combination of a lesser information with a greater one yields the greater information - where $s \sqsubseteq s'$ is interpreted to mean that the information s is less than or at most equal to s'.

The notion of *information flow* models how the knowledge of an attacker changes due to information release. For any initial knowledge $s \in \mathcal{I}$ that the attacker might have, $f(s)$ represents the final knowledge of the attacker due to the information flow $f \in \mathcal{F}lows$ that the attacker receives. The extensivity property of f (that is, $\forall s \in \mathcal{I}, s \sqsubseteq f(s)$) intuitively means that an attacker's knowledge may only increase by gaining information, and the monotonicity (that is, $\forall s, s' \in \mathcal{I}, s \sqsubseteq s' \implies f(s) \sqsubseteq f(s')$) means that the greater the initial knowledge of the attacker before information release the greater the knowledge afterwards.

Using this ordered structure of information, we can now define what it means for a system to have secure information flow with respect to what information the system releases and a given information flow policy[1].

Definition 2 (*What* Policies and Security). *Given the lattice $\langle \mathcal{I}, \sqsubseteq \rangle$ of information and the set $\mathcal{F}lows$ of information flows over this lattice. An information flow policy $\mathscr{P}$ with respect to the lattice $\mathcal{I}$ is a* subset *of $\mathcal{F}lows$ which specifies the permitted information flows. An information flow $f \in \mathcal{F}lows$ is said to be* permitted *or* allowed *by a policy $\mathscr{P} \subseteq \mathcal{F}lows$ iff there exists a flow function $f' \in \mathscr{P}$ such that $f \sqsubseteq f'$.*

Let P be a program, modelling a system. Furthermore, let $[\![P]\!]^{\mathcal{I}} \subseteq \mathcal{F}lows$ be the information flow property *of the system modelled by P, which describes the information flows caused by this system. The system* satisfies*, and is said to be* secure *with respect to a policy $\mathscr{P} \subseteq \mathcal{F}lows$ iff for all $f \in [\![P]\!]^{\mathcal{I}}$ there exists $f' \in \mathscr{P}$ such that $f \sqsubseteq f'$.*

The order $f \sqsubseteq f'$ between flow functions is the usual pointwise ordering of functions induced by the partial order $\sqsubseteq$ on the lattice $\mathcal{I}$. This partial order regulates the level of information that we allow a system to release. Intuitively, this definition says that the program P (or the system it models) is secure (with respect to the policy $\mathscr{P}$) only if every flow $f \in [\![P]\!]^{\mathcal{I}}$ that is caused by the system is permitted by the policy ($\exists f' \in \mathscr{P}$ such that $f \sqsubseteq f'$). This extensional view of policy enforcement abstractly describes, in terms of the information lattice order, what information flows are permitted in the system.

[1] We shall sometimes refer to an information flow policy simply as *policy* or *security policy*.

Since $\mathcal{I}$ is a complete lattice, it is easy to show that $\mathcal{F}lows$ also forms a complete lattice under the pointwise function ordering. In particular, the noninterference policy, which prevents any flow of information to the attacker may be modelled by the *identity* map ($id_{\mathcal{I}}$) on the lattice $\mathcal{I}$. This means that a system that satisfies the *noninterference policy* $\{id_{\mathcal{I}}\}$ has the information flow property that regardless of the attacker's previous level of knowledge, the final knowledge remains unchanged and the attacker is unable to benefit information by observing this noninterfering system. The baseline nature of the noninterference policy model $\{id_{\mathcal{I}}\}$ is established by the fact that $id_{\mathcal{I}}$ is the least element of the lattice $\mathcal{F}lows$ of information flows. However, it is clear that other less restrictive policies than $\{id_{\mathcal{I}}\}$ may be specified. Let us examine some example policy patterns under the lattice model of information flow.

2.1 Information Flow Policy Patterns

We examine in this section, *policy patterns*, other than the noninterference pattern, which allow deliberate, but controlled, release of information.

We may wish to have partial (but unconditional) release of information $s' \in \mathcal{I}$ but not more in a system. The required information flow property is captured by the policy $\mathscr{P} = \{f | \forall s \in \mathcal{I}, f(s) = s' \sqcup s\}$, which allows the attacker to combine its knowledge s with the declassified information s', but the attacker may not learn more than this by observing the system which is secure with respect to $\mathscr{P}$. An example of this scenario arises during password authentication, where we wish to release (unconditionally) the information about the equality or not of the stored password and the user-supplied password.

Another scheme is the *conditional release* pattern, where information (s') is released based on having some initial knowledge (s''). This is modelled by the policy $\{f\}$ where $\forall s \in \mathcal{I}, f(s) = s' \sqcup s$ if $s'' \sqsubseteq s$, and $f(s) = s$ otherwise. Under this scheme, the attacker gains some information on the condition that the attacker has at least a given initial information s''. A scenario where such a policy is needed is during decryption in a symmetric key system, where the plaintext may be learnt (the knowledge s') only when the decryption key is known (the knowledge s'').

Another pattern, called *disjunctive flow policy* - after the disjunctive flow pattern of [16], is $\{f, f' | f \not\sqsubseteq f', f' \not\sqsubseteq f\} \subseteq \mathcal{F}lows$. This policy permits at most one of f or f' to be released but not *both* at the same time. It is clear that the notion of disjunctive information flow is only meaningful for incomparable information and information flows, because whenever two information are comparable then the greater already contains the lesser information. An information flow $f'' \in \mathcal{F}lows$ is permitted by the disjunctive policy $\{f, f' \mid f \not\sqsubseteq f', f' \not\sqsubseteq f\}$, when f'' is smaller than or equal to at most one of f and f' - since f and f' are incomparable. Also, a flow $f'' \sqsupseteq f \sqcup f'$, which contains both f and f' is not permitted since there is no such $f_1 \in \{f, f' \mid f \not\sqsubseteq f', f' \not\sqsubseteq f\}$ for which $f'' \sqsubseteq f_1$.

In Definition 2 we defined the security property of a system, with respect to a policy, in terms of the system's *information flow property*. In the next section we shall show a way to derive the information flow property of a system from a relation which describes how the system transforms its inputs to publicly observable outputs.

3 The Relational System Model

The question that we want to answer is whether a system that processes confidential data is secure with respect to a given security policy. So, using a suitable representation of information, we want to check whether the information released conforms to our policy requirement. The particular choice of *representation* of information may depend on the model of the system being analysed or the kind of information that we are interested in modelling. In this paper we shall consider information flow analysis under *deterministic* and *nondeterministic* system models using *qualitative* representations of information and show that the representations fit into the lattice model of information. We note that it is possible to consider *quantitative* representations of information, such as *entropy* and other information-theoretic measures, under the lattice model of information because the measures, which are numbers, are naturally ordered.

We shall model a computational system, by an *input-output relation*: $S \subseteq \Sigma \times \mathcal{V}$, where the system accepts inputs taken from the set Σ and produces public outputs (with respect to an attacker model) in the set $\mathcal{V}$. The relation S models what the attacker observes given the supplied input. Depending on the attacker model, the system model may be *deterministic*, in which case S is a *function*. More generally, however, the model S of the system is said to be *nondeterministic* (with respect to the attacker's view) when the attacker's observation is not necessarily unique for a given input. In the sequel, we shall sometimes refer to the system modelled by the relation S simply as "system S".

Definition 3 (The Relational Model). *The input-output* relational model *of a system is defined as a relation $S \subseteq \Sigma \times \mathcal{V}$, over the set Σ of the system's inputs and the set $\mathcal{V}$ of observable outputs, where for all input $\sigma \in \Sigma$ and possible output $v \in \mathcal{V}$, $\sigma\ S\ v$ holds iff the system can produce the output v when supplied with the input σ.*

The inverse image of the relation S at $v \in \mathcal{V}$ is denoted by $S^{-1}(v) \triangleq \{\sigma \in \Sigma \mid \sigma\ S\ v\}$. When the relational model f is a *function*, we write $f : \Sigma \to \mathcal{V}$ and for any $\sigma \in \Sigma$, $f(\sigma)$ stands for the unique output observed when the input σ is supplied. We refer to f as the *functional model* of the system.

4 Analysis for Deterministic System Models

We shall use *Partial Equivalence Relations*[2] (PERs) as a qualitative representation of information for the deterministic system model. We start by motivating the use of PERs for information representation in this setting. Since the deterministic system model is a (total) function $g : \Sigma \to \mathcal{V}$, where for any input $\sigma \in \Sigma$ supplied to the system, the attacker observes $g(\sigma) \in \mathcal{V}$, the knowledge gained by the attacker can be described by the ability to *distinguish* which inputs might have been supplied based on the observed output. Thus, given the observed output $v \in \mathcal{V}$ of the system, the attacker knows that the input to the system has been taken from the set $g^{-1}(v) \subseteq \Sigma$. However, based on this observation the attacker cannot *distinguish* between the inputs σ and σ'

[2] A partial equivalence relation is a *symmetric* and *transitive* binary relation. If in addition the relation is also *reflexive*, then it is an *equivalence relation*.

if $\sigma, \sigma' \in g^{-1}(v)$. More generally, we can model all the input pairs that the attacker cannot distinguish by the *kernel* κ_g of g, which is the *equivalence relation* given by $\forall \sigma, \sigma' \in \Sigma, \sigma\ \kappa_g\ \sigma' \iff g(\sigma) = g(\sigma')$. Thus, κ_g describes the *information* (or more directly, the *ignorance*) of the attacker based on the *indistinguishability* of input pairs after observing the system's outputs. An input pair is *distinguishable* by the attacker if the pair is *not related* by the equivalence relation κ_g.

We can generalise this idea a bit further from equivalence relations to PERs to obtain certain expressive powers. Similarly to equivalence relations, we say that a PER cannot distinguish a pair if the pair is *related* by that PER, but any value that is not in the domain of definition of the PER is considered as *not possible* and therefore distinguishable. For example, we can represent the knowledge of parity of an integer secret via the equivalence relation Par defined as $\forall n, m \in \mathbb{Z}, n\, \text{Par}\, m$ iff $n \mod 2 = m \mod 2$, which distinguishes a pair of integers with different parity. However, the PER Par+ over integers, defined as $\forall n, m \in \mathbb{Z}, n\ (\text{Par+})\ m$ iff $n\, \text{Par}\, m$ and $n, m \geq 0$, represents the knowledge of *parity* and *sign* - since it only relates natural numbers (negative integers are not possible). The use of equivalence relations, and PERs in general, to describe the security property of programs is not new [11,15]. In this paper we are interested in the lattice properties for the enforcement of secure information flow.

Definition 4. *Let Σ be a set. Define PER(Σ) to be the set of all PERs over the set Σ and define the information order relation $\sqsubseteq$ on PERs such that for any $R, R' \in$ PER(Σ), $R \sqsubseteq R'$ iff for all $\sigma, \sigma' \in \Sigma$, $\sigma\ R'\ \sigma'$ implies $\sigma\ R\ \sigma'$. The associated information combination, or join operation $\sqcup$, on PER(Σ) is defined as $\sigma\ (R \sqcup R')\ \sigma'$ iff $\sigma\ R\ \sigma'$ and $\sigma\ R'\ \sigma'$. More generally, for any subset $\mathcal{R} \subseteq$ PER(Σ) let the join of $\mathcal{R}$ be the PER $\bigsqcup \mathcal{R}$, defined for all $\sigma, \sigma' \in \Sigma$ as $\sigma \bigsqcup \mathcal{R}\, \sigma'$ iff $\forall R \in \mathcal{R}$, $\sigma\ R\ \sigma'$.*

We know from the definition of $\sqsubseteq$ that $R \sqsubseteq R'$ means that R' can distinguish whatever R can, and thus R' contains more information than R. For example, Par $\sqsubseteq$ Par+ agrees with the intuition of the relative information content of these two PERs. The ordering of PERs by their information content forms a complete lattice of information.

Theorem 1. *The partially ordered set $\langle$PER$(\Sigma), \sqsubseteq, \sqcup\rangle$ is a complete lattice.*

Definition 5 (Deterministic Information flow). *Let $\mathcal{I}$ = PER(Σ) be the lattice of information for a system S whose relational model is given by a function $g_S : \Sigma \to \mathcal{V}$. The information flow property of this system may be defined as $[\![S]\!]^{\mathcal{I}} = \{f \mid \forall R \in$ PER$(\Sigma), f(R) = R \sqcup \kappa_{g_S}\}$, where κ_{g_S} is the kernel of the function g_S.*

4.1 Language-Based Analysis

In this section, we shall demonstrate the application of the analysis approach presented above in a language-based setting. We shall use the core deterministic imperative *While* language of Fig. 1, which has outputs, as our basis. The language is similar to the language of [8], and its operational semantics is fairly standard.

While expressions may be boolean-valued (with values taken from $\mathbb{B} \triangleq \{\mathtt{tt}, \mathtt{ff}\}$), or integer-valued (taken from $\mathbb{Z}$). Program states, are maps from variables to values. The evaluation of the expression e at the state σ is summarised as $\sigma(e)$. Expression

$$c ::= \texttt{skip} \mid z := e \mid \texttt{write}\; e \mid c; c \mid \texttt{if}\;(b)\;\texttt{then}\; c\; \texttt{else}\; c \mid \texttt{while}\;(b)\;\texttt{do}\; c$$

Fig. 1. The *While* Language

evaluations are performed atomically and have no side-effect on state. Program actions, ranged over by a, can either be internal ε, which is not observable ordinarily, or output (via a *write* command), where the expression value can be observed. The operational semantics is presented via transition relations between expression configurations ($\langle e, \sigma\rangle \xrightarrow{\varepsilon} \langle \sigma(e), \sigma\rangle$) and command configurations ($\langle c, \sigma\rangle \xrightarrow{a} \langle c', \sigma'\rangle$). A special, terminal command configuration $\langle \cdot, \sigma\rangle$ indicates the termination of a command in state σ. The operational semantics is shown in Fig. 2.

$$\langle \texttt{skip}, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma\rangle \qquad \langle z := e, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma[z \mapsto \sigma(e)]\rangle \qquad \langle \texttt{write}\; e, \sigma\rangle \xrightarrow{\sigma(e)} \langle \cdot, \sigma\rangle$$

$$\frac{\langle c_1, \sigma\rangle \xrightarrow{a} \langle \cdot, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{a} \langle c_2, \sigma'\rangle} \qquad \frac{\langle c_1, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{a} \langle c_1'; c_2, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{tt}, \sigma\rangle \quad \langle c_1, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle}{\langle \texttt{if}\;(b)\;\texttt{then}\; c_1\; \texttt{else}\; c_2, \sigma\rangle \xrightarrow{a} \langle c_1', \sigma'\rangle} \qquad \frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{ff}, \sigma\rangle \quad \langle c_2, \sigma\rangle \xrightarrow{a} \langle c_2', \sigma'\rangle}{\langle \texttt{if}\;(b)\;\texttt{then}\; c_1\; \texttt{else}\; c_2, \sigma\rangle \xrightarrow{a} \langle c_2', \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{ff}, \sigma\rangle}{\langle \texttt{while}\;(b)\;\texttt{do}\; c, \sigma\rangle \xrightarrow{\varepsilon} \langle \cdot, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \xrightarrow{\varepsilon} \langle \mathbf{tt}, \sigma\rangle \quad \langle c, \sigma\rangle \xrightarrow{a} \langle c', \sigma'\rangle}{\langle \texttt{while}\;(b)\;\texttt{do}\; c, \sigma\rangle \xrightarrow{a} \langle c'; \texttt{while}\;(b)\;\texttt{do}\; c, \sigma'\rangle}$$

Fig. 2. Operational Semantics of *While*

4.2 The Attacker's Observational Power

Let Σ be the set of all states of a program P. The trace of P, starting from the state $\sigma \in \Sigma$, is denoted by $t_{\langle P, \sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a_0} \langle P_1, \sigma_1\rangle \xrightarrow{a_1} \cdots$, according to the operational semantics. A trace of P at the state σ is said to *terminate* if there is a natural number n such that $t_{\langle P, \sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} \langle \cdot, \sigma'\rangle$, otherwise, the trace is said to be *nonterminating* and P *diverges* at σ.

We introduce the notion of an attacker's *observational power* (*obs*) as a map from program traces to what the attacker observes. The relative powers of two attackers A and A', modelled respectively by the observational powers obs_A and $obs_{A'}$, may be compared under the proposed model, where we say that the attacker A' is more powerful than the attacker A if there exists a function f such that $obs_A = f \circ obs_{A'}$. A more powerful attacker will gain at least as much information as a less powerful attacker. In this paper we shall consider an attacker model, whose observational power is the function $obs(\cdot)$, which is able to observe only outputs generated by *write* statements and is also able to observe the termination or not of a program. The latter assumption about the attacker appears to be strong, and is usually not modelled in language-based security. However, given the source code of a program, an attacker may be able to determine when a given program trace will not terminate without actually observing it. Modelling the ability to "observe" nontermination is also important as it may be possible to leak arbitrary amount of information via nontermination channels [2].

Definition 6 (The Semantic Attacker). *Let* $\mathbf{\Sigma}$ *be the set of all states of a* While *program* P *and let* $\xrightarrow{\varepsilon}^*$ *be the reflexive, transitive closure of the transition relation* $\xrightarrow{\varepsilon}$*. Furthermore, let* $t_{\langle P,\sigma\rangle} = \langle P,\sigma\rangle \xrightarrow{\varepsilon}^* \langle P',\sigma'\rangle \xrightarrow{a_1} \langle P_1,\sigma_1\rangle \xrightarrow{\varepsilon}^* \langle P_1',\sigma_1'\rangle \xrightarrow{a_2} \cdots$ *be a canonical representation of the trace of* P *at* $\sigma \in \mathbf{\Sigma}$*, where for all* i*,* $a_i \neq \varepsilon$*. Define the semantic attacker's observation of this trace as*

$$obs(t_{\langle P,\sigma\rangle}) \triangleq \begin{cases} \langle a_1, a_2, \cdots, \uparrow\rangle & \textit{if } P \textit{ diverges at } \sigma \\ \langle a_1, a_2, \cdots, \downarrow\rangle & \textit{otherwise.} \end{cases}$$

The set of all possible observations that the attacker can make is given by $\mathcal{V} = \{obs(t_{\langle P,\sigma\rangle}) \mid \sigma \in \mathbf{\Sigma}\}$*. The functional model induced by this attacker's observational power is thus given by* $g_P : \mathbf{\Sigma} \to \mathcal{V}$*, defined as* $g_P(\sigma) = obs(t_{\langle P,\sigma\rangle})$ *which maps the supplied input to the observed output of* P *under the attacker model.*

The information gained by the semantic attacker from P *is thus given by the equivalence relation* $\lfloor T_P \rfloor$ *over states, defined such that for any pair of states* $\sigma, \sigma' \in \mathbf{\Sigma}$*,* $\sigma \lfloor T_P \rfloor \sigma'$ *iff* $obs(t_{\langle P,\sigma\rangle}) = obs(t_{\langle P,\sigma'\rangle})$ *- the kernel of* g_P*. The information flow property of* P*, over the lattice* $\mathcal{I} = PER(\mathbf{\Sigma})$ *may therefore be defined as* $[\![P]\!]^{\mathcal{I}} = \{f \mid \forall R \in PER(\mathbf{\Sigma}), f(R) = R \sqcup \lfloor T_P \rfloor\}$*.*

The definition of $obs(\cdot)$ formalises the idea that the semantic attacker cannot observe $\xrightarrow{\varepsilon}$ transitions. For nonterminating traces, the token $\uparrow$ is introduced which, in addition to the sequence of output tokens observed on the trace, signals the divergence of the program. Similarly, the token $\downarrow$ identifies a terminating trace. Although the operational semantics definition does not have a direct notion of nontermination, $obs(\cdot)$ can account for it since it is defined over the trace.

To illustrate the definition's termination properties, let *loop* = `while` $(\mathbf{tt})$ `do skip`, and consider the programs $P_1 =$ `if` (h) `then skip else` *loop* and $P_2 =$ `write` h; *loop*. Both P_1 and P_2 insecurely reveal the boolean secret h. The analysis shows this because $\lfloor T_{P_1} \rfloor = \lfloor T_{P_2} \rfloor$ is the identity equivalence relation on h. As the analysis of P_2 demonstrates, we can easily show that by appending a trailing *loop* to any program P that always terminates, as in $P' = P$; *loop*, the information released is the same because $\lfloor T_P \rfloor = \lfloor T_{P'} \rfloor$. However, the analysis of $P_3 =$ *loop*; `write` h shows that P_3 is safe because, as the semantics shows: $\langle P_3, \sigma\rangle \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \langle P_3, \sigma\rangle \xrightarrow{\varepsilon} \cdots$, the trailing `write` h is never executed, and hence $\lfloor T_{P_3} \rfloor$ relates all states, revealing no information.

In the next section we shall demonstrate the use of *what* policies for the enforcement of secure information flow in *While* programs.

4.3 Policies For Encryption

Encryption is an important security primitive which is used widely as a security foundation in many systems. However, in order to protect the secrecy of sensitive data used during encryption, we require policies that permit encryption to be used safely. Noninterference policies cannot be used for encryption since the resulting (public) cyphertext in an encryption scheme will depend on the supplied plaintext and key which are considered secret. Thus, there remains the problem of the specification of policies that allow safe use of encryption in programs. We demonstrate in this section how our approach can be used to specify and enforce policies that allow encryption to be used safely.

We start by considering an encryption function $\mathcal{E}_1 : K \times M \to C$, which accepts a key chosen from the set K of keys and message or plaintext chosen from the set M, and produces a cyphertext in the set C. Now suppose that $\mathcal{E}_1$ is considered secure and that its implementation, which we shall denote by the expression $\boldsymbol{enc}(k, m)$, is correct, so that under any state $\sigma(\boldsymbol{enc}(k, m)) = \mathcal{E}_1(\sigma(k), \sigma(m))$. Therefore, we allow the attacker to observe the cyphertext $\boldsymbol{enc}(k, m)$ for any choice $k \in K$ of key and plaintext $m \in M$ values. Hence, we can define an equivalence relation $\lfloor \mathcal{E}_1 \rfloor$ which captures this intentional information release, where $\lfloor \mathcal{E}_1 \rfloor$ relates every pair of states σ and σ', where $\mathcal{E}_1(\sigma(k), \sigma(m)) = \mathcal{E}_1(\sigma'(k), \sigma'(m))$. The resulting information flow policy $\mathscr{P}_{\mathcal{E}_1} = \{f \mid R \in \text{PER}(\Sigma), f(R) = R \sqcup \lfloor \mathcal{E}_1 \rfloor\}$ allows the attacker to observe the ciphertext generated by a correct implementation of the encryption function. Firstly, since the implementation ***enc*** is secure, it satisfies the policy $\mathscr{P}_{\mathcal{E}_1}$.

Now consider a secure (and insecure) data backup scenario (adapted from [1]) as shown in the program listings of Fig. 3. The LHS program P_1 securely releases the encrypted data (*ctxt*) to a public output channel after encrypting the data (*data*) with the key (k). However, the RHS implementation P_2 is insecure because the programmer releases the plaintext data instead of the ciphertext. The analysis detects that the RHS program violates the policy because $\lfloor T_{P_2} \rfloor \nsubseteq \lfloor \mathcal{E}_1 \rfloor$ as required by $\mathscr{P}_{\mathcal{E}_1}$ - unless the encryption function by definition reveals the encrypted data, which violates initial assumption that it is *secure*. Thus, the analysis detects this flaw. This would be useful, for example, to a programmer who can avoid such programming error by checking his or her implementation against the desired policy.

The reason why the noninterference policy cannot be used for encryption lies in the fact that noninterference prohibits any sort of variation in the observed output from being induced by a variation in the secret input to the encryption function. However, one of the reasons why encryption is widely used as a security primitive is the fact that the security lies in the ability to protect secret data even when the encryption function is known. Thus, a good encryption function is already designed so that it is not easily invertible into its constituent arguments, although a variation in its input would cause a variation in its output for the function to be useful. The safe input-to-output variation caused by the *definition* of the encryption function is captured by the equivalence relation $\lfloor \mathcal{E}_1 \rfloor$ in the example above, which allows only the output variations due to the definition of the encryption function to be observed by the attacker.

```
ctxt := enc(k, data);          | ctxt := enc(k, data);
write ctxt;                    | write data;
```

Fig. 3. Secure versus Insecure Data Backup

Nondeterministic Encryption. In nondeterministic encryption, such as *cipher-block chaining* encryption mode, an *initialisation vector* (iv) is used along with the key and plaintext such that if a different iv is used, a different ciphertext is generated under the same key and plaintext pair. The term “nondeterministic” refers to the fact that the implementation of such encryption algorithms generally have the property that encrypting the same plaintext several times using the same key would yield different ciphertexts. Let the function $\mathcal{E}_2 : IV \times K \times M \to C$ denote such an encryption scheme, where IV is the set of initialisation vectors. We shall represent by the expression $\boldsymbol{enc}_*(iv, k, m)$ a correct implementation of $\mathcal{E}_2$. Similarly to the encryption policy above, we define $\lfloor \mathcal{E}_2 \rfloor$ as the equivalence relation which relates all states which evaluate $\boldsymbol{enc}_*(iv, k, m)$ to the same ciphertext, and the required declassification policy is similarly defined.

Now, a known problem with declassification schemes is that of *occlusion* [16], where a legitimately declassified information masks the release of other secrets. Being a *what* policy, our policy enforcement prevents such a flow by permitting only the information release that is explicitly allowed by the policy. This problem is illustrated by the program listing of Fig. 4 (adapted from [1]). Suppose that we have declassified the encryption result, then revealing the content of l_1 and l_2 through the *write* statements in the program is permitted, however the value of the boolean secret h will be released additionally by this program because the inequality of l_1 and l_2 will reveal the fact that the *then* branch was executed. The analysis shows this. Let us call this program P, its analysis $\lfloor T_P \rfloor$ has the property that whenever it relates any two states σ and σ' which disagree on the observed ciphertexts, then they must both agree to a value $\sigma(h) = \sigma'(h) = \mathtt{tt}$ of h - revealing h. Hence P violates the declassification policy ($\lfloor T_P \rfloor \not\subseteq \lfloor \mathcal{E}_2 \rfloor$) because $\lfloor \mathcal{E}_2 \rfloor$ requires, for example, that $\sigma[h \mapsto \mathbf{ff}]$ must be indistinguishable from σ, which $\lfloor T_P \rfloor$ distinguishes in this case.

```
$l_1 := \boldsymbol{enc}_*(iv_1, k, m)$;
if ($h$) then $l_2 := \boldsymbol{enc}_*(iv_2, k, m)$; else $l_2 := l_1$;
write $l_1$; write $l_2$;
```

Fig. 4. The Occlusion Problem

5 Analysis for Nondeterministic System Models

We now turn our attention to the security analyses of information flow under nondeterministic system models. We start by motivating the use of families of sets as a representation of information in the nondeterministic setting, which generalises the PER representation used earlier for the deterministic system model.

Consider a system, whose relational model is given by $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$. We can describe the information that the attacker gains on observing the output $v \in \mathcal{V}$ of the system by the inverse image of v under S. The inverse image $S^{-1}(v)$ of v represents the set of all *possible* inputs that can produce the output v in the system modelled by S, and thus describes the attacker's uncertainty about the inputs given the observation of v. It is thus easy to see that the family of sets $\{S^{-1}(v) \mid v \in \mathcal{V}\}$ models the uncertainties of the attacker under the observation of individual outputs of the system modelled by S.

In the special case that S models a deterministic system, in which case S is a function, it is clear that $\{S^{-1}(v) \mid v \in \mathcal{V}\}$ corresponds to the set of partitions of the kernel of the function S, which uniquely identifies the equivalence relation over $\mathbf{\Sigma}$ used to describe the information released in the previous sections. In this sense, the family of sets representation generalises the PER representation.

However, unlike the deterministic model presented earlier, where for any $v, v' \in \mathcal{V}$, $v \neq v'$ implies $S^{-1}(v) \cap S^{-1}(v') = \varnothing$, the inverse images are not necessarily disjoint under a nondeterministic model since the outputs resulting from any given input may not necessarily be unique. This leads to an avenue of information release in nondeterministic systems. The property that the nondeterministic system modelled by S does not necessarily partition its domain introduces the possibility that an attacker might gain further information by repeated execution of the system under a fixed input. To illustrate this, suppose $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$ models a nondeterministic system, where $\mathbf{\Sigma} = \{\sigma_1, \sigma_2, \sigma_3\}$ and $\mathcal{V} = \{v_1, v_2\}$ and where the graph of the relation S is given by $\mathit{graph}(S) = \{(\sigma_1, v_1), (\sigma_2, v_1), (\sigma_2, v_2), (\sigma_3, v_2)\}$. The model is nondeterministic since the input σ_2 can produce outputs v_1 or v_2. By observing an output v_1 the attacker learns that the input must be one of σ_1 and σ_2, as suggested by $S^{-1}(v_1) = \{\sigma_1, \sigma_2\}$. Similarly, on observing the output v_2, the attacker learns that the input is in the set $S^{-1}(v_2) = \{\sigma_2, \sigma_3\}$. However, if under a fixed input the attacker observes outputs v_1 and v_2 in different runs of the system, then the attacker confirms that the input to the system must be σ_2 - derived by taking the intersection $S^{-1}(v_1) \cap S^{-1}(v_2)$. This avenue of information leakage is not available under the deterministic system model since for a fixed input, the output of the system always remains the same. This leads us to a definition of information based on families of sets.

Definition 7 (Lattice of Possibilistic Information). *Let* $\Sigma_J = \{\Sigma_j \subseteq \mathbf{\Sigma} \mid j \in J\}$ *be a family of subsets of* $\mathbf{\Sigma}$ *indexed by a set J. Define the operation* $\langle\!\langle \cdot \rangle\!\rangle$ *on families of subsets of* $\mathbf{\Sigma}$ *as* $\langle\!\langle \Sigma_J \rangle\!\rangle \triangleq \bigcup_{K \subseteq J} \{\bigcap \Sigma_K\}$, *which closes the family under intersections. Define the* possibilistic information *set over*$\mathbf{\Sigma}$ *as*$\mathit{FAM}(\mathbf{\Sigma}) \triangleq \{\langle\!\langle \Sigma_J \rangle\!\rangle | \Sigma_J$ *is a family of subsets of* $\mathbf{\Sigma}\}$ *to be the information contained in families of sets over* $\mathbf{\Sigma}$. *For any* $\langle\!\langle \Sigma_J \rangle\!\rangle, \langle\!\langle \Sigma_K \rangle\!\rangle \in \mathit{FAM}(\mathbf{\Sigma})$ *define the join operation as* $\langle\!\langle \Sigma_J \rangle\!\rangle \sqcup \langle\!\langle \Sigma_K \rangle\!\rangle = \langle\!\langle \Sigma_{J \cup K} \rangle\!\rangle$ *and define the partial order* $\sqsubseteq$ *to be the subset ordering of families in* $\mathit{FAM}(\mathbf{\Sigma})$.

The intuition behind the partial ordering $\langle\!\langle \Sigma_J \rangle\!\rangle \sqsubseteq \langle\!\langle \Sigma_K \rangle\!\rangle$ is that every information token in $\langle\!\langle \Sigma_J \rangle\!\rangle$ (which is derivable by taking the intersection of some elements of Σ_J) is also present in $\langle\!\langle \Sigma_K \rangle\!\rangle$. This leads us to a description of information that can be derived about the nondeterministic system modelled by $S \subseteq \mathbf{\Sigma} \times \mathcal{V}$ as $\llbracket S \rrbracket \triangleq \langle\!\langle \{S^{-1}(v) \mid v \in \mathcal{V}\} \rangle\!\rangle$ which identifies the minimal sets of inputs that can produce a given output in the system. Based on this, and choosing $\mathcal{I} = \mathit{FAM}(\mathbf{\Sigma})$, we can define the possibilistic information flow as

$$\llbracket S \rrbracket^{\mathcal{I}} = \{f \mid \forall F \in \mathit{FAM}(\mathbf{\Sigma}), f(F) = F \sqcup \llbracket S \rrbracket\}.$$

The ordering of information content of elements of $\mathit{FAM}(\mathbf{\Sigma})$ forms a complete lattice of information.

Theorem 2. *The family of sets* $\langle \mathit{FAM}(\mathbf{\Sigma}), \sqsubseteq, \sqcup \rangle$ *over* $\mathbf{\Sigma}$ *representing the set of possibilistic information is a complete lattice.*

5.1 Language-Based Instantiation

We make a conservative extension to the *While* language, through the introduction of a nondeterministic choice constructor $[]$ to obtain the language *While-ND* (*While* with NonDeterminism) which exhibits nondeterministic behaviour. In *While-ND*, the program $c_1 \,[]\, c_2$ makes an invisible but arbitrary choice in the execution either command c_1 or command c_2. Consequently, the operational semantics of *While-ND* extends that of *While* as shown in Fig. 5.

$$\langle c_1 \,[]\, c_2, \sigma\rangle \xrightarrow{\varepsilon} \langle c_1, \sigma\rangle \qquad \langle c_1 \,[]\, c_2, \sigma\rangle \xrightarrow{\varepsilon} \langle c_2, \sigma\rangle$$

Fig. 5. Extending *While* with Possibilistic Nondeterminism

To deal with the fact that the trace of a *While-ND* program P is no more unique for a given starting state, the observational power is extended to sets of traces so that $obs^*(\cdot)$ is the set of observations for a given starting state, obtained by applying $obs(\cdot)$ to all the traces that can result from that state. This produces a relational model S_P of P, which relates σ to v iff there exists $v \in obs^*(t_{\langle P,\sigma\rangle})$. The resulting possibilistic analysis is also termination-sensitive.

Suppose the integer (secret) h is a parameter to the *While-ND* program given by $P = \texttt{if}\,(h = 0)\,\texttt{then}\,\texttt{skip}\,[]\,\textit{loop}\,\texttt{else}\,\texttt{skip}$. This program may either terminate or loop indefinitely when the secret value h is chosen to be zero. Thus, it is easy to see that the attacker may learn the value of h to be zero when the program fails to terminate. The set of possible observation of P is given by $\mathcal{V}_P = \{\langle\downarrow\rangle, \langle\uparrow\rangle\}$ where $\langle\downarrow\rangle$ corresponds to the observation during the terminating traces and $\langle\uparrow\rangle$ corresponds to the observation of the diverging trace. If we represent the set of program states as one-tuples: $\{(n) \mid n \in \mathbb{Z}\}$, then the relational model of P is $S_P \subseteq \Sigma \times \mathcal{V}_P$, whose graph is given by $\{((0), \langle\uparrow\rangle), ((n), \langle\downarrow\rangle) \mid n \in \mathbb{Z}\}$. Thus, we have the following inverse images: $S_P^{-1}(\langle\uparrow\rangle) = \{(0)\}$ and $S_P^{-1}(\langle\downarrow\rangle) = \Sigma$, and $[\![S_P]\!] = \{\{(0)\}, \Sigma\}$ reflecting the fact that the attacker can learn when the secret value is zero. Now consider another program $P_A = \texttt{if}\,(h = 0)\,\texttt{then}\,\texttt{skip}\,[]\,\textit{loop}\,\texttt{else}\,\textit{loop}$. In this case, the observation of *termination* reveals to the attacker that the value of the integer secret h is zero. The analysis is similar to that of P, but now we have $graph(S_{P_A}) = \{((0), \langle\downarrow\rangle), ((n), \langle\uparrow\rangle) | n \in \mathbb{Z}\}$ and $S_{P_A}^{-1}(\langle\downarrow\rangle) = \{(0)\}$ and $S_{P_A}^{-1}(\langle\uparrow\rangle) = \Sigma$. Thus, $[\![S_{P_A}]\!] = \{\{(0)\}, \Sigma\}$, and it is intuitive P_A releases the same information as P.

We can define a noninterference policy, which prevents any information from being gained about h in the examples above. This policy is given by $\mathscr{P}_\Sigma \triangleq \{f \mid \forall F \in \textit{FAM}(\Sigma), f(F) = F\}$, which is modelled by the identity map on the lattice $\textit{FAM}(\Sigma)$. We see that both P and P_A above violate this policy. This is clear, for example, given an initial knowledge $\{\Sigma\}$ of the attacker representing lack of information, the attacker learns $\{\{(0)\}, \Sigma\} \not\sqsubseteq \{\Sigma\}$ through these programs.

Now consider the program $P_B = \texttt{if}\,(h = 0)\,\texttt{then}\,\texttt{skip}\,[]\,\textit{loop}\,\texttt{else}\,\texttt{skip}\,[]$ *loop* which may or may not terminate regardless of the chosen value of h. Intuitively, this program should not reveal any information to the attacker as its behaviour is independent of the choice of h. This is confirmed by the analysis because

$graph(S_{P_B}) = \{((n), \langle\downarrow\rangle), ((n), \langle\uparrow\rangle) \mid n \in \mathbb{Z}\}$. Thus, $S^{-1}_{P_B}(\langle\downarrow\rangle) = S^{-1}_{P_B}(\langle\uparrow\rangle) = \Sigma$. This means that $\llbracket S_{P_B} \rrbracket = \{\Sigma\}$, showing the fact that the attacker learns nothing by observing the execution of P_B. Thus, the program P_B satisfies the noninterference policy $\mathscr{P}_\Sigma$ defined above.

Now suppose that P_C is a *While-ND* program which always terminates. Similarly to the analysis under deterministic programs, the information flow of P_C is preserved in the program $P' \triangleq P_C; loop$. This is easy to see because there is set isomorphism between the sets $\mathcal{V}_{P_C}$ and $\mathcal{V}_{P'}$ of outputs of P_C and P' respectively, which appends $\uparrow$ to all $\langle a\rangle \in \mathcal{V}_{P_C}$ such that $\forall\sigma \in \Sigma, \sigma\ S_{P_C}\ \langle a\rangle \iff \sigma\ S_{P'}\ \langle a, \uparrow\rangle$ where $S_{P_C} \subseteq \Sigma \times \mathcal{V}_{P_C}$ and $S_{P'} \subseteq \Sigma \times \mathcal{V}_{P'}$ are respectively the relational models of P_C and P'. Hence, we have $\llbracket S_{P_C} \rrbracket = \llbracket S_{P'} \rrbracket$. Finally, let P_D be a *While-ND* program such that $P' \triangleq loop; P_D$. Like the deterministic analysis, this program reveals no information since for all $\sigma \in \Sigma, \sigma\ S_{P'}\langle\uparrow\rangle$ holds and hence $\llbracket S_{P'} \rrbracket = \{\Sigma\}$.

5.2 Information-Theoretic Representation

We can perform information-theoretic analyses under the relational model $S \subseteq \Sigma \times \mathcal{V}$ of a system by assigning probability distributions to the input space Σ to model the attacker's uncertainty about the choice of inputs, and by considering the probability distribution of $\mathcal{V}$ induced by the execution of the system. Because of space restrictions, specific information-theoretic analysis techniques are not shown in this paper. However, we note that quantitative measures, such as Shannon's entropy measure, which describe the attacker's uncertainty about the input distribution before and after observing program outputs can be arranged on a lattice according to the order of the quantitative amount of information that is released about the system inputs. This observation allows us to use the lattice-based approach to enforce security under this setting, where the amount of information that is permitted to be released about a given secret is captured by the numerical order relation $\leq$ on the quantitative information measure.

6 Related Work

This paper falls into the area of language-based security, which is an established and active research area [14]. We have proposed a lattice-theoretic approach to the enforcement of *what* declassification policies [16,17]. Several approaches, such as in [4,5,6,9,11,15], have been applied to study the *what* dimension of information release. While these approaches study the properties of specific representation of information used, we study the problem from a more general viewpoint of information as a lattice structure which can be used to enforce *what* policies. The resulting lattice-based policy model has been shown to be capable of handling controlled information release, such as during encryption, and it does not suffer from the *occlusion* problem.

In [15], PERs are used to describe the security properties of a program. Given a function $f : A \to B$ which is a Scott-style denotation of a program P (information flow due to nontermination is handled by requiring that secret inputs do not affect the termination behaviour), the security property of P is stated as a PER-transformer $(\!|f|\!) : R_A \twoheadrightarrow R_B$, where $R_A \in \mathrm{PER}(A)$ and $R_B \in \mathrm{PER}(B)$. The property $(\!|f|\!) : R_A \twoheadrightarrow R_B$ holds iff for all

$a, a' \in A, a\, R_A\, a' \implies f(a)\, R_B\, f(a')$. Thus, $(\!|f|\!) : R_A \rightarrowtriangle R_B$ may be interpreted as a policy which P satisfies. If we assume that $A = B = \Sigma = H \times L$ represents the set of all states of P, partitioned to the *high*-secret (H) and *low*-public (L) parts, then P is said to be (noninterference) secure iff $(\!|f|\!) : \mathit{all} \bullet \mathit{id} \rightarrowtriangle \mathit{all} \bullet \mathit{id}$, where $\sigma_{\downarrow L}$ is the projection of the state σ to L and for all $\sigma, \sigma' \in \Sigma, \sigma\ \mathit{all} \bullet \mathit{id}\ \sigma'$ iff $\sigma_{\downarrow L} = \sigma'_{\downarrow L}$. The meaning of $(\!|f|\!) : \mathit{all} \bullet \mathit{id} \rightarrowtriangle \mathit{all} \bullet \mathit{id}$ is that the attacker which can only observe the public part of state before $P's$ execution and on its termination cannot distinguish two runs which agree on the initial L-input. This corresponds to an observational model under our approach, which for any initial input state σ is given by $\mathit{obs}_{\rightarrowtriangle}(t_{\langle P,\sigma\rangle}) = \langle \sigma_{\downarrow L}, f(\sigma)_{\downarrow L}\rangle$. Consequently, the information released is characterised by the PER defined such that $\sigma, \sigma' \in \Sigma, \sigma \lfloor T_{\langle P, \rightarrowtriangle\rangle} \rfloor \sigma' \iff \mathit{obs}_{\rightarrowtriangle}(t_{\langle P,\sigma\rangle}) = \mathit{obs}_{\rightarrowtriangle}(t_{\langle P,\sigma'\rangle})$. This definition computes the least PER $\lfloor T_{\langle P, \rightarrowtriangle\rangle} \rfloor$, that is, the most refined policy for which P is secure under the attacker model $\mathit{obs}_{\rightarrowtriangle}(\cdot)$, because $(\!|f|\!) : R \bullet \mathit{id} \rightarrowtriangle \mathit{all} \bullet \mathit{id} \implies \lfloor T_{\langle P, \rightarrowtriangle\rangle} \rfloor \sqsubseteq R \bullet \mathit{id}$. For any $R \bullet \mathit{id}$ that is strictly less than $\lfloor T_{\langle P, \rightarrowtriangle\rangle} \rfloor$, P will produce outputs that can be distinguished by $\mathit{obs}_{\rightarrowtriangle}(\cdot)$ under some variation of the H-projection of inputs that are related by R.

The abstract noninterference definition of [6] introduces attacker models as abstract interpretations which can observe only properties of data in the concrete domain. The concrete domain is partitioned into two sets H and L, which represent the domain of secret and public values respectively, and state is modelled as tuples in $\Sigma = H \times L$. The attacker is modelled as a pair of abstractions $\langle \eta, \rho \rangle$, where $\eta, \rho \in \mathit{uco}(\mathcal{P}(L))$ are upper closure operators (*extensive*, *monotone*, and *idempotent* maps) on the powerset lattice of public values ordered by subset inclusion. The closure operators η and ρ model what the attacker can observe about the program's public inputs and outputs respectively. The concrete semantics of the program P is formalised using *angelic denotational semantics*, which associates an input-output function, $[\![P]\!] : \Sigma \to \Sigma$, with P and ignores nontermination. Furthermore, the observation of (public) values occur at the beginning of program execution and on program termination. To slightly simplify the notations, we shall denote the concrete semantics of P as a map $[\![P]\!] : H \times L \to L$, throwing away the H projection of state on termination, which is not used.

Our observational model is more general since we place no restriction on the nature of the observational power function, as opposed to the requirement in [6] where they must be closure operators. Furthermore, our observational model is not restricted to the observation of values at the beginning and end of program execution. In particular, the attacker $\langle \eta, \rho \rangle$ may be obtained under our model by defining an observational power function on traces, where for any $\sigma, \hat{\sigma} \in \Sigma$, and trace $t_{\langle P,\sigma\rangle} = \langle P, \sigma\rangle \xrightarrow{a} \cdots \xrightarrow{a'} \langle \cdot, \hat{\sigma}\rangle$ we have $\mathit{obs}_{\langle \eta, \rho\rangle}(t_{\langle P,\sigma\rangle}) = \langle \eta(\{\sigma_{\downarrow L}\}), \rho(\{\hat{\sigma}_{\downarrow L}\})\rangle$. This definition says that the attacker only observes the η-property of the L-projection of the initial state and the ρ-property of the L-projection of the terminating state of P. Consequently, the information released under this observational model is the PER $\lfloor T_{P_{\langle \eta, \rho\rangle}} \rfloor$ over Σ defined such that for any $\sigma, \sigma' \in \Sigma$, $\sigma \lfloor T_{P_{\langle \eta, \rho\rangle}} \rfloor \sigma'$ iff $\mathit{obs}_{\langle \eta, \rho\rangle}(t_{\langle P,\sigma\rangle}) = \mathit{obs}_{\langle \eta, \rho\rangle}(t_{\langle P,\sigma'\rangle})$. It is thus clear that for any $\sigma, \sigma' \in \Sigma$, where $\langle P, \sigma\rangle \xrightarrow{a_1} \cdots \xrightarrow{a_n} \langle \cdot, \hat{\sigma}\rangle$ and $\langle P, \sigma'\rangle \xrightarrow{a'_1} \cdots \xrightarrow{a'_m} \langle \cdot, \hat{\sigma}'\rangle$ we have

$$\begin{aligned}\sigma \left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor \sigma' &\iff \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \wedge \rho(\{\hat{\sigma}_{\downarrow L}\}) = \rho(\{\hat{\sigma}'_{\downarrow L}\}) \\ &\implies \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\{\hat{\sigma}_{\downarrow L}\}) = \rho(\{\hat{\sigma}'_{\downarrow L}\}).\end{aligned}$$

By this we immediately obtain the narrow abstract noninterference (NANI) definition:

$$[\eta]P[\rho] \iff \forall \sigma, \sigma' \in \Sigma, \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho([\![P]\!](\sigma)) = \rho([\![P]\!](\sigma')).$$

Thus, $\left\lfloor T_{P_{\langle \eta,\rho\rangle}} \right\rfloor$ is the least PER over states for which any pair of states that it relates satisfies NANI in P.

The NANI definition causes what is referred to as "deceptive flows", when η-undistinguished public input values cause a variation which makes P to violate NANI. In order to deal with this problem, abstractions of L values are passed as program parameters and another abstraction $\phi \in uco(\mathcal{P}(H))$ is introduced on the input secret values. This results in the abstract noninterference (ANI) property, $[\eta]P(\phi \rightsquigarrow [\!]\rho)$, of [6].

Let $\sigma \in \Sigma$ be a state, and define the set $\Sigma_\sigma^{\eta,\phi}$ of L-projections of the terminating states of P due to the execution of P from any starting state which agrees with σ on the η-property of the L-projection and on the ϕ-property of the H-projection to be

$$\Sigma_\sigma^{\eta,\phi} \triangleq \left\{ \hat{\sigma}_{\downarrow L} \;\middle|\; \begin{array}{l} \sigma'' \in \Sigma . \langle P, \sigma'' \rangle \xrightarrow{a} \cdots \xrightarrow{a'} \langle \cdot, \hat{\sigma} \rangle . \\ \eta(\{\sigma''_{\downarrow L}\}) = \eta(\{\sigma_{\downarrow L}\}), \phi(\{\sigma''_{\downarrow H}\}) = \phi(\{\sigma_{\downarrow H}\}). \end{array} \right\}$$

In [6] the ANI property, $[\eta]P(\phi \rightsquigarrow [\!]\rho)$, is now defined to hold iff for all $\sigma, \sigma' \in \Sigma$, $\eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi})$.

We can obtain this observational model under our framework by defining an observational power function on traces, such that for any $\sigma \in \Sigma$, and terminating trace $t_{\langle P,\sigma\rangle}$ we have $obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma\rangle}) = \langle \eta(\{\sigma_{\downarrow L}\}), \rho(\Sigma_\sigma^{\eta,\phi})\rangle$. This definition requires that no public output can be distinguished by ρ for any initial state which is L-indistinguishable from to σ under η and H-indistinguishable from σ under ϕ. Thus, as usual, the information released under our relational model is the PER $\left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor$ over Σ defined such that for any $\sigma, \sigma' \in \Sigma$, $\sigma \left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor \sigma'$ iff $obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma\rangle}) = obs_{\langle \eta,\phi,\rho\rangle}(t_{\langle P,\sigma'\rangle})$. Hence, for all $\sigma, \sigma' \in \Sigma$, we have that

$$\begin{aligned}\sigma \left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor \sigma' &\iff \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \wedge \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi}) \\ &\implies \eta(\{\sigma_{\downarrow L}\}) = \eta(\{\sigma'_{\downarrow L}\}) \implies \rho(\Sigma_\sigma^{\eta,\phi}) = \rho(\Sigma_{\sigma'}^{\eta,\phi}).\end{aligned}$$

By this we obtain ANI property $[\eta]P(\phi \rightsquigarrow [\!]\rho)$, where $\left\lfloor T_{P_{\langle \eta,\phi,\rho\rangle}} \right\rfloor$ is the least PER over Σ, for which any pair of states that it relates satisfies ANI in P. We note that this property, which prevents the attacker from gaining information ϕ about secret inputs (see [12]), can also be arranged on a lattice of information, in particular, because $\phi \in uco(\mathcal{P}(H))$ forms a complete lattice.

In [3,8], action-based operational semantics approaches are used in deterministic language settings to check program security, similar to our labelled-transition semantics.

However, the analyses are not termination-sensitive. Furthermore, these approaches assume a fixed attacker model, whereas analysis is parametrised by a chosen attacker's observational power under our approach. With respect to the model of information, the gradual release knowledge of [3] is modelled by sets $\mathit{\Sigma} \subseteq \mathbf{\Sigma}$, which represent the knowledge (more precisely, the uncertainty) of the attacker at any point in time. These sets shrink over time and the knowledge is monotone, which agrees with the extensivity and monotonicity properties our information flow function definition. However, the PER over $\mathbf{\Sigma}$ representation generalises the subsets of $\mathbf{\Sigma}$ representation, since for any $\mathit{\Sigma} \subseteq \mathbf{\Sigma}$, there is a PER $R_{\mathit{\Sigma}}$ which encodes this set, where $\sigma \; R_{\mathit{\Sigma}} \; \sigma'$ holds iff $\sigma, \sigma' \in \mathit{\Sigma}$. Hence, every instantiation of knowledge in [3] can be encoded as PERs.

7 Conclusion

We have presented a policy model for secure information flow based on lattices of information to enforce *what* declassification policies. We demonstrated that various representations of information such as PERs, families of sets, information-theoretic measures, and closure operators may be unified under the lattice model of information, providing us with a uniform way to enforce policies based on the lattice order. A termination-sensitive analysis method was also presented, which is parametric to a chosen attacker model, and which derives a system's information flow property from the operational semantics. A static analysis technique and type system is currently being developed as an application of the relational model approach. An area of future work is to study the integration of the lattice-of-information model with other mechanisms such as security classes in a multi-level security system for the enforcement of secure information flow. Such an integration would allow us to express not only *what* properties in policies, but also *who* properties.

References

1. Askarov, A., Hedin, D., Sabelfeld, A.: Cryptographically-masked flows. Theoretical Computer Science 402(2-3), 82–101 (2008)
2. Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 333–348. Springer, Heidelberg (2008)
3. Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: IEEE Symposium on Security and Privacy, pp. 207–221. IEEE Computer Society, Los Alamitos (2007)
4. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15(3), 321–371 (2007)
5. Cohen, E.S.: Information transmission in computational systems. In: SOSP 1977: Proceedings of the sixth ACM symposium on Operating systems principles, pp. 133–139. ACM Press, New York (1977)
6. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 186–197. ACM Press, New York (2004)

7. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, pp. 11–20. IEEE Computer Society Press, Los Alamitos (1982)
8. Le Guernic, G., Banerjee, A., Jensen, T.P., Schmidt, D.A.: Automata-based confidentiality monitoring. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 75–89. Springer, Heidelberg (2006)
9. Joshi, R., Leino, K.R.M.: A semantic approach to secure information flow. Science of Computer Programming 37(1-3), 113–138 (2000)
10. Kohlas, J.: Information Algebras: Generic Structures for Inference. Springer, Heidelberg (2003)
11. Landauer, J., Redmond, T.: A lattice of information. In: Proceedings of the Computer Security Foundations Workshop VI (CSFW 1993), Washington, Brussels, Tokyo, pp. 65–70. IEEE, Los Alamitos (1993)
12. Mastroeni, I.: On the Rôle of abstract non-interference in language-based security. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 418–433. Springer, Heidelberg (2005)
13. Ryan, P., McLean, J., Millen, J., Gligor, V.: Non-interference, who needs it? In: 14th IEEE Computer Security Foundations Workshop (CSFW 2001), Washington, Brussels, Tokyo, pp. 237–240. IEEE, Los Alamitos (2001)
14. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
15. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation 14(1), 59–91 (2001)
16. Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: CSFW 2005: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW 2005), Washington, DC, USA, pp. 255–269. IEEE Computer Society, Los Alamitos (2005)
17. Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. Journal of Computer Security (2007)

A General Framework for Nondeterministic, Probabilistic, and Stochastic Noninterference

Alessandro Aldini and Marco Bernardo

University of Urbino "Carlo Bo" – Italy
Information Science and Technology Institute

Abstract. We introduce a notion of stochastic noninterference aimed at extending the classical approach to information flow analysis with fine-grain information describing the temporal behavior of systems. In particular, we refer to a process algebraic setting that joins durational activities expressing time passing through exponentially distributed random variables, zero duration activities allowing for prioritized/ probabilistic choices, and untimed activities with unspecified duration. In this setting unifying time, priority, probability, and nondeterminism, we highlight the expressive power of stochastic noninterference with respect to the existing definitions of nondeterministic and probabilistic noninterference. From this comparison, we obtain that stochastic noninterference turns out to be very strict and limiting in real-world applications and, therefore, requires the use of relaxation techniques. Among them we advocate performance evaluation as a means for achieving a reasonable balance between security requirements and quality.

1 Fine-Grain Models and Noninterference

Information flow analysis is a basic approach to the verification of security properties of systems which, in general, require to control who has access to what and when. Among the several conditions that describe the characteristics of unauthorized information flows, called covert channels, one of the most interesting, for its intuitive and wide-used idea, is the noninterference requirement [11]. Very briefly, in a multi-level secure system, the user at the high security level should not be able to affect what the user at the low security level can observe. Independently of the specific formalization of this notion, the underlying approach is based on the idea that checking noninterference is actually checking the indistinguishability of the different low-level views of the system that are obtained by changing the high-level behavior, see e.g. [9,17,22] and the references therein.

Along the 90s generalized notions of noninterference were designed to analyze deterministic and nondeterministic systems. However, it was immediately clear that moving to a quantitative framework including fine-grain information, such as probability distributions associated with event execution, augments the distinguishing power of the observer [3,14,18,19]. More recently, the awareness that perfect noninterference is difficult to achieve has urged to estimate the information flow by employing this quantitative information and a relaxed notion of

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 18–33, 2009.

noninterference, see e.g. [4] and the references therein. A quantitative notion of noninterference can be based not only on probabilistic information, but also on temporal information. In particular, in this paper we refer to the representation of time passing that uses non-negative random variables, which is particularly appropriate when the time taken by an event fluctuates according to some probability distribution. Among the many probability distributions that can be used to model event timing, we concentrate on exponential distributions, which are equipped with a widely developed theory.

As known in the probabilistic setting, the more information is added to the system, the higher the number of potential vulnerabilities revealed through fine-grain notions of noninterference. Hence, a noninterference property taking into account the timing of events, described through exponentially distributed random variables, would be sensitive to stochastically timed covert channels. In the literature, some proposals concerned with discrete-time extensions of noninterference have been proposed, e.g. in the setting of real-time process algebra [8] and probabilistic timed automata [7,12], while other time-based aspects of noninterference have been investigated in the general setting of Shannon information theory, see e.g. [13]. However, to the best of our knowledge, no formal definition of noninterference has been proposed in the case of stochastic time.

This paper faces the problem of defining noninterference in the setting of a stochastic process algebra encompassing nondeterminism, probability, priority, and stochastic time. The semantics of this paradigm is defined in terms of a fine-grain notion of bisimulation that extends the classical bisimulation relation of Milner [15]. In this general framework, it is possible to formalize not only stochastic noninterference properties, but also to evaluate the expressive power that is gained when adding fine-grain information in an incremental way. The result we obtain is that when moving from nondeterministic noninterference to fine-grain definitions of noninterference, security constraints become severe and hard to accomplish in practice. In the most restrictive scenario where temporal behavior and stochastic noninterference come into play, we will see that these constraints may be impractical, so that in real-world applications we need relaxation techniques aiming at trading the amount of (unavoidable) information leakage with the complexity of the adopted securing countermeasures.

We will discuss the expressive power of the different notions of noninterference through a client-server example with two access clearance levels, high and low. The clients compete for the resource on the basis of their access clearance (assume that high-level clients take precedence over low-level clients). The service center is composed of two buffers of capacity 1 – one for each security level – and one shared server providing a service to the clients of both levels. This toy example is more than enough to highlight the severeness of the constraints to which a completely secure system should be subjected.

The paper is organized as follows. In Sect. 2 we describe the stochastic process algebra framework, on the base of which in Sect. 3 we introduce the stochastic version of noninterference together with a comparison with the nondeterministic/probabilistic setting. Some conclusions are drawn in Sect. 4.

2 Stochastic Process Algebra Framework

The general framework for the analysis of fine-grain noninterference we use is based on a Markovian process calculus extended with prioritized/weighted immediate actions, in which multiway communication is based on a mixture of the generative and reactive models of [10]. All these features, which are equipped with a widely developed theory, provide the expressiveness needed to model fine-grain details of real systems.

In the following, we explain the syntax and semantics of this calculus, how to derive nondeterministic and probabilistic sub-calculi, and the bisimulation semantics.

2.1 Markovian Process Calculus

The basic elements of the calculus are durational actions. They are represented as pairs of the form $<a, \tilde{\lambda}>$, where a is the action name and $\tilde{\lambda}$ is the action rate. There are three kinds of actions: exponentially timed, immediate, and passive.

Exponentially timed actions are of the form $<a, \lambda>$ with $\lambda \in \mathbb{R}_{>0}$. The duration of each such action is exponentially distributed with parameter equal to the action rate (hence its average duration is the inverse of its rate). Whenever several exponentially timed actions are enabled, the race policy is adopted, hence the action that is executed is the fastest one. As a consequence, the execution probability of an exponentially timed action is proportional to its rate.

Immediate actions are of the form $<a, \infty_{l,w}>$, where $l \in \mathbb{N}_{>0}$ is the priority level and $w \in \mathbb{R}_{>0}$ is the weight. Each immediate action has duration zero and takes precedence over exponentially timed actions, which are assumed to have priority level 0. Whenever several immediate actions are enabled, the generative preselection policy is adopted. This means that the lower priority immediate actions are discarded, whereas each of the highest priority immediate actions is given an execution probability equal to the action weight divided by the sum of the weights of all the highest priority immediate actions.

Passive actions are of the form $<a, *_{w}^{l'}>$, where $l' \in \mathbb{N}$ is the priority constraint and $w \in \mathbb{R}_{>0}$ is the weight. Each passive action with priority constraint l' can synchronize only with another passive action having the same name and the same priority constraint, or with a non-passive action having the same name and priority level l such that $l = l'$. Whenever several passive actions are enabled, the reactive preselection policy is adopted. This means that, within every set of enabled passive actions having the same name, each such action is given an execution probability equal to the action weight divided by the sum of the weights of all the actions in the set. Instead, the choice among passive actions having different names is nondeterministic. Likewise, the choice between a passive action and a non-passive action is nondeterministic.

Definition 1. *Let $Act = Name \times Rate$ be a set of actions, with Name being a set of action names containing a distinguished symbol τ for the invisible action and $Rate = \mathbb{R}_{>0} \cup \{\infty_{l,w} \mid l \in \mathbb{N}_{>0} \wedge w \in \mathbb{R}_{>0}\} \cup \{*_{w}^{l'} \mid l' \in \mathbb{N} \wedge w \in \mathbb{R}_{>0}\}$ being* *a set of action rates (ranged over by $\tilde{\lambda}$). The set $\mathcal{L}$ of process terms is generated by the following syntax:*

$$P ::= \underline{0} \mid <a, \tilde{\lambda}>.P \mid P + P \mid A \mid P/L \mid P \parallel_S P$$

where $L, S \subseteq Name - \{\tau\}$ and A is a process constant defined through the (possibly recursive) equation $A \stackrel{\Delta}{=} P$.

The semantics for the set $\mathcal{P}$ of closed and guarded process terms of $\mathcal{L}$ is defined in the usual operational style. The behavior of each term P is given by a labeled multitransition system $[\![P]\!]$, whose states correspond to process terms and whose transitions – each of which has a multiplicity equal to the number of proofs of its derivation – are labeled with actions.

The null term $\underline{0}$ cannot execute any action, hence the corresponding semantics is given by a state with no transitions. The action prefix term $<a, \tilde{\lambda}>.P$ can execute an action with name a and rate $\tilde{\lambda}$ and then behaves as P:

$$<a, \lambda>.P \xrightarrow{a,\lambda} P \qquad <a, \infty_{l,w}>.P \xrightarrow{a,\infty_{l,w}} P \qquad <a, *_w^{l'}>.P \xrightarrow{a,*_w^{l'}} P$$

The alternative composition $P_1 + P_2$ behaves as either P_1 or P_2 depending on whether P_1 or P_2 executes an action first:

$$\frac{P_1 \xrightarrow{a,\tilde{\lambda}} P'}{P_1 + P_2 \xrightarrow{a,\tilde{\lambda}} P'} \qquad \frac{P_2 \xrightarrow{a,\tilde{\lambda}} P'}{P_1 + P_2 \xrightarrow{a,\tilde{\lambda}} P'}$$

The process constant A behaves as the right-hand side process term in its defining equation:

$$\frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad A \stackrel{\Delta}{=} P}{A \xrightarrow{a,\tilde{\lambda}} P'}$$

The hiding term P/L behaves as P with the difference that the name of every action executed by P that belongs to L is turned into τ:

$$\frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad a \in L}{P/L \xrightarrow{\tau,\tilde{\lambda}} P'/L} \qquad \frac{P \xrightarrow{a,\tilde{\lambda}} P' \quad a \notin L}{P/L \xrightarrow{a,\tilde{\lambda}} P'/L}$$

The parallel composition $P_1 \parallel_S P_2$ behaves as P_1 in parallel with P_2 as long as actions are executed whose name does not belong to S:

$$\frac{P_1 \xrightarrow{a,\tilde{\lambda}} P_1' \quad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a,\tilde{\lambda}} P_1' \parallel_S P_2} \qquad \frac{P_2 \xrightarrow{a,\tilde{\lambda}} P_2' \quad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a,\tilde{\lambda}} P_1 \parallel_S P_2'}$$

Generative-reactive synchronizations are forced between any non-passive action executed by one term and any passive action executed by the other term that have the same name belonging to S and the same priority level/constraint:

$$\frac{P_1 \xrightarrow{a,\lambda} P'_1 \quad P_2 \xrightarrow{a,*^{0}_{w}} P'_2 \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda\cdot\frac{w}{weight(P_2,a,0)}} P'_1 \parallel_S P'_2} \qquad \frac{P_1 \xrightarrow{a,*^{0}_{w}} P'_1 \quad P_2 \xrightarrow{a,\lambda} P'_2 \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\lambda\cdot\frac{w}{weight(P_1,a,0)}} P'_1 \parallel_S P'_2}$$

$$\frac{P_1 \xrightarrow{a,\infty_{l,v}} P'_1 \quad P_2 \xrightarrow{a,*^{l}_{w}} P'_2 \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,v}\cdot\frac{w}{weight(P_2,a,l)}} P'_1 \parallel_S P'_2} \qquad \frac{P_1 \xrightarrow{a,*^{l}_{w}} P'_1 \quad P_2 \xrightarrow{a,\infty_{l,v}} P'_2 \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,\infty_{l,v}\cdot\frac{w}{weight(P_1,a,l)}} P'_1 \parallel_S P'_2}$$

where $weight(P, a, l) = \sum\{\!| \, w \mid \exists P' \in \mathcal{P}. \, P \xrightarrow{a,*^{l}_{w}} P' \, |\!\}$. Following [6], the adopted synchronization discipline mixes generative and reactive probabilistic models. Among all the enabled timed actions whose names belong to a synchronization set, the proposal of an action name is generated after a selection based on the rates of those actions. Then the enabled passive actions that have the same name as the proposed one react by means of a selection based on their weights. Finally, the timed action winning the generative selection and the passive action winning the reactive selection synchronize with each other.

Reactive-reactive synchronizations are forced between any two passive actions of the two terms that have the same name belonging to S and the same priority constraint:

$$\frac{P_1 \xrightarrow{a,*^{l}_{w_1}} P'_1 \quad P_2 \xrightarrow{a,*^{l}_{w_2}} P'_2 \quad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a,*^{l}_{\frac{w_1}{weight(P_1,a,l)}\cdot\frac{w_2}{weight(P_2,a,l)}\cdot(weight(P_1,a,l)+weight(P_2,a,l))}} P'_1 \parallel_S P'_2}$$

We will use the abbreviation $P\backslash S$ to stand for $P \parallel_S \underline{0}$, which intuitively describes the behavior of a restriction operator. Moreover, if $P \xrightarrow{a_1,\tilde{\lambda}_1} \ldots \xrightarrow{a_n,\tilde{\lambda}_n} P'$, with $n \in \mathbb{N}$, then we say that P' is a derivative of P and we denote with $Der(P)$ the set of derivatives of P. We denote with $\mathcal{P}_{\mathrm{pc}}$ the set of performance closed process terms of $\mathcal{P}$, i.e. those terms with no passive transitions. The stochastic process underlying $P \in \mathcal{P}_{\mathrm{pc}}$ is a continuous-time Markov chain (CTMC) possibly extended with immediate transitions. States having outgoing immediate transitions are called vanishing as the sojourn time in these states is zero. In order to retrieve a pure CTMC stochastically equivalent to an extended CTMC, all the vanishing states can be adequately eliminated.

Example 1. Let us exemplify the use of the operators through the running example modeling the client-server system. The arrival process for clients of level $k \in \{L, H\}$, where L stands for low access level and H stands for high access level, can be modeled as:

$$A_L \stackrel{\Delta}{=} <a_L, \lambda_L>.A_L \quad \text{and} \quad A_H \stackrel{\Delta}{=} <a_H, \lambda_H>.A_H$$

where λ_k is the parameter of the Poisson arrival process for clients of access level k. The buffer for clients of access level k – which we assume to have capacity 1 – can be modeled as a completely passive entity as follows:

$$B_k \stackrel{\Delta}{=} <a_k, *_w^0>.<d_k, *_w^{f(k)}>.B_k$$

where, for instance, we assume $f(L) = 1$ and $f(H) = 2$ to denote that high-level requests pre-empt low-level ones. The shared server can be modeled as follows:

$$S \stackrel{\Delta}{=} \sum_{k \in \{L,H\}} <d_k, \infty_{f(k),w}>.<s_k, \mu_k>.S$$

where μ_k is the service rate for clients of access level k. Hence, the overall client-server system can be modeled as follows:

$$CS \stackrel{\Delta}{=} (A_L \parallel_{\emptyset} A_H) \parallel_{\{a_L,a_H\}} (B_L \parallel_{\emptyset} B_H) \parallel_{\{d_L,d_H\}} S$$

■

2.2 Nondeterministic and Probabilistic Sub-calculi

From the calculus above it is straightforward to derive two sub-calculi: a nondeterministic one and a probabilistic one. The former is obtained by removing all the fine-grain information related to time, probabilities, and priorities. Syntactically, durational actions of the form $<a, \lambda>$ and $<a, \infty_{l,w}>$ are replaced by the output action a, while passive actions of the form $<a, *_w^l>$ become the input action a_*. Semantically, the result is a classical calculus with CSP-like communication, whose underlying model is a labeled transition system. We denote with $nd(P)$ the nondeterministic version of process P obtained in this way.

The latter sub-calculus is obtained by removing all the fine-grain information related to time and priorities. Syntactically, the exponentially timed action $<a, \lambda>$ is mapped to the immediate action $<a, \infty_{0,\lambda}>$. In this way, the action rate is interpreted as the action weight that governs the execution probability of the action. The immediate action $<a, \infty_{l,w}>$ is simply mapped to $<a, \infty_{0,w}>$, thus losing the information about its priority. Similarly, the passive action $<a, *_w^l>$ is turned into the action $<a, *_w^0>$. Semantically, the result is a calculus where the execution probabilities of the actions are calculated on the basis of the associated weights. The underlying model for this calculus is a mixture of the generative and reactive probabilistic transition systems. A variant of this calculus, where the probabilities are associated with the operators rather than attached to the actions as weights, has been introduced in [6] and used in [3] to define probabilistic noninterference. We denote with $pr(P)$ the probabilistic version of process P obtained in this way.

2.3 Bisimulation Semantics

We now recall from [5] an extension of Markovian bisimilarity, whose basic idea is to compare the exit rates of the process terms by taking into account the various kinds of actions (exponentially timed, immediate, and passive). This is accomplished by parameterizing the notion of exit rate with respect to a number in $\mathbb{Z}$ representing the priority level of the action, which is 0 if the action is exponentially timed, l if the action rate is $\infty_{l,w}$, $-l' - 1$ if the action rate is $*_w^{l'}$.

Definition 2. *Let $P \in \mathcal{P}$, $a \in Name$, $l \in \mathbb{Z}$, and $C \subseteq \mathcal{P}$. The exit rate of P when executing actions with name a and priority level l that lead to C is defined through the following non-negative real function:*

$$rate(P, a, l, C) = \begin{cases} \sum\{\!| \lambda \mid \exists P' \in C. P \xrightarrow{a,\lambda} P' |\!\} & \textit{if } l = 0 \\ \sum\{\!| w \mid \exists P' \in C. P \xrightarrow{a,\infty_{l,w}} P' |\!\} & \textit{if } l > 0 \\ \sum\{\!| w \mid \exists P' \in C. P \xrightarrow{a,*_w^{-l-1}} P' |\!\} & \textit{if } l < 0 \end{cases}$$

where each sum is taken to be zero whenever its multiset is empty.

Extended Markovian bisimilarity compares the process term exit rates for all possible action names and priority levels, except for those actions that will always be pre-empted by higher priority actions of the form $<\tau, \infty_{l,w}>$. We denote by $pri^{\tau}_{\infty}(P)$ the priority level of the highest priority internal immediate action enabled by P, and we set $pri^{\tau}_{\infty}(P) = 0$ if P does not enable any internal immediate action. Moreover, given $l \in \mathbb{Z}$, we use $no\text{-}pre(l, P)$ to denote that no action of level l can be pre-empted in P. Formally, this is the case whenever $l \geq pri^{\tau}_{\infty}(P)$ or $-l - 1 \geq pri^{\tau}_{\infty}(P)$.

Definition 3. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P} \times \mathcal{P}$ is an extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$, equivalence classes $C \in \mathcal{P}/\mathcal{B}$, and priority levels $l \in \mathbb{Z}$ such that $no\text{-}pre(l, P_1)$ and $no\text{-}pre(l, P_2)$:*

$$rate(P_1, a, l, C) = rate(P_2, a, l, C)$$

Extended Markovian bisimilarity, denoted by $\sim_{\mathrm{EMB}}$, is the union of all the extended Markovian bisimulations.

We relax the definition of exit rate by means of a notion of reachability that involves the internal actions with zero duration $<\tau, \infty_{l,w}>$, which are unobservable. The idea is that, if a given class of process terms is not reached directly after executing an action with a certain name and level, then we have to explore the possibility of reaching that class indirectly via a finite-length unobservable path starting from the term reached after executing the considered action.

Definition 4. *Let $P \in \mathcal{P}$ and $l \in \mathbb{N}_{>0}$. We say that P is l-unobservable iff $pri^{\tau}_{\infty}(P) = l$ and P does not enable any immediate non-τ-action with priority level $l' \geq l$, nor any passive action with priority constraint $l' \geq l$.*

Definition 5. *Let $n \in \mathbb{N}_{>0}$ and $P_1, P_2, \ldots, P_{n+1} \in \mathcal{P}$. A path π of length n:*

$$P_1 \xrightarrow{\tau,\infty_{l_1,w_1}} P_2 \xrightarrow{\tau,\infty_{l_2,w_2}} \ldots \xrightarrow{\tau,\infty_{l_n,w_n}} P_{n+1}$$

is unobservable iff for all $i = 1, \ldots, n$ process term P_i is l_i-unobservable. In that case, the probability of executing path π is given by:

$$prob(\pi) \ = \ \prod_{i=1}^{n} \frac{w_i}{rate(P_i, \tau, l_i, \mathcal{P})}$$

Definition 6. *Let $P \in \mathcal{P}$, $a \in Name$, $l \in \mathbb{Z}$, and $C \subseteq \mathcal{P}$. The weak exit rate at which P executes actions of name a and level l that lead to C is defined through the following non-negative real function:*

$$rate_{\mathrm{w}}(P, a, l, C) = \sum_{P' \in C_{\mathrm{w}}} rate(P, a, l, \{P'\}) \cdot prob_{\mathrm{w}}(P', C)$$

where C_{w} is the weak backward closure of C:

$$C_{\mathrm{w}} = C \cup \{Q \in \mathcal{P} - C \mid Q \textit{ can reach } C \textit{ via unobservable paths}\}$$

and $prob_{\mathrm{w}}$ is a $\mathbb{R}_{]0,1]}$-valued function representing the sum of the probabilities of all the unobservable paths from a term in C_{w} to C:

$$prob_{\mathrm{w}}(P', C) = \begin{cases} 1 & \textit{if } P' \in C \\ \sum\{\!| \, prob(\pi) \mid \pi \textit{ unobs. path from } P' \textit{ to } C \, |\!\} & \textit{if } P' \in C_{\mathrm{w}} - C \end{cases}$$

When comparing process term weak exit rates, besides taking pre-emption into account, we also have to skip the comparison for classes that contain certain unobservable terms. More precisely, we distinguish among observable, initially unobservable, and fully unobservable terms. An observable term is a term that enables a non-τ-action that cannot be pre-empted by any enabled immediate τ-action. An initially unobservable term is a term in which all the enabled non-τ-actions are pre-empted by some enabled immediate τ-action, but at least one of the paths starting at this term with one of the higher priority enabled immediate τ-actions reaches an observable term. A fully unobservable term is a term in which all the enabled non-τ-actions are pre-empted by some enabled immediate τ-action, and all the paths starting at this term with one of the higher priority enabled immediate τ-actions are unobservable (note that $\underline{0}$ is fully unobservable).

The weak exit rate comparison with respect to observable and fully unobservable classes must obviously be performed. In order to maximize the abstraction power in the presence of quantitative information attached to immediate τ-actions, the comparison should be conducted with respect to the whole set $\mathcal{P}_{\mathrm{fu}}$ of fully unobservable process terms of $\mathcal{P}$. By contrast, the comparison with respect to initially unobservable classes should be skipped as each of them can reach an observable class.

Definition 7. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P} \times \mathcal{P}$ is a weak extended Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$ and levels $l \in \mathbb{Z}$ such that no-pre(l, P_1) and no-pre(l, P_2):*

$$rate_{\mathrm{w}}(P_1, a, l, C) = rate_{\mathrm{w}}(P_2, a, l, C) \quad \textit{for all observable } C \in \mathcal{P}/\mathcal{B}$$
$$rate_{\mathrm{w}}(P_1, a, l, \mathcal{P}_{\mathrm{fu}}) = rate_{\mathrm{w}}(P_2, a, l, \mathcal{P}_{\mathrm{fu}})$$

Weak extended Markovian bisimilarity, denoted by $\approx_{\mathrm{EMB}}$, is the union of all the weak extended Markovian bisimulations.

As examples of weakly extended Markovian bisimilar process terms we mention:

$$P_1 \equiv <a, \lambda>.<\tau, \infty_{l,w}>.<b, \mu>.\underline{0}$$
$$P_2 \equiv <a, \lambda>.<b, \mu>.\underline{0}$$

and also:

$$P_3 \equiv <a, \lambda>.(<\tau, \infty_{l,w_1}>.<b, \mu>.\underline{0} + <\tau, \infty_{l,w_2}>.<c, \gamma>.\underline{0})$$
$$P_4 \equiv <a, \lambda \cdot \tfrac{w_1}{w_1 + w_2}>.<b, \mu>.\underline{0} + <a, \lambda \cdot \tfrac{w_2}{w_1 + w_2}>.<c, \gamma>.\underline{0}$$

which is related to vanishing state elimination. We also point out that $\approx_{\mathrm{EMB}}$ can abstract not only from intermediate immediate τ-actions, but also from intermediate unobservable self-loops, consistently with the fact that the probability to escape from them is 1. Moreover, $\approx_{\mathrm{EMB}}$ cannot abstract from initial immediate

τ-actions, otherwise compositionality with respect to the alternative composition operator would be broken. The careful classification of states on the basis of their functional and performance observability is a key ingredient thanks to which congruence and axiomatization can be achieved for $\approx_{\mathrm{EMB}}$. In particular, compositionality with respect to parallel composition is preserved by restricting to a well-prioritized subset of non-divergent process terms of $\mathcal{P}$ [5].

3 Noninterference Properties

In a typical two-level access system we distinguish between the high security access level and the low security access level. Information flowing from low level to high level is authorized, while the inverse flow is not. The noninterference analysis aims at revealing direct and indirect information flows, called covert channels, in the unauthorized path. In a process algebraic setting this classification of security levels is realized by assuming that the set *Name* of action names includes a set $Name_L \subseteq Name$ of low-level names and a set $Name_H \subseteq Name$ of high-level names, such that $Name_L \cap Name_H = \emptyset$. These sets describe the interactions between the system and the low-level and high-level parts of the environment, respectively. We can also have that $Name_L \cup Name_H = Name$. If this is not the case, we assume that the remaining action names can be disregarded as they are not related to interactions with the users at different access clearances. For instance, they could represent synchronizing activities performed by different system components that cooperate to handle the service requests. Hence, they will be hidden by turning them into unobservable actions.

In the following, we describe two noninterference properties relying on this classification and on the bisimulation semantics, named strong noninterference property and strong local noninterference property.

3.1 Bisimulation-Based Strong Noninterference

The requirement at the base of the lack of any interference from high level to low level can be easily expressed by a property called Strong Nondeterministic Noninterference, which informally says that a system is secure if its observable low-level behavior is the same in the presence and in the absence of high-level interactions. The stochastic version of this property is called Bisimulation-based Strong Stochastic Noninterference (*BSSNI*) and is defined as follows, where $P/Name_H$ expresses the system view in the presence of high-level actions, which are hidden because they cannot be explicitly seen by a low-level observer, while $P \backslash Name_H$ represents the system view whenever the high-level actions are absent, i.e. they are prevented from execution by means of the restriction operator.

Definition 8. ***(BSSNI)*** *$P \in \mathcal{P}_{\mathrm{pc}}$ is secure iff $P/Name_H \approx_{\mathrm{EMB}} P \backslash Name_H$.*

For brevity, we assume that P is performance closed. For the formal treatment of open systems with respect to high-level inputs, the interested reader is referred to [3]. The nondeterministic version of *BSSNI*, termed *BSNNI* [9], is obtained

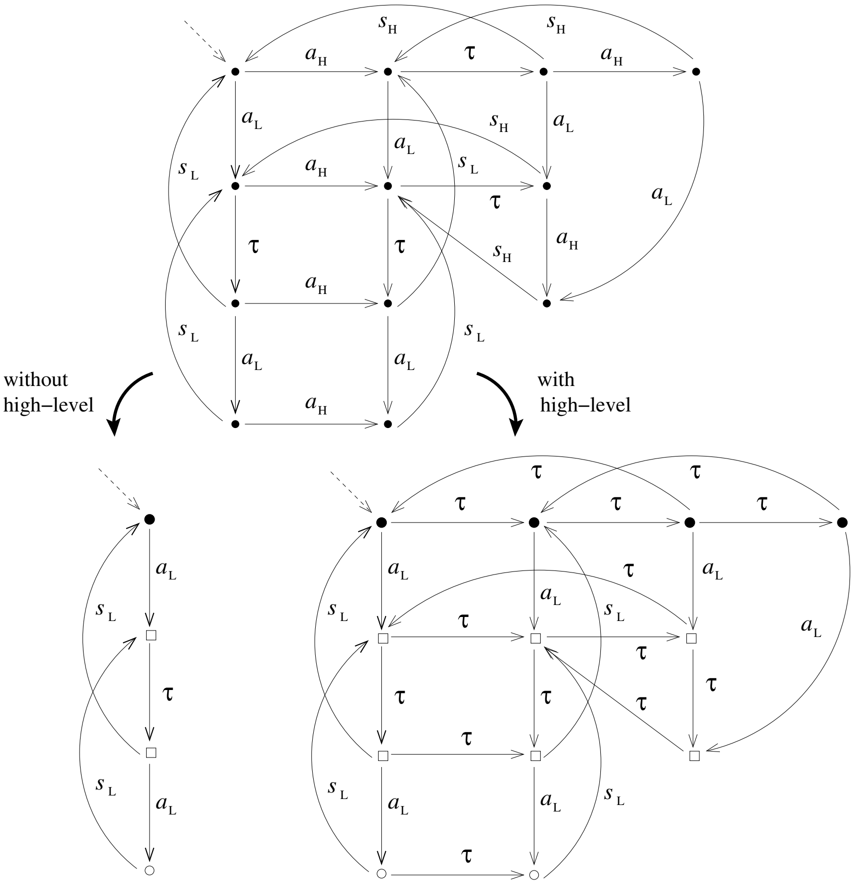


Fig. 1. Nondeterministic version of the client-server system and its low-level views

in this framework by replacing $\approx_{\mathrm{EMB}}$ with the Milner's weak bisimilarity [15], denoted by $\approx_{\mathrm{B}}$.

Definition 9. ***(BSNNI)*** *$nd(P)$ is secure iff $nd(P)/Name_H \approx_{\mathrm{B}} nd(P)\backslash Name_H$.*

Similarly, the probabilistic version of *BSSNI*, termed *BSPNI* [3], is defined in the same framework by replacing $\approx_{\mathrm{EMB}}$ with the weak probabilistic bisimilarity of [6], denoted by $\approx_{\mathrm{PB}}$.

Definition 10. ***(BSPNI)*** *$pr(P)$ is secure iff $pr(P)/Name_H \approx_{\mathrm{PB}} pr(P)\backslash Name_H$.*

Example 2. We now discuss an incremental analysis of the client-server system introduced in Ex. 1. This is done by considering the three fine-grain notions of noninterference in order to show which kind of information flow can be revealed and what strategies should be implemented to circumvent the related covert channels. We start by classifying the activities performed by the client-server system into low level and high level. The low-level client is represented by process A_L, thus the action name modeling the arrival process, a_L, and the action name modeling the delivered service, s_L, are the unique action names in $Name_L$. Similarly, given that A_H represents the high-level client, we assume that $Name_H = \{a_H, s_H\}$. The communications between the buffers and the server represent internal activities among components of the service center. Thus, they should not be observed by any client. Hence, the noninterference check is applied to system $CS/\{d_L, d_H\}$ even if, for brevity, we will continue to refer to CS.

First, consider the property *BSNNI* and the system $nd(CS)$. The underlying semantic model is represented by the labeled transition system $[\![nd(CS)]\!]$ in the upper part of Fig. 1, which also includes in its lower part the two labeled transition systems to compare according to the noninterference property. The weak bisimulation relating these two sub-systems is illustrated by graphically representing in the same way the states that belong to the same class. Hence, it is easy to see that the noninterference check is satisfied and the functional behavior of the system is secure. Intuitively, the availability to serve the low-level requests is never compromised independently of the behavior of the high-level client.

Second, consider the property *BSPNI* and the system $pr(CS)$. The semantics of this system is obtained from $[\![\mathrm{nd}(CS)]\!]$ by enriching every edge in Fig. 1 with a weight. The intuition is that with this extension we add fine-grain information which allow nondeterministic choices to be solved according to probability distributions modeling the quantitative behavior of system activities. In spite of this enriched, detailed description, the noninterference check based on *BSPNI* is satisfied and the system turns out to be secure. As a sketch of the proof of this result, consider the semantics of the sub-system $pr(CS)\backslash Name_H$.

From the unique state of class $\bullet$ it is possible to reach, through the action a_L, class $\Box$ with probability 1. Then, after an internal move within the same class, the unique probabilistic choice of this sub-system governs the choice among the action s_L leading to class $\bullet$ and the action a_L leading to class $\circ$. Finally, from class $\circ$ the unique enabled transition, which is labeled with s_L, leads to class $\Box$ with probability 1. Now, consider the semantics of the sub-system $pr(CS)/Name_H$. From states of class $\bullet$ it is possible to pass, through the weak transition $\tau^* a_L$, to class $\Box$ with probability 1. With the same probability, we then reach, by following unobservable paths, one of two states of class $\Box$ enabling the same probabilistic choice between s_L and a_L that characterizes the former sub-system. Finally, the two states of class $\circ$ lead with probability 1 to states of class $\Box$ through the weak transition $\tau^* s_L$. Summing up, adding probabilistic information to the system does not increase the discriminating power of the low-level observer, because any high-level activity is not able to alter the unique low-level probabilistic choice.

Third, consider the property *BSSNI* and the system CS. The intuition is that with this extension the clients observe the passage of time and the priority-based behavior of the server. The introduction of this fine-grain information causes an interfering information flow from high level to low level, which is revealed by the violation of the *BSSNI* condition. In the following, we show the nature of these covert channels and how the diagnostic information provided by $\approx_{\mathrm{EMB}}$ has been exploited to avoid them.

On the one hand, the high-level process A_H is immediately revealed by the low-level client. Indeed, $A_H / Name_H$ describes a working process that, according to the race policy, competes with the other durational processes, while $A_H \backslash Name_H$ does not (first interference). On the other hand, from the viewpoint of the low-level client, the time spent by the server to deliver the high-level services describes an observable busy waiting phase (second interference). From a performance-oriented perspective, these interferences affect the low-level throughput of the client-server system, in terms of number of s_L actions executed per unit of time, which is higher in the absence of high-level interactions. The removal of the two interferences requires strong control mechanisms that, as expected, aim at degrading the performance of the system in order to make the behavior of the high-level client transparent to the low-level client.

As far as the first interference is concerned, it is necessary to confine the behavior of the high-level client in order to hide its impact on the timing of the system activities. This can be done by employing a sort of high-level box which, somehow, limits and controls the activities performed by the high-level client. Formally, we replace process A_H with the following one:

$$\begin{aligned} H\_box \stackrel{\Delta}{=}\ & {<}\tau, \infty_{l,w}{>}. \\ & ({<}a_H, \infty_{h+1,w}{>}.{<}a_H, \lambda_H{>}.H\_box + {<}\tau, \infty_{h,w}{>}.{<}\tau, \lambda_H{>}.H\_box) \end{aligned}$$

The action ${<}\tau, \infty_{l,w}{>}$ represents the activation of the box and is technically needed because it allows $\approx_{\mathrm{EMB}}$ to abstract from the subsequent intermediate immediate τ-actions. The branch guarded by the action ${<}\tau, \infty_{h,w}{>}$ is enabled if and only if the high-level client is absent. Its role is to simulate the presence of such a client in a way that makes the high-level behavior invisible to the low-level client. Intuitively, process *H_box* is a black box acting as an interface between the system and the high-level client, who can interact with such a box by pushing a button whenever he decides to communicate with the system.

As far as the second interference is concerned, making the server transparent to the low-level client is achieved by following an approach borrowed by round-robin scheduling strategies. In practice, the intuition is similar to that underlying the definition of process *H_box*. The service activities are divided into temporal slots, each one dedicated to a class of clients in a round-robin fashion. Independently of the presence of a pending request from a client of the currently managed class, the temporal slot is spent. In this way a low-level client cannot deduce whether the high-level slot has been actively exploited by the high-level client. Formally, we replace process S with the following one:

$$S_L \stackrel{\Delta}{=} {<}d_L, \infty_{f(L),w}{>}.{<}s_L, \mu_L{>}.S_H + {<}\tau, \mu_L{>}.S_H$$

$$S_H \stackrel{\Delta}{=} {<}d_H, \infty_{f(H),w}{>}.{<}s_H, \mu_H{>}.S_L + {<}\tau, \mu_H{>}.S_L$$

This is not enough to hide completely the interference of the high-level client. Indeed, whenever such a client is blocked because the related buffer is not available to accept further requests, then process *H_box* does not compete for the resource time. This observable behavior would reveal to the low-level client that the high-level buffer is full. The covert channel can be avoided by changing the high-level buffer as follows:

$$B_H \overset{\Delta}{=} <a_H, *_w^0>.B'_H \qquad B'_H \overset{\Delta}{=} <d_H, *_w^{f(H)}>.B_H + <\tau, \lambda_H>.B'_H$$

■

3.2 Bisimulation-Based Strong Local Noninterference

The introduction of fine-grain information about time makes the satisfaction of noninterference properties a very hard task, which can be accomplished through invasive strategies aiming at controlling the temporal behavior of the system. This claim is strengthened by the fact that we employed one of the least restrictive noninterference properties in the literature. As an example, we ignored the interference caused by a high-level user that blocks the high-level output modeling the service delivery, because this problem can be easily avoided by making such a communication asynchronous [1]. This and more subtle problems can be revealed by security properties that have been defined in the literature with the aim of capturing more information flows with respect to the noninterference property surveyed above. For instance, the strongest property of [9] is the Strong Bisimulation Nondeducibility on Compositions, which corresponds to the Strong Local Noninterference property that has been separately defined in [16]. In our framework, we consider such a property under the name Bisimulation-based Strong Stochastic/Probabilistic/Nondeterministic Local Noninterference, respectively, depending on the nature of the fine-grain information we take into consideration. The underlying intuition states that the absence of any interference is ensured when the low-level user cannot distinguish which, if any, high-level event has occurred at some point in the past.

Definition 11. ***(BSSLNI)*** *P is secure if and only if* $\forall P' \in Der(P)$ *and* $\forall P''$ *such that* $P' \xrightarrow{a,\tilde{\lambda}} P''$, *with* $a \in Name_H$, *then* $P' \backslash Name_H \approx_{\rm EMB} P'' \backslash Name_H$.

In essence, this property states that the low-level view of the system is not affected by the execution of a high-level action. The nondeterministic version, *BSNLNI*, and the probabilistic version, *BSPLNI*, can be derived as before. However, the introduction of the temporal aspect changes the nature of the relations among these properties. The reason stems from the fine-grain information associated with the high-level actions. The intuition is that the strong local noninterference property analyzes the low-level view of the system before and after the execution of a high-level action without taking into account neither the time spent by its execution (if it is durational) nor its priority (if it is immediate). In practice, in the stochastic setting the original notions of strong noninterference and strong local noninterference cannot be compared.

Theorem 1. ***(Inclusion relations)***
(*i*) *BSNLNI* $\subset$ *BSNNI*
(*ii*) *BSPLNI* $\subset$ *BSPNI*
(*iii*) *BSSLNI* $\not\subset$ *BSSNI*

Proof. *(i) and (ii) are standard results shown in [9] and [3], respectively. To show (iii), consider the process term $P_7 \equiv <h, \lambda>.<l, \mu>.\underline{0} + <l, \mu>.\underline{0}$, which satisfies BSSLNI, because its low-level view is $<l, \mu>.\underline{0}$ before and after the execution of the high-level action. However, P_7 is not BSSNI secure, because $<l, \mu>.\underline{0}$ and $P_8 \equiv <\tau, \lambda>.<l, \mu>.\underline{0} + <l, \mu>.\underline{0}$ are not weakly extended Markovian bisimilar. The motivation is given by the race policy between the two exponentially timed actions. Similarly, consider the process term $P_9 \equiv <h, \infty_{2,w}>.P_{10} + P_{10}$, where $P_{10} \equiv <l, \infty_{1,w}>.\underline{0} + <l', \infty_{2,w}>.\underline{0}$. P_9 is clearly BSSLNI secure, but not BSSNI secure: in $P_9/Name_H$ the low-level action with name l is initially pre-empted, thus altering the probability of executing the two low-level actions.*

Example 3. As far as the client-server system is concerned, $nd(CS)$ and $pr(CS)$ satisfy *BSNLNI* and *BSPLNI*, respectively, provided that the delivery of the high-level service is made asynchronous. Such a relaxation is not enough to make process CS a *BSSLNI*-secure system. For this purpose, all the exponentially timed actions executed by process H_box must be made invisible in order to keep them out of the control of the high-level client. Then, process H_box should be further complicated in such a way that the (priority, probabilistic, and temporal) low-level view of the system must be the same before and after the execution of action $<a_H, \infty_{h,w}>$.

The stochastic noninterference check pinpoints the covert channels that affect the quantitative behavior of the system and, therefore, the delivered quality of service. The low-level throughput of the original client-server system offers different results depending on the presence of high-level interactions. This different quantitative behavior that is exhibited by the two sub-systems is formally captured by the $\approx_{\rm EMB}$-based check as a diagnostic information revealing the interference. Estimating the metric above in the presence and in the absence of the high-level client is a way to measure the covert channel bandwidth. The same performance metric reveals the quality of service degradation that is paid by making the system completely secure through the high-level black box and the revised versions of the service center components. Hence, a balanced trade-off between security and performance strictly depends on the value of this metric. From the analysis standpoint, since the semantic model underlying the system CS is a CTMC, such a metric can be easily evaluated through reward-based steady-state numerical analysis in order to obtain an estimation of the information leakage on the long run [20]. ■

In conclusion, strong notions of noninterference are hard to accomplish when modeling real-world systems that turn out to be much more complex than the client-server example. Adding fine-grain information such as time opens new scenarios where covert channels cannot be completely eliminated without severely limiting the system behavior. On the basis of the discussed strategies that are needed to pass the *BSSNI*-based check, it is easy to observe that the minimization of the covert channel bandwidth corresponds to a reduction of the quality of service. Hence, an approach towards the mitigation of the strict, unrealistic constraints imposed by the noninterference properties should be based on tolerance thresholds, which are expressed in terms of negligible difference with respect to a family of performance metrics related to quality of service. A similar approach is used in [2], where the impact of the high-level strategies (and of the related countermeasures adopted by the system) is estimated from the performance standpoint.

4 Conclusions

The definition and analysis of stochastic noninterference is not only yet another extension of classical nondeterministic noninterference. In fact, it allowed us to highlight some interesting relations between the noninterference approach to information flow analysis and the quality of the service delivered by real-world systems. In a framework where time is important, the objective is not to guarantee the complete absence of any kind of interference, which is a task that is either impossible to achieve or impractical because of the strong functional limitations that should be imposed. Instead, the attention should pass to the analysis of the performance measures of interest that are affected by the high-level interferences. The goal is to estimate whether such interferences are negligible (e.g. because they are revealed if and only if the system execution is observed for a long time) and, therefore, cause a tolerable amount of information leakage in terms of sensitive data that are revealed on the long run.

From a practical standpoint, we intend to use the notion of stochastic noninterference to support a methodology for the analysis of a balanced trade-off between the security degree of the system and the impact of different securing strategies on the system quality of service. The intuition is that the stronger the noninterference validation is, the more complicated and invasive the related securing strategy should be. From a theoretical standpoint, it would be useful to investigate properties stronger than *BSSNI* which do not imply the explicit analysis of every possible high-level user, as instead required, e.g., by the Nondeducibility on Compositions property.

Acknowledgement. The authors thank the anonymous referees for their valuable comments. This work has been funded by MIUR-PRIN project *PaCo – Performability-Aware Computing: Logics, Models, and Languages.*

References

1. Aldini, A.: Classification of security properties in a Linda-like process algebra. J. of Science of Computer Programming 63, 16–38 (2006)
2. Aldini, A., Bernardo, M.: A formal approach to the integrated analysis of security and QoS. J. of Reliability Engineering & System Safety 92, 1503–1520 (2007)
3. Aldini, A., Bravetti, M., Gorrieri, R.: A process-algebraic approach for the analysis of probabilistic noninterference. J. of Computer Security 12, 191–245 (2004)
4. Aldini, A., Bravetti, M., Di Pierro, A., Gorrieri, R., Hankin, C., Wiklicky, H.: Two formal approaches for approximating noninterference properties. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2001. LNCS, vol. 2946, pp. 1–43. Springer, Heidelberg (2004)

5. Bernardo, M., Aldini, A.: Weak Markovian bisimilarity: abstracting from prioritized/weighted internal immediate actions. In: 10th Italian Conf. on Theoretical Computer Science, pp. 39–56. World Scientific, Singapore (2007)
6. Bravetti, M., Aldini, A.: Discrete time generative-reactive probabilistic processes with different advancing speeds. Theoretical Comp. Science 290, 355–406 (2003)
7. Di Pierro, A., Wiklicky, H.: Quantifying timing leaks and cost optimisation. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 81–96. Springer, Heidelberg (2008)
8. Focardi, R., Martinelli, F., Gorrieri, R.: Information flow analysis in a discrete-time process algebra. In: IEEE Computer Security Foundations Workshop, pp. 170–184 (2000)
9. Focardi, R., Gorrieri, R.: A classification of security properties. J. of Computer Security 3, 5–33 (1995)
10. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative and stratified models of probabilistic processes. Information and Comput. 121, 59–80 (1995)
11. Goguen, J.A., Meseguer, J.: Security policy and security models. In: IEEE Symp. on Security and Privacy, pp. 11–20 (1982)
12. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: A classification of time and/or probability dependent security properties. In: 3rd Int. Workshop on Quantitative Aspects of Programming Languages. ENTCS, vol. 153, pp. 177–193 (2005)
13. Mantel, H., Sudbrock, H.: Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework. In: IEEE Computer Security Foundations Symposium, pp. 326–340 (2007)
14. McLean, J.: Security models and information flow. In: IEEE Symp. on Research in Security and Privacy, pp. 180–187 (1990)
15. Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs (1989)
16. Roscoe, A.W., Reed, G.M., Forster, R.: The successes and failures of behavioural models. Millenial Perspectives in Computer Science (2000)
17. Ryan, P.Y.A., Schneider, S.A.: Process algebra and non-interference. J. of Computer Security 9, 75–103 (2001)
18. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: IEEE Computer Security Foundations Workshop, pp. 200–214 (2000)
19. Smith, G.: Probabilistic noninterference through weak probabilistic bisimulation. In: IEEE Computer Security Foundations Workshop, pp. 3–13 (2003)
20. Stewart, W.J.: Introduction to the numerical solution of Markov chains. Princeton University Press, Princeton (1994)
21. Volpano, D., Smith, G.: Probabilistic noninterference in a concurrent language. In: IEEE Computer Security Foundations Workshop, pp. 34–43 (1998)
22. Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symp. on Research in Security and Privacy, pp. 144–161 (1990)

Validating Security Protocols under the General Attacker

Wihem Arsac[1], Giampaolo Bella[2], Xavier Chantry[1], and Luca Compagna[1]

[1] SAP Research Labs, 805 Avenue du Dr Maurice Donat, 06250 Mougins, France
{wihem.arsac,xavier.chantry,luca.compagna}@sap.com
[2] Dipartimento di Matematica e Informatica,
Università di Catania, Viale A.Doria 6, 95125 Catania, Italy
giamp@dmi.unict.it

Abstract. Security protocols have been analysed using a variety of tools and focusing on a variety of properties. Most findings assume the ever so popular Dolev-Yao threat model. A more recent threat model called the Rational Attacker [1] sees each protocol participant decide whether or not to conform to the protocol upon their own cost/benefit analysis. Each participant neither colludes nor shares knowledge with anyone, a feature that rules out the applicability of existing equivalence results in the Dolev-Yao model. Aiming at mechanical validation, we abstract away the actual cost/benefit analysis and obtain the General Attacker threat model, which sees each principal blindly act as a Dolev-Yao attacker.

The analysis of security protocols under the General Attacker threat model brings forward yet more insights: *retaliation attacks* and *anticipation attacks* are our main findings, while the tool support can scale up to the new analysis at a negligible price. The general threat model for security protocols based on set-rewriting that was adopted in AVISPA [2] is leveraged so as to express the General Attacker. The state-of-the-art model checker SATMC [3] is then used to automatically validate a protocol under the new threats, so that retaliation and anticipation attacks can automatically be found.

1 Introduction

The analysis of security protocols stands among the most attractive niches of research in computer science, as it has attracted efforts from many communities. It is difficult to even provide a satisfactory list of citations, which would have to at least include process calculi [4,5], strand spaces [6,7], the inductive method [8,9] and advanced model checking techniques [2,10].

Any meaningful statement about the security of a system requires a clear specification of the threats that the system is supposed to withstand. Such a specification is usually referred to as *threat model*. Statements that hold under a threat model may no longer hold under other models. For example, if the threat model only accounts for attackers that are outsiders, then Lowe's famous attack on the Needham-Schroeder Public-Key (NSPK) protocol [11] cannot succeed,

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 34–51, 2009.

and the protocol may be claimed secure. But the protocol is notoriously insecure under a model that allows the attacker as a registered principal.

The standard threat model for symbolic protocol analysis is the Dolev-Yao model (DY in brief), which sees a powerful attacker control the whole network traffic. The usual justification is that a protocol secure under DY certainly is secure under a less powerful, perhaps more realistic attacker. By contrast, a large group of researchers consider DY insufficient because a DY attacker cannot do cryptanalysis, and their probabilistic reasoning initiated with a foundational paper [12]. This sparked off a research thread that has somewhat evolved in parallel with the DY research line, although some efforts exist in the attempt to conjugate them [13,14,15]. The present paper is not concerned with probabilistic protocol analysis. Our main argument is that security protocols may still hide important subtleties even after they are proved correct under DY. These subtleties can be discovered by symbolic protocol analysis under a new threat model that adheres to the present real world more strictly than DY does.

The new model we develop here is the *General Attacker* (GA in brief), which features each protocol participant as a DY attacker who does not collude or share knowledge with anyone else. In GA, it is meaningful to continue the analysis of a protocol *after* an attack is mounted, or to anticipate the analysis by looking for extra flaws *before* that attack, something that has never been seen in the relevant literature. This can assess whether additional attacks can be mounted either by the same attacker or by different attackers. Even novel scenarios whereby principals attack each other become possible. A significant scenario is that of *retaliation* [16], where an attack is followed by a counterattack. This paper recasts that scenario into the new GA threat model. Also, a completely new scenario is defined, that of *anticipation*, where an attack is anticipated, before its termination, by another attack by some other principal.

As its main contribution, our research tailors an existing formalism suited for model checking to accommodate the GA threat model. This makes it possible to analyse protocol subtleties that go beyond standard security properties such as confidentiality and authentication. We begin by extending an existing set-rewriting formalisation of the classical DY model to capture the GA model. Then, we leverage an established model checking tool for security protocols to tackle the validation problems arisen from the new threat model. Finally, we run the tool over the NSPK protocol and its variants to investigate retaliation and anticipation attacks.

The structure of this manuscript is simple. The GA threat model is introduced in Section 2 and then our formalisation and analysis results are described in Sections 3 and 4. Some conclusions terminate the presentation.

2 The General Attacker

DY can be considered the standard threat model to study security protocols [17]. The DY attacker controls the entire network although he cannot perform cryptanalysis. Some historical context is useful here. The model was defined in the late 1970s when remote computer communications were still confined to military/espionage contexts. It was then natural to imagine that the entire world would collude joining their forces against a security protocol session between two secret agents of the same country. The DY model has remarkably favoured the discovery of significant protocol, but the prototype attacker is significantly changed today. To become an attacker has never been so easy as in the present technological setting because hardware is inexpensive, while security skill is at hand—malicious exploits are even freely downloadable from the web.

A seminal threat model called BUG [16], which dates back to 2002 [18], must be recalled here. The name is a permuted acronym for the "Good", the "Bad" and the "Ugly". This model attempts stricter adherence than DY's to the changed reality by partitioning the participants in a security protocol into three groups. The Good principals would follow the protocol, the Bad would in addition try to subvert it, and the Ugly would be ready to either behaviour. This seems the first account in the literature of formal protocol verification on the chance that attackers may attempt to attack each other without sharing knowledge. More recently, Bella observed [1] that the partition of the principals had to be dynamically updated very often, in principle at each event of a principal's sending or receiving a message, depending on whether the principal respected the protocol or not. Thus, BUG appeared overly detailed, and he simplified it as the Rational Attacker threat model: each principal may at any time make cost/benefit decisions to either behave according to the protocol or not [1]. After BUG's inception [18], homologous forms of rational attackers were specifically carved out in the area of game theory [19,20] and therefore are, as such, not directly related to our symbolic analysis.

Analyzing a protocol under the Rational Attacker requires specifying each principal's cost and benefit functions, but this still seems out of reach, especially for classical model checking. By abstracting away the actual cost/benefit analysis, we derive the following simplified model:

> The General Attacker (GA) threat model: *each principal is a Dolev-Yao attacker.*

The change of perspective in GA with respect to DY is clear: principals do not collude for a common aim but, rather, each of them acts for his own personal sake. Although there is no notion of collusion constraining this model, the human protocol analyser can define some for their particular investigations. The GA model has each principal endowed with the entire potential of a DY attacker. So, each principal may at any stage send any of the messages he can form to anyone. Of course, such messages include both the legal ones, conforming to the protocol in use, and the illegal, forged ones.

As we shall see, analysing the protocols under the GA threat model yields unknown scenarios featuring retaliation or anticipation attacks. This paves the way for a future analysis under the Rational Attacker. For example, if an attack can be retaliated under GA, such a scenario will not occur under the Rational Attacker because the cost of attacking clearly overdoes its benefit, and hence the

1. $A \rightarrow B : \{Na, A\}_{Kb}$
2. $B \rightarrow A : \{Na, Nb\}_{Ka}$
3. $A \rightarrow B : \{Nb\}_{Kb}$

$4a.$ $A \rightarrow B : \{$"Transfer X1€ from A's account to $Y1$'s"$\}_{\langle Na, Nb \rangle}$

$4b.$ $B \rightarrow A : \{$"Transfer X2€ from B's account to $Y2$'s"$\}_{\langle Na, Nb \rangle}$

Fig. 1. NSPK++: the NSPK protocol terminated with the completion steps

attacker will not attack in the first place. This argument is after all not striking: even a proper evaluation of the "realism" of classical attacks found under DY would have required a proper cost/benefit analysis.

Our research suggests that if in the real world an attacker can mount an attack that can be retaliated, then he may rationally opt for not attacking in the first place. In consequence, even a deployed protocol suffering an attack that can be retaliated may perhaps be kept in place.

The BUG threat model was demonstrated over the public-key Needham-Schroeder protocol [16]. In the sequel of this section, we recast that analysis under the GA model. The protocol version studied here, which we address as NSPK++, is its original design terminated with the completion steps for reciprocal, authenticated money transfers (Figure 1). It can be seen that principal A issues a fresh nonce Na in message 1, which she sees back in message 2. Because message 1 is encrypted under B's public key, the nonce was fresh and cryptanalysis cannot be broken by assumption, A learns that B acted at the other end of the network. Messages 2 and 3 give B the analogous information about A by means of nonce Nb. This protocol version is subject to Lowe's attack [11], as described in Figure 2. It can be seen how the attacker C masquerades as A with B to carry out an illegal money transfer at B (which intuitively is a bank) from A's to C's account. It is known that the problems originated with the confidentiality attack upon the nonce Nb. Another observation is that there is a second nonce whose confidentiality is violated—by B, not by C—in this scenario: it is Na. Although it is invented by A to be only shared with C, also B learns it. This does not seem to be an issue in the DY model, where all principals except C followed the protocol like soldiers follow orders. What one of them could do with a piece of information not meant for him therefore became uninteresting. To what extent this is appropriate to the current real world, where there often are various attackers with targets of their own, is at least questionable. Strictly speaking, B's learning of Na is a new attack because it violates the confidentiality policy upon the nonces, which are later used to form a session key. It can be easily captured in the GA threat model, where more than one principal may act illegally at the same time for their own sake.

We are facing a new perspective of analysis. Principal B did not have to act to learn a nonce not meant for him, therefore this is named an *indeliberate attack* [18]. To use a metaphor, B does not know which lock the key Na can open. This is not an issue in the GA threat model, where each principal just sends out anything he can send to anyone. Nevertheless, there are at least four methods to help B practically evaluate the potential of Na [21].

$$\begin{array}{llll}
1. & A \rightarrow C : \{Na, A\}_{Kc} & & \\
 & & 1'. & C(A) \rightarrow B : \{Na, A\}_{Kb} \\
 & & 2'. & \quad B \rightarrow A : \{Na, Nb\}_{Ka} \\
2. & C \rightarrow A : \{Na, Nb\}_{Ka} & & \\
3. & A \rightarrow C : \{Nb\}_{Kc} & & \\
 & & 3'. & C(A) \rightarrow B : \{Nb\}_{Kb}
\end{array}$$

$$4a'. \quad C(A) \rightarrow B : \{\text{“Transfer 1000€ from } A\text{'s account to } C\text{'s”}\}_{\langle Na,Nb\rangle}$$

Fig. 2. Lowe's attack to the NSPK++ protocol

$$4b. \quad B(C) \rightarrow A : \{\text{“Transfer 2000€ from } C\text{'s account to } B\text{'s”}\}_{\langle Na,Nb\rangle}$$

Fig. 3. Retaliation attack following Lowe's attack

The natural consequence of B's learning Na is the *retaliation attack* [16] in Figure 3. Note that the first method indicated above gives B a reasonable set of target principals to try retaliation against, while the second one gives him a probabilistic answer originated from traffic analysis. However, the remaining two methods exactly tell him who the target for retaliation is.

It is worth remarking once more that a retaliation attack cannot be captured in the standard DY threat model, where all potential attackers merely collude to form a super-potent one. However, the GA model can support this notion.

An important finding that will be detailed below is that B learns Na before C learns Nb (Figure 2). This may lead to the unknown scenario that sees B steal money by step $4b$ from Figure 3 before C does it by step $4a'$ from Figure 2. The more quickly does B use any of the first three methods given above to evaluate Na and pinpoint C, the more realistic this scenario. Potentially, B's illegal activity may even succeed before message 3 reaches C disclosing Nb. This attack will be addressed as *anticipation attack.*

This section attempted to motivate our interest in studying protocols beyond the DY model. It is well known that machine support is dramatically needed for protocol analysis. The next sections show how to tailor established model-checking techniques to validate protocols under the GA threat model.

3 Extending the Validation Method over the General Attacker

We have used one of the AVISPA backends to perform our experiments: the SAT-based model checker SATMC. This tool has successfully tackled the problem of determining whether the concurrent execution of a finite number of sessions of a protocol enjoys certain security properties in spite of the DY attacker

[22,23]. Leveraging on that work, we aim at relaxing the assumption of a single, super-potent attacker to specify the GA threat model, where principals can even compete each other. It was already stated above that our aim is to study novel protocol subtleties that go beyond the violation of standard security properties such as confidentiality and authentication. We are currently focusing on retaliation attacks and anticipation attacks. Also these notions can be recast into a model checking problem, as we shall see below.

3.1 Basics of SAT-Based Model Checking

This subsection outlines the basic definitions and concepts underlying SAT-based model checking. The reader who is familiar with such concepts can skip this. Let us recall that a model checking problem can be stated as

$$M \models G \tag{1}$$

where M is a labelled transition system modelling the initial state of the system and the behaviours of the principals (including their malicious activity) and G is an LTL formula expressing the security property to be checked.

The states of M are represented as sets of *facts* i.e. ground atomic formulas of a first-order language with sorts. If S is a state, then its facts are interpreted as the propositions holding in the state represented by S, all other facts being false in that state (closed-world assumption). A state is written down by the convenient syntax of a list of facts separated by the **.** symbol, as we shall see.

The transitions of M are represented as *set-rewriting rules* that define mappings between states. Each rule has a *label* expressing what the rule is there for: for example, the label $\mathtt{step}_i$ is for a rule that formalises the i-th legal protocol step, the label `overhear` is for a rule whereby an attacker reads some traffic, and so on. Each rule label is parameterised by the rule variables or proper instances of them, and we will encounter a number of self-explaining rule labels below.

Let $(L \xrightarrow{\rho} R)$ be (an instance of) a rewriting rule and S be a set of facts. If $L \subseteq S$ then we say that ρ is applicable in S and that $S' = \mathrm{app}_\rho(S) = (S \setminus L) \cup R$ is the state resulting from the execution of ρ in S. A *path* π is an alternating sequence of states and rules $S_0\rho_1S_1 \ldots S_{n-1}\rho_nS_n$ such that $S_i = \mathrm{app}_{\rho_i}(S_{i-1})$ (i.e. S_i is a state resulting from the execution of ρ_i in S_{i-1}), for $i = 1, \ldots, n$. Let $\mathcal{I}$ be the initial state of the transition system; if $S_0 \subseteq \mathcal{I}$, then we say that the path is *initialised*. Let $\pi = S_0\rho_1S_1 \ldots S_{n-1}\rho_nS_n$ be a path. We define $\pi(i) = S_i$ and $\pi_i = S_i\rho_{i+1}S_{i+1} \ldots S_{n-1}\rho_nS_n$. Therefore, $\pi(i)$ and π_i are the i-th state of the path and the suffix of the path starting with the i-th state respectively. Also, it is assumed that paths have infinite length. This can be always obtained by adding stuttering transitions to the transition system.

The language of LTL used here has facts and equalities over ground terms as atomic propositions, the usual propositional connectives (namely, $\neg$, $\vee$) and the temporal operators $\mathbf{X}$ (next), $\mathbf{F}$ (eventually) and $\mathbf{O}$ (once). Let π be an initialised path of M, an LTL formula ϕ *is valid on* π, written $\pi \models \phi$, if and only if $(\pi, 0) \models \phi$, where $(\pi, i) \models \phi$ (ϕ holds in π at time i) is inductively defined as:

$$\begin{array}{ll}
(\pi, i) \models f & \text{iff } f \in \pi(i) \ (f \text{ is a fact}) \\
(\pi, i) \models (t_1 = t_2) & \text{iff } t_1 \text{ and } t_2 \text{ are the same terms} \\
(\pi, i) \models \neg\phi & \text{iff } (\pi, i) \not\models \phi \\
(\pi, i) \models (\phi_1 \vee \phi_2) & \text{iff } (\pi, i) \models \phi_1 \text{ or } (\pi, i) \models \phi_2 \\
(\pi, i) \models \mathbf{X}\phi & \text{iff } (\pi, i+1) \models \phi \\
(\pi, i) \models \mathbf{F}\phi & \text{iff } \exists j \in [i, \infty).(\pi, j) \models \phi \\
(\pi, i) \models \mathbf{O}\phi & \text{iff } \exists j \in [0, i].(\pi, j) \models \phi
\end{array}$$

In the sequel we use $(\phi_1 \wedge \phi_2)$, $(\phi_1 \Rightarrow \phi_2)$ and $\mathbf{G}\phi$ as abbreviations of $\neg(\neg\phi_1 \vee \neg\phi_2)$, $(\neg\phi_1 \vee \phi_2)$ and $\neg\mathbf{F}\neg\phi$ respectively.

In Section 3.2 we show how the behaviours of the principals can be specified using a set-rewriting formalism. This amounts to specifying the model M of (1). In Section 3.3 we show how interesting protocol properties can be formalised by means of LTL formulae, which correspond to G in the problem (1)).

3.2 Formalising the General Attacker

We conveniently adopt the IF language as it can specify inputs to the AVISPA backends and more specifically to SATMC, a successful SAT-based model checker [22] that will be used in the final validation phase. The following syntactical conventions are adopted in the sequel.

- Lower-case typewriter fonts, such as `na`, denote IF constants.
- Upper-case typewriter fonts, such as `A`, denote IF variables.
- Lower-case italics fonts of 0 arity, such as s indicating a session identifier, compactly denote IF terms.
- Lower-case italics fonts of positive arity, such as $attack(a, v, s)$, denote meta-predicates that aim at improving the readability of this manuscript, but in fact do not belong to the current IF formalisation.
- Upper-case italics fonts serve diverse purposes, as specified each time.

To specify the GA threat model, a number of new facts must be defined in our language. They are summarised with their informal meaning in Table 1.

If S is a set of facts representing a state, then the state of principal a is represented by the facts of form $\mathtt{state}_r(j, a, es, s)$ (called *state-facts*) and of form $\mathtt{ak}(a, m)$ occurring in S. It is assumed that for each session s and for each principal a there exists at most one fact of the form $\mathtt{state}_r(j, a, es, s)$ in S. This does not prevent a principal from playing different roles in different sessions.

The dedicated account on the attacker's knowledge in DY must be extended as a general account on principals' knowledge in GA. In practice, what was the `ik` fact to represent the DY attacker knowledge is now replaced by `ak`, which has as an extra parameter the principal's identity whose knowledge is being defined. Incidentally, we conveniently write down the set of facts in a state by enumerating the facts and interleaving them a dot. Here is the definition of `ak`:

Table 1. New facts and their informal meaning

Fact	Holds when
$\mathtt{state}_r(j, a, es, s)$	Principal a, playing role r, is ready to execute step j in session s of the protocol, and has internal state es, which is a list of expressions affecting her future behaviour.
$\mathtt{ak}(a, m)$	Principal a knows message m.
$\mathtt{nt}(a)$	Principal a is not trustworthy.
$\mathtt{c}(n)$	Term n is the current value of the counter used to issue fresh terms, and is incremented as $\mathtt{s}(n)$ every time a fresh term is issued.
$\mathtt{msg}(rs, b, a, m)$	Principal rs has sent message m to principal a pretending to be principal b.
$\mathtt{contains}(db, m)$	Message m is contained into set db. Sets are used, e.g., to share data between honest principals.
$\mathtt{confidential}(m, g)$	Message m is a secret shared among the group of principals g (the set g is clearly populated through occurrences of $\mathtt{contains}(g, a)$ for each principal a intended to be in g).
$\mathtt{transferred}(rs, a, b, c, x, s)$	rs, in the disguise of a, transferred x€ from a's account to $c's$ at b (which intuitively is a bank) in session s.

$$\begin{aligned}
&\mathtt{ak}(a,m)\,.\,\mathtt{ak}(a,k) && \xrightarrow{\mathtt{encrypt}(\mathit{VARS}(a,k,m))} && \mathtt{ak}(a,\{m\}_k)\,.\,\mathit{LHS}\\
&\mathtt{ak}(a,\{m\}_k)\,.\,\mathtt{ak}(a,\overline{k}) && \xrightarrow{\mathtt{decrypt}(\mathit{VARS}(a,\overline{k},m))} && \mathtt{ak}(a,m)\,.\,\mathit{LHS}\\
&\mathtt{ak}(a,m_1)\,.\,\mathtt{ak}(a,m_2) && \xrightarrow{\mathtt{pairing}(\mathit{VARS}(a,m_1,m_2))} && \mathtt{ak}(a,\langle m_1,m_2\rangle)\,.\,\mathit{LHS}\\
&\mathtt{ak}(a,\langle m_1,m_2\rangle) && \xrightarrow{\mathtt{decompose}(\mathit{VARS}(a,m_1,m_2))} && \mathtt{ak}(a,m_1)\,.\,\mathtt{ak}(a,m_2)\,.\,\mathit{LHS}
\end{aligned}$$

where $\mathit{VARS}(t_1, \ldots, t_h)$ and LHS abbreviate in each rule, respectively, all the IF variables occurring in the IF terms represented by $t_1, \ldots, t_h$ and the set of facts occurring in the left hand side of the rule. Also, k and $\overline{k}$ are the inverse keys of one another.

Under the GA threat model any principal may behave as a DY attacker. We may nonetheless need to formalise protocols that encompass trusted third parties, or where a principal loses her trustworthiness due to a certain event. Reading through Table 1, it can be seen that our machinery features the $\mathtt{nt}(a)$ predicate, holding of a principal a that is not to be trusted. It may conveniently be used either statically, if added to the initial state of principals, or dynamically when introduced by rewriting rules. It is understood that if all principals but one are declared as trustworthy, then we are back to the DY threat model.

The fact $\mathtt{c}(n)$ holds of the current counter n used to generate fresh nonces. For example, $\mathtt{c}(0)$ holds in the initial state, and $\mathtt{c}(\mathtt{s}(0))$ after the generation of the first fresh nonce, which takes the value 0. More generally, if the fact $\mathtt{c}(\mathtt{na})$ holds, a fresh nonce $\mathtt{na}$ can be issued producing another state with counter $\mathtt{c}(\mathtt{s}(\mathtt{na}))$.

The initial state of the system defines the initial knowledge and the state-facts of all principals involved in the considered protocol sessions. Its standard definition is omitted here but can be found elsewhere [3]. Appropriate rewriting rules specify the evolution of the system. Those for honest principals, which also serve to demonstrate the remaining facts enumerated in Table 1, are of the form:

$$\begin{aligned}&\mathtt{msg}(rs, b_1, a, m_1)\,\mathbf{.}\,\mathtt{state}_r(i, a, es, s) \xrightarrow{\mathtt{step}_l(\mathit{VARS}(a,b_1,b_2,rs,es,es',m_1,m_2,s))} \\ &\qquad \mathtt{msg}(a, a, b_2, m_2)\,\mathbf{.}\,\mathtt{state}_r(j, a, es', s)\,\mathbf{.}\,\mathtt{ak}(a, m_1)\,\mathbf{.}\,\mathtt{ak}(a, m_2) \qquad (2)\end{aligned}$$

where l is the step label, i and j are integers and r is a protocol role (e.g., the NSPK++ has two roles Alice and Bob, also said Initiator and Responder roles). Rule 2 models the reception by a principal of a message and the principal's sending of the next message according to the protocol. More precisely, it states that if principal a, who is playing role r at step i with internal state es in session s of the protocol, has received message m_1 supposedly from b_1 (while the real sender is rs), then she can honestly send message m_2 to b_2. In doing so, a updates her internal state as es' and her knowledge accordingly, that is the new state registers the facts $\mathtt{ak}(a, m_1)$ and $\mathtt{ak}(a, m_2)$. Note that rule 2 may take slightly different forms depending on the protocol step it models. For example, if $j = 1$ and a sends the first message of the protocol, the fact $\mathtt{msg}(rs, b_1, a, m_1)$ does not appear in the left hand side of the rule, reflecting a's freedom to initiate the protocol at anytime. Similarly, for generating and sending a fresh term, $\mathtt{c}(\mathtt{N})$ is included in the left hand side of the rule, while $\mathtt{c}(\mathtt{s}(\mathtt{N}))$ and $\mathtt{ak}(a, \mathtt{N})$ appear in the right hand side to express the incremented counter and the principal's learning the fresh term. A further variant is necessary when the step involves either a membership test or an update of a set of elements. In this case, facts of the form $\mathtt{contains}(db, m)$ must be properly defined. Facts such as $\mathtt{confidential}(m, g)$ or $\mathtt{transferred}(rs, b, a, c, x, s)$ added to the right hand side express respectively confidentiality for a group of principals, and a successful transfer of money.

To illustrate the specification of concrete protocol rules we consider two steps of the NSPK++ protocol. The transition in which B receives the first message of the protocol (supposedly) from A and replies with the second protocol message is modelled by the following rule:

$$\begin{aligned}&\mathtt{msg}(\mathtt{RS}, \mathtt{A}, \mathtt{B}, \{\langle \mathtt{A}, \mathtt{NA}\rangle\}_{\mathtt{KB}})\,\mathbf{.}\,\mathtt{state}_{\mathtt{bob}}(1, \mathtt{B}, [\mathtt{A}, \mathtt{KA}, \mathtt{KB}, \mathtt{G}], \mathtt{S})\,\mathbf{.}\,\mathtt{c}(\mathtt{NB}) \\ &\xrightarrow{\mathtt{step}_2(\mathtt{B,A,RS,KA,KB,NA,NB,G,S})} \\ &\mathtt{msg}(\mathtt{B}, \mathtt{B}, \mathtt{A}, \{\langle \mathtt{NA}, \mathtt{NB}\rangle\}_{\mathtt{KA}})\,\mathbf{.}\,\mathtt{state}_{\mathtt{bob}}(3, \mathtt{B}, [\mathtt{A}, \mathtt{KA}, \mathtt{KB}, \mathtt{NA}, \mathtt{NB}, \mathtt{G}], \mathtt{S})\,\mathbf{.}\,\mathtt{ak}(\mathtt{B}, \{\langle \mathtt{A}, \mathtt{NA}\rangle\}_{\mathtt{KB}})\,\mathbf{.} \\ &\mathtt{ak}(\mathtt{B}, \mathtt{NB})\,\mathbf{.}\,\mathtt{contains}(\mathtt{G}, \mathtt{A})\,\mathbf{.}\,\mathtt{contains}(\mathtt{G}, \mathtt{B})\,\mathbf{.}\,\mathtt{confidential}(\mathtt{NB}, \mathtt{G})\,\mathbf{.}\,\mathtt{c}(\mathtt{s}(\mathtt{NB}))\end{aligned}$$

where G represents the group of principals that are allowed to share the freshly generated nonce NB. This also illustrates our different treatment of confidentiality with respect to DY's. While DY reduced confidentiality of a message to keeping the message confidential from the attacker, GA requires the original, subtler and unsimplified, definition of confidentiality: "confidentiality is the protection of information from disclosure to those not intended to receive it" [24]. For example, it regards the confidentiality of a message as compromised if ever anyone beyond its intended peers learns it.

To provide another example of a protocol rule, the completion step 4*b* in which A receives and then executes a money transfer from B is modeled by two rules, one for B's sending and one for A's reception. Here is the latter:

$$\begin{array}{l}\mathtt{msg}(\mathtt{RS},\mathtt{B},\mathtt{A},\{\mathtt{tr}(\mathtt{B},\mathtt{C},\mathtt{X})\}_{\langle\mathtt{NA},\mathtt{NB}\rangle})\,\mathbf{.}\,\mathtt{state}_{\mathtt{alice}}(4,\mathtt{A},[\mathtt{B},\mathtt{KA},\mathtt{KB},\mathtt{NA},\mathtt{NB},\mathtt{G}],\mathtt{S})\\ \quad\xrightarrow{\mathtt{step}_{\mathtt{4b\_rec}}(\mathtt{A,B,C,RS,KA,KB,NA,NB,X,G,S})}\\ \mathtt{state}_{\mathtt{alice}}(4,\mathtt{A},[\mathtt{B},\mathtt{KA},\mathtt{KB},\mathtt{NA},\mathtt{NB},\mathtt{G}],\mathtt{S})\,\mathbf{.}\,\mathtt{ak}(\mathtt{A},\{\mathtt{tr}(\mathtt{B},\mathtt{C},\mathtt{X})\}_{\langle\mathtt{NA},\mathtt{NB}\rangle})\,\mathbf{.}\,\mathtt{ak}(\mathtt{A},\mathtt{X})\,\mathbf{.}\\ \mathtt{transferred}(\mathtt{RS},\mathtt{B},\mathtt{A},\mathtt{C},\mathtt{X},\mathtt{S})\end{array}$$

The rule states the fact $\mathtt{transferred}(\mathtt{RS},\mathtt{B},\mathtt{A},\mathtt{C},\mathtt{X},\mathtt{S})$ to record that RS, who is not necessarily B, transferred X€ from B's account to $\mathtt{C}'s$ at A (which intuitively is a bank) in session S. This is not to be confused with $\{\mathtt{tr}(\mathtt{B},\mathtt{C},\mathtt{X})\}_{\langle\mathtt{NA},\mathtt{NB}\rangle}$, which is a message expressing the request of a transfer.

The malicious behaviour of each principal C acting as a DY attacker can be specified by the following rules:

$$\begin{array}{lll}\mathtt{nt}(\mathtt{C})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{M})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{A})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{B}) & \xrightarrow{\mathtt{fake(C,A,B,M)}} & \mathtt{msg}(\mathtt{C},\mathtt{A},\mathtt{B},\mathtt{M})\,\mathbf{.}\,\mathit{LHS}\\ \mathtt{nt}(\mathtt{C})\,\mathbf{.}\,\mathtt{msg}(\mathtt{RS},\mathtt{A},\mathtt{B},\mathtt{M}) & \xrightarrow{\mathtt{overhear(RS,A,B,C,M)}} & \mathtt{ak}(\mathtt{C},\mathtt{M})\,\mathbf{.}\,\mathit{LHS}\\ \mathtt{nt}(\mathtt{C})\,\mathbf{.}\,\mathtt{msg}(\mathtt{RS},\mathtt{A},\mathtt{B},\mathtt{M}) & \xrightarrow{\mathtt{intercept(RS,A,B,C,M)}} & \mathtt{ak}(\mathtt{C},\mathtt{M})\,\mathbf{.}\,\mathtt{nt}(\mathtt{C})\end{array}$$

Although the model outlined so far is accurate, it is not the most appropriate to perform automatic analysis efficiently. This is not a big issue for validating the NSPK++ protocol in particular, but it is in general. The main problem is the specification of the malicious behaviour of principals. It allows for the forging of messages that will clearly not help to attack the protocol, as they do not correspond to the forms that the protocol prescribes. In other words, forging messages that no one will ever accept is of no use to any attacker. Efficiency of the analysis improves by adopting a refined model of malicious activity, for example by introducing a forged message only if it conforms to one of the forms that belong to the protocol. More narrow-scoped, though realistic, impersonate rules detailed elsewhere [25, §3.2.4] can easily be recast in the GA threat model. For example, the following impersonate rule models a principal C who, pretending to be A, sends the first message of our example protocol to B, who is exactly waiting for a message of that form:

$$\begin{array}{l}\mathtt{nt}(\mathtt{C})\,\mathbf{.}\,\mathtt{state}_{\mathtt{bob}}(2,\mathtt{B},[\mathtt{A},\mathtt{KA},\mathtt{KB},\mathtt{G}],\mathtt{S})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{A})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{NA})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\mathtt{KB})\\ \quad\xrightarrow{\mathtt{impersonate}_{2}(\mathtt{C,A,B,KA,KB,NA,G,S})}\\ \mathtt{msg}(\mathtt{C},\mathtt{A},\mathtt{B},\{\langle\mathtt{A},\mathtt{NA}\rangle\}_{\mathtt{KB}})\,\mathbf{.}\,\mathtt{ak}(\mathtt{C},\{\langle\mathtt{A},\mathtt{NA}\rangle\}_{\mathtt{KB}})\,\mathbf{.}\,\mathit{LHS}\end{array}$$

3.3 Formalising Protocol Properties under the General Attacker

One may expect that standard security properties such as confidentiality and authentication can be routinely specified and checked under the GA threat model using a model checker. Yet, confidentiality deserves particular consideration in what the GA model differs from the DY one.

In general, the confidentiality of a message m w.r.t. a group of principals g is guaranteed if and only if m is only known to principals in g. This property is clearly violated anytime a principal a outside g learns, for any reason, m. However, under DY, where all principals but the attacker meticulously follow the protocol, the violation is by design not considered so unless a is the attacker. There are many real scenarios—ranging from a betrayed person who publishes his/her partner's credit card details, to a scenario where a fired employee discloses sensitive information about its former employer—in which this violation is significant and therefore not negligible. In the GA model this confidentiality breach is not overlooked, as it can be checked by the following meta-predicate formalising a violation of confidentiality:

$$voc(a, m, g) = \mathbf{F}(\mathtt{confidential}(m, g) \wedge \mathtt{ak}(a, m) \wedge \neg\, \mathtt{contains}(g, a)\ ) \tag{3}$$

The negation of the meta-predicate given in Definition 3 corresponds to G in the model checking problem 1 stated in Section 3.1, and represents the confidentiality property.

What is more interesting is that the GA threat model paves the way to investigate subtler protocol properties than confidentiality and authentication. In Section 2, we observed that retaliation is a common practice in the real world. (Even anyone who is most peaceful may react under attack—whether this is fortunate or unfortunate lies outside our focus.) Depending on who reacts against the attacker, retaliation can be named differently [16]: if it is the victim, then there is *direct retaliation*; if it is someone else, then there is *indirect retaliation.* Let $attack(c, a, b)$ be a meta-predicate holding if and only if an attacker c has successfully attacked a victim a (being $a \neq c$ and $a \neq b$) with the (unaware) support of b (if any). Of course, "has successfully attacked" denotes the violation of the specific property that the protocol under analysis is supposed to achieve. Any violation of a protocol property should make the predicate hold, and therefore the predicate should be defined by cases. If we focus for simplicity on the didactic attack to the NSPK++ protocol seen above (Figure 2), which is an illegal money transfer, then the meta-predicate can be defined as:

$$attack(c, a, b) = \mathtt{transferred}(c, a, b, r, x, s) \wedge c \neq a \tag{4}$$

Clearly, this definition can be extended to check other kinds of attacks. Because not all money transfers are illegal, the second conjunct in the formula is crucial: the illegal transfers are only those that are not requested by the account holder.

Direct and indirect retaliation can be respectively modelled as the following meta-predicates:

$$direct\_retaliation(a, c) = \mathbf{F}(\ attack(c, a, b1) \wedge \mathbf{X}\,\mathbf{F}\, attack(a, c, b2)\ ) \tag{5}$$

$$indirect\_retaliation(a, c, b) = \mathbf{F}(\ attack(c, a, b) \wedge \mathbf{X}\,\mathbf{F}\, attack(b, c, a)\ ) \tag{6}$$

It can be seen that the formula defined by Definition 5 is valid on those paths where an attacker hits a victim who hits the attacker back. Each attack can be carried out with the help of potentially different supporters $b1$ and $b2$. By contrast, Definition 6 of indirect retaliation shows that who hits the attacker back is not the victim but the supporter, who perhaps realises what he has just done and decides to rebel against the attacker.

The definition of *anticipation attack* is deferred to the next section because it is best demonstrated upon the actual experiments.

4 Validating the NSPK++ Protocol under the General Attacker

The previous Section outlined the model checking problem in general, and described how to formalise the GA threat model for the validation of security protocols. For the sake of demonstration, it presented the formalisation of a step of the NSPK++ protocol. It concluded with the specification of protocol properties under the GA threat model.

Having digested the innovative aspects, deriving the full formalisation of the NSPK++ protocol under the General Attacker, which is omitted here, became an exercise. That formalisation was fed to SATMC, a state-of-the-art SAT-based model checker for security protocols, to carry out the first protocol validation experiments under GA. The details of SATMC appear elsewhere [3,23]. Its core is a procedure that automatically generates a propositional formula. The satisfying assignments of this formula, if any exist, correspond to counterexamples (i.e. execution traces of M that falsify G) of length bounded by some integer k, which can be iteratively deepened. Finding violations (of length k) on protocol properties boils down to solving propositional satisfiability problems. SATMC accomplishes this task by invoking state-of-the-art SAT solvers, which can handle satisfiability problems with hundreds of thousands of variables and clauses.

In running SATMC over the formalisation of the NSPK++ protocol, we considered the classical scenario in which `a` wants to talk with `b` in one session and with `c` in another session. Because the formalisation accounted for the new threat model, we were pleased to observe that it passed simple sanity checks: SATMC outputs the known confidentiality attack whereby `c` learns `nb`, and the known authentication attack whereby `c` impersonates `a` with `b`. As these are well known, they are omitted here. More importantly, the tool also reported what are our major findings: `b`'s confidentiality attack upon `na`, and interesting retaliation attacks, which are detailed in the following. We argue that this is the first mechanised treatment of these subtle properties, laying the ground for much more computer-assisted analysis of security protocols.

```
0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
1: [overhear(c,a,a,c,{na,a}kc)]
2: [decrypt(c,inv(kc),{na,a})]
3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
4: [step_2(b,a,c,ka,kb,na,nb,set_ab,2)]
5: [decrypt(b,inv(kb),{na,a})]
```

Fig. 4. Trace of NSPK++ featuring `b`'s confidentiality attack

```
 6: [impersonate_3(b,c,a,ka,kc,na,nb,set_ac,1)]
 7: [step_3(a,c,b,ka,kc,na,nb,set_ac,1)]
 8: [overhear(c,a,a,c,{nb}kc)]
 9: [decrypt(c,inv(kc),{nb})]
10: [impersonate_3_rec(c,a,b,ka,kb,nb,set_ab)]
11: [step_3_rec(b,a,c,ka,kb,na,nb,set_ab,2)]
12: [impersonate_4a_rec(c,a,b,c,ka,kb,na,nb,set_ab,2)]
13: [step_4a_rec(b,a,c,c,ka,kb,na,nb,1K,set_ab,2)]
14: [impersonate_4b_rec(b,c,a,b,ka,kc,na,nb,set_ac,1)]
15: [step_4b_rec(a,c,b,b,ka,kc,na,nb,2K,set_ac,1)]
```

Fig. 5. Trace of NSPK++ (continuation) featuring b's indirect retaliation

Figure 4 reports the protocol trace that the tool outputs when checking a violation of confidentiality by the formula $voc(\mathtt{A}, \mathtt{M}, \mathtt{G})$. Precisely, the trace is obtained by mildly polishing the partial-order plan returned by SATMC. Each trace element reports the label of the rule that fired, and hence the trace can be interpreted as the history of events leading to the confidentiality attack.

A full description of the trace requires a glimpse at the protocol formalisation —each rule label in a trace element must be matched to the actual rule— but we shall see that the trace can be automatically converted into a more user-friendly version. Element 0 means that principal a initiates the protocol with principal c. Elements 1, 2 and 3 show c's illegal activities respectively of getting the message, decrypting it and forging a well-formed one for principal b. Although c does not need in practice to overhear a message meant for him, element 1 is due to our monolithic formalisation of the acts of receiving a message and of sending out its protocol-prescribed reply. Therefore, for a principal to abuse a message, the principal must first overhear it. Element 3 reminds that c's forging of the message for b counts as an attempt to impersonate a. The last two elements signal b's legal participation in the protocol by receiving the message meant for him, and his subsequent deciphering of the nonce. SATMC now returns because $voc(\mathtt{b}, \mathtt{na}, \mathtt{set\_ac})$ holds (where the set $\mathtt{set\_ac} = \{a, c\}$).

To illustrate the detection of a retaliation attack, SATMC can be launched on a significant property such as $indirect\_retaliation(\mathtt{A}, \mathtt{B}, \mathtt{C})$. It returns a trace that continues as in Figure 5 the one seen in Figure 4. The impersonate rules show that both b and c are acting illegally in this trace, a development that has never been observed by previous analyses under DY. Elements 6 and 7 show that b is impersonating c with a, who naively replies to c. The next three elements confirm c's attempt at fooling b, who legally replies as element 11 indicates. Now c can finalise his attack as in elements 12 and 13. The latter element indicates the firing of rule $4a$, which makes $attack(\mathtt{c}, \mathtt{a}, \mathtt{b})$ hold. The last two elements witness b's retaliation attack by making $attack(\mathtt{b}, \mathtt{c}, \mathtt{a})$ hold. Therefore, by Definition 6, the property $indirect\_retaliation(\mathtt{a}, \mathtt{c}, \mathtt{b})$ holds, indicating that the tool reports the retaliation attack described above (Figure 3).

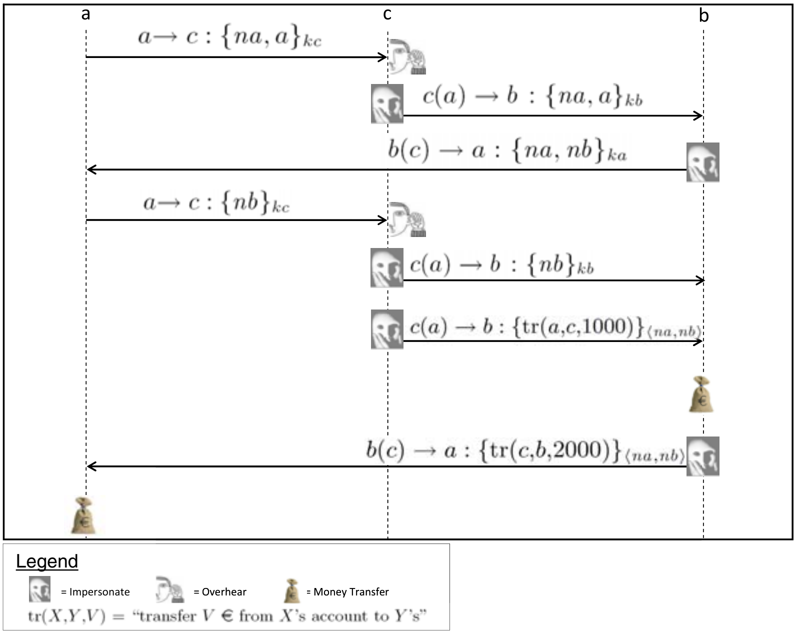

a
c
b
$a \rightarrow c : \{na, a\}_{kc}$
$c(a) \rightarrow b : \{na, a\}_{kb}$
$b(c) \rightarrow a : \{na, nb\}_{ka}$
$a \rightarrow c : \{nb\}_{kc}$
$c(a) \rightarrow b : \{nb\}_{kb}$
$c(a) \rightarrow b : \{\mathrm{tr}(a,c,1000)\}_{\langle na,nb \rangle}$
$b(c) \rightarrow a : \{\mathrm{tr}(c,b,2000)\}_{\langle na,nb \rangle}$
Legend
= Impersonate
= Overhear
= Money Transfer
$\mathrm{tr}(X,Y,V)$ = "transfer V € from X's account to Y's"

Fig. 6. Graphics of b's indirect retaliation in NSPK++

A graphical illustration of this retaliation attack is in Figure 6, and can be easily built. First, we run a procedure to transform the output of the tool into a more readable and intuitive version. Then, we coherently associate the significant phrases of this version to graphical elements, and hence build the image. The behaviours and interactions of the principals can be observed by looking at the figure from top to bottom. The overhear and impersonate icons emphasise the significant number of illegal steps taken by both principals c and b).

We run the tool repeatedly on the protocol model, each time adjusting the property to check. In particular, we tried an alternative definition of indirect retaliation where the **X** operator was left out (see Definition 6). Somewhat to our surprise, the tool returned a trace leading to a state where both attacks held, as if retaliation did not need be triggered by another attack. This called for more attention at the trace.

In consequence, we observed from element 3 in Figure 4 that c initialises his attack very early—precisely, with his impersonation of a based upon the repetition to b of a's nonce na. This means that b learns na when c has not yet learned nb and hence cannot attack yet. We thus defined the self-explaining fact nonce_leak(c, a, b). In the GA threat model, it is plausible that b exploits his knowledge before c does. This can be interpreted as a scenario in which a potential victim realises that he is going to be attacked, and therefore reacts successfully before being actually attacked. We name such a successful reaction *anticipation attack* and define it as a meta-predicate below. For the sake of

```
 0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
 1: [overhear(c,a,a,c,{na,a}kc)]
 2: [decrypt(c,inv(kc),{na,a})]
 3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
 4: [overhear(b,c,a,b,{na,a}kb)]
 5: [decrypt(b,inv(kb),{na,a})]
 6: [i_got_you(c,a,b,na,set_ac)]
 7: [impersonate_3(b,c,a,ka,kc,na,nb,set_ac,1)]
 8: [step_3(a,c,b,ka,kc,na,nb,set_ac,1)]
 9: [impersonate_4b_rec(b,c,a,b,ka,kc,na,nb,set_ac,1)]
10: [step_4b_rec(a,c,b,b,ka,kc,na,nb,2K,set_ac,1)]
```

Fig. 7. Trace of NSPK++ featuring b's anticipation attack

efficiency, we decided to add a rule that introduces the fact $\mathtt{nonce\_leak}(\mathtt{C},\mathtt{A},\mathtt{B})$ following c's attack initialisation:

$$\begin{array}{l}\mathtt{confidential}(\mathtt{NA},\mathtt{G})\,\textbf{.}\,\mathtt{ak}(\mathtt{B},\mathtt{NA})\,\textbf{.}\,\neg\,\mathtt{contains}(\mathtt{G},\mathtt{B})\,\textbf{.}\\ \mathtt{ak}(\mathtt{B},\mathtt{KB}^{-1})\,\textbf{.}\,\mathtt{msg}(\mathtt{C},\mathtt{A},\mathtt{B},\{\langle\mathtt{A},\mathtt{NA}\rangle\}_{\mathtt{KB}})\\ \quad\xrightarrow{\mathtt{i\_got\_you}(\mathtt{C},\mathtt{A},\mathtt{B},\mathtt{NA},\mathtt{G})}\\ \mathtt{nonce\_leak}(\mathtt{C},\mathtt{A},\mathtt{B})\end{array}$$

A meta-predicate $\mathit{attack\_init}(c,a,b)$ must be introduced to formalise some c's act of initialising an attack, which is yet to be carried out, while interleaving sessions with some a and b. In the same fashion of Definition 4 of $\mathit{attack}(c,a,b)$, we provide a definition that is appropriate for the attack under analysis, and corresponds to the nonce leak:

$$\mathit{attack\_init}(c,a,b) = \mathtt{nonce\_leak}(c,a,b)$$

The definition of anticipation attack can be given now. It insists that an attack initiated by someone, such as c, is followed by an actual attack carried out by someone else, such as b:

$$\begin{aligned}\mathit{anticipation\_attack}(c,a,b) = \mathbf{F}(\ &\mathit{attack\_init}(c,a,b)\ \wedge\\ &\mathbf{X}\,\mathbf{F}\,\mathit{attack}(b,c,a)\ )\end{aligned}\tag{7}$$

SATMC validated our intuition. When run on $\mathit{anticipation\_attack}(\mathtt{C},\mathtt{A},\mathtt{B})$ as a goal, the tool produced the partial order plan reported in Figure 7. Element 6 in the trace indicates that c leaked a nonce created by a by sending it to b. So, the meta-predicate $\mathit{attack\_init}(\mathtt{c},\mathtt{a},\mathtt{b})$ holds. Moreover, the last two elements witness b's anticipation attack by making $\mathit{attack}(\mathtt{b},\mathtt{c},\mathtt{a})$ hold. Therefore, by Definition 7, we have that $\mathit{anticipation\_attack}(\mathtt{c},\mathtt{a},\mathtt{b})$ is true, which signifies that the tool reports the anticipation attack described above (Figure 8).

Our various experiments produced another interesting finding—SATMC reported a trace describing the following scenario. In a session, a discloses to c her nonce generated for b. In another session, a discloses to b another nonce of hers generated for c. In consequence, both b and c become capable of attacking each

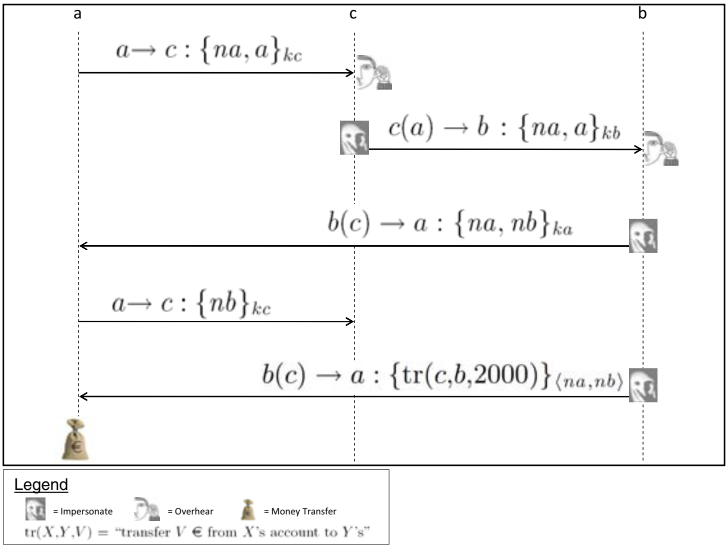

a
c
b
$a \rightarrow c : \{na, a\}_{kc}$
$c(a) \rightarrow b : \{na, a\}_{kb}$
$b(c) \rightarrow a : \{na, nb\}_{ka}$
$a \rightarrow c : \{nb\}_{kc}$
$b(c) \rightarrow a : \{\text{tr}(c,b,2000)\}_{\langle na,nb \rangle}$
Legend
= Impersonate
= Overhear
= Money Transfer
$\text{tr}(X,Y,V)$ = "transfer V € from X's account to Y's"

Fig. 8. Graphics of b's anticipation attack in NSPK++

other with a's support if needed. Moreover, b and c may be initially unaware of each other's capability of attack. This reflects the real-world situation in which someone creates strife in a couple that starts fighting.

5 Conclusions

The General Attacker threat model seems most appropriate to the present social/technological setting. Reasoning that was impossible under DY can now be carried out, highlighting protocol niceties that are routinely overseen. Our work required care in formalising the protocols under the new threat model, but no changes to the state-of-the-art model checker SATMC.

Retaliation. teaches us that we can perhaps live with flawed protocols. We are used to go back to design when a protocol is found flawed, even if already deployed. However, an attack that can be retaliated may in practice convince an attacker to refrain from attacking in the first place. If the "cost" of attacking overdoes its "benefits" then the attacker will not be carried out. Retaliation makes that precondition hold.

Anticipation. teaches us to ponder the entire sequence of events underlying an attack. An attack typically is an interleaving of legal and illegal steps rather than a single illegal action. Therefore, we may face a scenario, unreported so far, where a principal mounts an attack by successfully exploiting for his own sake the illegal activity initiated but not yet finalised by someone else. This is routine for the present hackers' community.

So used as we are, as protocol analysers, to reasoning in terms of a DY attacker, we might think of our novel investigations as unrealistic. However, scenarios of retaliation and anticipation often appear in our era of computer networks, notably in the area of intrusion management. With the increasing awareness of computer security issues, successful attacks rarely are instantaneous moves without consequences. Rather, they often involve a combination of smaller illegal achievements or the breaking of subsequent protection levels to succeed. These are typically monitored by either administrators or other attackers, and hence attacks can be anticipated or retaliated. The field of protocol analysis seems bound to develop much further.

Acknowledgements. This work was supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (`www.avantssar.eu`).

References

1. Bella, G.: The Rational Attacker (2008) (invited talk at SAP Research France, Sophia Antipolis),
`http://www.dmi.unict.it/~giamp/Seminars/rationalattackerSAP08.pdf`
2. Armando, A., Basin, D.A., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
3. Armando, A., Compagna, L.: SAT-based Model-Checking for Security Protocols Analysis. International Journal of Information Security 7(1), 3–32 (2008)
4. Abadi, M., Gordon, A.: A calculus for cryptographic protocols: the spi calculus. Information and Computation 148(1), 1–70 (1999)
5. Ryan, P.Y.A., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A.W.: Modelling and Analysis of Security Protocols. In: AW (2001)
6. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. Journal of Computer Security 7, 191–230 (1999)
7. Caleiro, C., Viganò, L., Basin, D.: Relating strand spaces and distributed temporal logic for security protocol analysis. Logic Journal of the IGPL 13(6), 637–663 (2005)
8. Bella, G.: Formal Correctness of Security Protocols. In: Information Security and Cryptography. Springer, Heidelberg (2007)
9. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128 (1998)
10. Blanchet, B.: Automatic verification of cryptographic protocols: a logic programming approach. In: Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, Uppsala, Sweden, August 27-29, pp. 1–3 (2003)
11. Lowe, G.: Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)

12. Bellare, M., Rogaway, P.: Provably secure session key distribution– the three party case. In: Proceedings 27th Annual Symposium on the Theory of Computing, pp. 57–66. ACM, New York (1995)
13. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 3–22. Springer, Heidelberg (2000)
14. Gollmann, D.: On the verification of cryptographic protocols — a tale of two committees. In: Proc. of the Workshop on Secure Architectures and Information Flow. ENTCS, vol. 32. Elsevier Science, Amsterdam (2000)
15. Backes, M., Pfitzmann, B.: Relating symbolic and cryptographic secrecy. In: IEEE Symposium on Security and Privacy (2005)
16. Bella, G., Bistarelli, S., Massacci, F.: Retaliation: Can we live with flaws? In: Essaidi, M., Thomas, J. (eds.) Proc. of the Nato Advanced Research Workshop on Information Security Assurance and Security. Nato through Science, vol. 6, pp. 3–14. IOS Press, Amsterdam (2006), `http://www.iospress.nl/loadtop/load.php?isbn=9781586036782`
17. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 2(29) (1983)
18. Bella, G., Bistarelli, S.: Confidentiality levels and deliberate/indeliberate protocol attacks. In: Christianson, B., Crispo, B., Harbison, W.S., Roe, M. (eds.) Security Protocols 2002. LNCS, vol. 2845, pp. 104–119. Springer, Heidelberg (2004)
19. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: Bar fault tolerance for cooperative services. ACM SIGOPS Operating Systems Review 39(5), 45–58 (2005)
20. Buttyán, L., Hubaux, J.P., Čapkun, S.: A formal model of rational exchange and its application to the analysis of syverson's protocol. Journal of Computer Security 12(3-4), 551–587 (2004)
21. Bella, G.: What is Correctness of Security Protocols? Springer Journal of Universal Computer Science 14(12), 2083–2107 (2008)
22. Armando, A., Compagna, L.: SAT-based Model-Checking for Security Protocols Analysis. International Journal of Information Security 6(1), 3–32 (2007)
23. Armando, A., Carbone, R., Compagna, L.: LTL Model Checking for Security Protocols. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF20), Venice, Italy, July 6-8. LNCS. Springer, Heidelberg (2007)
24. Neuman, B.C., Ts'o, T.: Kerberos: An authentication service for computer networks, from IEEE communications magazine. In: Stallings, W. (ed.) Practical Cryptography for Data Internetworks, September 1994. IEEE Press, Los Alamitos (1996)
25. Compagna, L.: SAT-based Model-Checking of Security Protocols. Phd, Università degli Studi di Genova, Italy, and University of Edinburgh, Scotland (2005)

Usage Automata

Massimo Bartoletti

Dipartimento di Matematica e Informatica, Università degli Studi di Cagliari

Abstract. Usage automata are an extension of finite stata automata, with some additional features (e.g. parameters and guards) that improve their expressivity. Usage automata are expressive enough to model security requirements of real-world applications; at the same time, they are simple enough to be statically amenable, e.g. they can be model-checked against abstractions of program usages. We study here some foundational aspects of usage automata. In particular, we discuss about their expressive power, and about their effective use in run-time mechanisms for enforcing usage policies.

1 Introduction

Security is a major concern in the design and implementation of modern programming languages. For instance, both Java and C# offer a whole range of security features, from the "safety pillars" of bytecode verification and secure class loading, to more high-level defence mechanisms, like e.g. stack inspection and cryptography APIs [10].

However, mainstream programming languages typically do not offer any facilities to specify and enforce custom policies on the usage of resources. Therefore, it is common practice to renounce to separating duties between functionality and security, and to implement the needed enforcement mechanism with local checks explicitly inserted into the code by programmers. Since forgetting even a single check might compromise the security of the whole application, programmers have to inspect their code very carefully. This may be cumbersome even for small programs, and it may also lead to unnecessary checking.

History-based security has been repeatedly proposed as a replacement for stack inspection [1,8,17]. Clearly, the ability of checking the whole execution history, instead of the call stack only, places history-based mechanisms a step forward stack inspection, from the expressivity viewpoint. However, since many possible history-based models can be devised, it is crucial to choose one which wisely conciliates the expressive power with the theoretical properties enjoyed. It is also important that the enforcement mechanism can be implemented transparently to programmers, and with an acceptable run-time overhead.

In this paper we study *usage automata* as a formalism for defining and enforcing usage policies. Usage automata extend finite state automata, by allowing edges to carry *variables* and *guards*. Variables represent universally quantified resources. Since variables may range over an *infinite* set of resources, usage automata are a suitable formalism for expressing policies with parameters ranging

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 52–69, 2009.

over infinite domains (e.g. the formal parameters of a procedure). For instance, a usage automaton might state that "for all files x, you can read or write x only if you have opened x, and not closed it by the while". Guards represent conditions among variables (and resources). For instance, "a file y cannot be read if some other file $x \neq y$ has been read in the past".

Usage automata have been proved expressive enough to model security requirements of real-world applications. For instance, we used them to specify the typical set of policies of a bulletin board system [13]. At the same time, usage automata are simple enough to be statically amenable, e.g. they can be model-checked against abstractions (namely, history expressions) of program usages [4].

In this paper we investigate some foundational issues about usage automata.

In Section 2 we formally define their semantics, and we show that usage automata recognize safety properties enforceable through finite state automata.

In Section 3 we turn our attention to expressivity issues. We show that removing guards diminishes the expressive power of usage automata, while removing polyadic events (i.e. only allowing for monadic ones) keeps it unaltered. We then prove that the expressive power of usage automata increases as the number of variables therein increases. To do that, we introduce a proof technique (a sort of "deflating lemma"), stating that a usage automaton with k variables cannot distinguish among more than k resources.

In Section 4 we discuss the feasibility of run-time monitoring policies defined by usage automata. This question naturally arises from the definition of the semantics of usage automata, which requires an unbounded amount of space to decide whether a program trace respects a policy or not. We define an enforcement mechanism based on usage automata, which only requires a bounded amount of space. Some possible optimizations of the enforcement mechanism, as well as some implementation issues, are then highlighted.

We conclude with Section 5, by discussing some related work.

2 Usage Automata

Assume a (possibly infinite) set of *resources* $r, r', \ldots \in \mathsf{Res}$, which can be accessed through a given finite set of *actions* $\alpha, \alpha', \ldots \in \mathsf{Act}$. Each action α has a given *arity* $|\alpha|$, that is the number of resources it operates upon. An *event* $\alpha(r_1, \ldots, r_k) \in \mathsf{Ev}$ models the action α (with arity k) being fired on the target resources $r_1, \ldots, r_k$. *Traces*, typically denoted by $\eta, \eta', \ldots \in \mathsf{Ev}^*$, are finite sequences of events. A *usage policy* is a set of traces.

In this paper we study usage automata as a formalism for defining and enforcing usage policies. Usage automata can be seen as an extension of finite state automata (FSA), where the labels on the edges may contain variables and guards. Variables $x, y, \ldots \in \mathsf{Var}$ represent universally quantified resources. Guards $g, g', \ldots \in G$ represent conditions among variables (and resources).

Before introducing usage automata, we formally define the syntax and semantics of guards.

Definition 1. Guards

Let $\rho, \rho', \ldots \in \mathsf{Res} \cup \mathsf{Var}$. *We inductively define the set* G *of guards as follows:*

$$G ::= true \mid \rho = \rho' \mid \neg G \mid G \wedge G$$

For all guards $g \in G$*, we define* $var(g)$ *as the set of variables occurring in* g*. Let* g *be a guard, and let* $\sigma : var(g) \to \mathsf{Res}$*. Then, we write* $\sigma \models g$ *when:*

- $g = true$*, or*
- $g = (\rho = \rho')$ *and* $\rho\sigma = \rho'\sigma$*, or*
- $g = \neg g'$ *and it is not the case that* $\sigma \models g'$*, or*
- $g = g' \wedge g''$*, and* $\sigma \models g'$ *and* $\sigma \models g''$*.*

We feel free to write $\rho \neq \rho'$ *for* $\neg(\rho = \rho')$*, and* $g \vee g'$ *for* $\neg(\neg g \wedge \neg g')$*.*

We now define the syntax of usage automata. They are much alike finite state automata, but with a different input alphabet. Instead of plain symbols, the edges of usage automata have the form $\alpha(\boldsymbol{\rho}) : g$, where $\boldsymbol{\rho} \in (\mathsf{Res} \cup \mathsf{Var})^{|\alpha|}$ and $g \in G$. The formal definition is in Def. 2.

Definition 2. Usage automata

A usage automaton φ *is a 5-tuple* $\langle S, Q, q_0, F, E \rangle$*, where:*

- $S \subseteq \mathsf{Act} \times (\mathsf{Res} \cup \mathsf{Var})^*$ *is the input alphabet,*
- Q *is a finite set of states,*
- $q_0 \in Q \setminus F$ *is the start state,*
- $F \subset Q$ *is the set of final "offending" states,*
- $E \subseteq Q \times S \times G \times Q$ *is a finite set of edges, written* $q \xrightarrow{\alpha(\boldsymbol{\rho}) : g} q'$

Example 1. Let DIFF(k) be the usage policy stating that the action α (with arity 1) can be fired at most on k different resources, i.e.:

$$\mathrm{DIFF}(k) = \mathsf{Ev}^* \setminus \{ \eta_0 \alpha(r_0) \eta_1 \cdots \eta_k \alpha(r_k) \eta_{k+1} \mid \forall i \neq j \in 0..k : r_i \neq r_j \}$$

The usage automaton $\varphi_{\mathrm{DIFF}(k)}$ in Fig. 1 denotes the usage policy DIFF(k). □

Each usage automaton φ denotes a usage policy P, i.e. a set of traces $\eta \in P$ that obey the policy. To define the semantics of φ, we consider all the possible instantiations of its variables $var(\varphi)$ to actual resources. This operation yields a (possibly infinite) set of automata with finite states and possibly infinite edges.

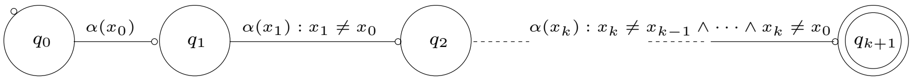

q_0
$\alpha(x_0)$
q_1
$\alpha(x_1) : x_1 \neq x_0$
q_2
$\alpha(x_k) : x_k \neq x_{k-1} \wedge \cdots \wedge x_k \neq x_0$
q_{k+1}

Fig. 1. The usage automaton $\varphi_{\mathrm{DIFF}(k)}$ models the policy DIFF(k)

The usage policy denoted by φ is then the union of the (complemented) languages recognized by the automata resulting from the instantiation of φ (the complement is needed because the final states of a usage automaton φ denote a violation of φ). We formally define the set of traces η that respect φ in Def. 3.

Definition 3. Policy compliance

Let $\varphi = \langle S, Q, q_0, F, E \rangle$ *be a usage automaton. For all* $R \subseteq \mathsf{Res}$ *and for all* $\sigma : var(\varphi) \rightarrow R$ *we define the automaton* $A_\varphi(\sigma, R) = \langle \Sigma, Q, q_0, F, \delta \rangle$ *as follows:*

$$\Sigma = \{\, \alpha(\boldsymbol{r}) \mid \alpha \in \mathsf{Act} \ \textit{and} \ \boldsymbol{r} \in R^{|\alpha|} \,\}$$

$$\delta = \textit{complete}(\{\, q \xrightarrow{\alpha(\boldsymbol{\rho}\sigma)} q' \mid q \overset{\alpha(\boldsymbol{\rho}):\, g}{\multimap} q' \in E \ \textit{and} \ \sigma \models g \,\}, R)$$

where $\textit{complete}(X, R)$ *is defined as follows:*

$$\textit{complete}(X, R) = X \cup \{\, q \xrightarrow{\alpha(\boldsymbol{r})} q \mid \boldsymbol{r} \in R^{|\alpha|} \ \textit{and} \ \nexists q' \in Q : q \xrightarrow{\alpha(\boldsymbol{r})} q' \in X \,\}$$

We say that η respects $A_\varphi(\sigma, R)$ *when* η *is* not *in the language of* $A_\varphi(\sigma, R)$*. In such a case, we write* $\eta \lhd A_\varphi(\sigma, R)$*.*

We say that η respects φ *when* $\eta \lhd A_\varphi(\sigma, \mathsf{Res})$ *for all* $\sigma : var(\varphi) \rightarrow \mathsf{Res}$*. In such a case, we write* $\eta \models \varphi$*; otherwise, we say* η violates φ*, and we write* $\eta \not\models \varphi$*.*

Example 2. Consider the implementation of a list datatype which allows for iterating over the elements of the list. In this scenario, a relevant usage policy is that preventing list modifications when the list is being iterated (this is actually implemented by Java collections through checks inserted in the source code). The usage automaton φ_{List} recognizing this policy is depicted in Fig. 2, left. The relevant events are $\textit{start}(l)$, for starting the iterator of the list l, $\textit{next}(l)$, for retrieving the next element of l, and $\textit{modify}(l)$, that models adding/removing elements from l. The automaton φ_{List} has three states: q_0, q_1, and *fail*. In the state q_0 both modifications of the list and iterations are possible. However, a modification (i.e. an add or a remove) done in q_0 will lead to q_1, where it is no longer possible to iterate on l, until a $\textit{start}(l)$ resets the situation. Let now: $\eta = \textit{start}(l_0)\, \textit{next}(l_0)\, \textit{start}(l_1)\, \textit{next}(l_1)\, \textit{modify}(l_1)\, \textit{modify}(l_0)\, \textit{next}(l_0)$. The trace η violates the policy, because the last event attempts to iterate on the list l_0, after l_0 has been modified. To check that φ_{List} correctly models this behaviour, consider the instantiation $A_0 = A_{\varphi_{\mathsf{List}}}(\{x \mapsto l_0\}, \{l_0, l_1\})$ depicted in Fig. 2, right. When supplied with the input η, the FSA A_0 reaches the offending state *fail*, therefore by Def. 3 it follows that $\eta \not\models \varphi_{\mathsf{List}}$. □

Our first result about usage automata is Lemma 1, the intuition behind which is more evident if the statement is read contrapositively. Let $\textit{res}(\eta)$ be the set of resources occurring in η. If η is recognized as offending by some instantiation $A_\varphi(\sigma, R')$ with $R' \supset \textit{res}(\eta)$, then η will be offending for *all* the instantiations $A_\varphi(\sigma, R)$ with $R \supseteq \textit{res}(\eta)$. In other words, the only relevant fact about the sets R used in instantiations is that R includes $\textit{res}(\eta)$.

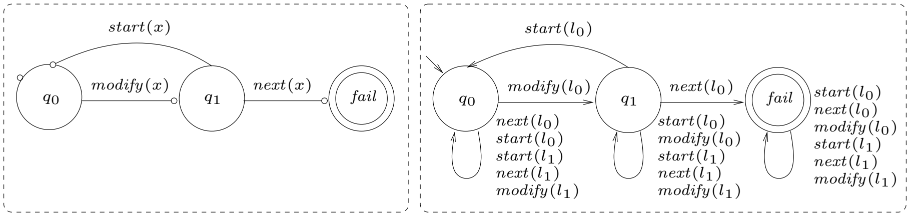


Fig. 2. The usage automaton φ_{List} (left) and the FSA $A_{\varphi_{\mathsf{List}}}(\{x \mapsto l_0\}, \{l_0, l_1\})$ (right)

Lemma 1. *For all usage automata φ, for all $\sigma : var(\varphi) \to \mathsf{Res}$, and traces η:*

$$\exists R \supseteq res(\eta) : \eta \lhd A_\varphi(\sigma, R) \implies \forall R' \supseteq res(\eta) : \eta \lhd A_\varphi(\sigma, R')$$

Proof. We prove the contrapositive. Assume there exists $R' \supseteq res(\eta)$ such that $\eta \ntriangleleft A_\varphi(\sigma, R')$, i.e. there is a run $q_0 \xrightarrow{\eta} q_k$ of $A_\varphi(\sigma, R')$ where $q_k \in F$. For all $R \supseteq res(\eta)$, we reconstruct an offending run of $A_\varphi(\sigma, R)$ on η. Let δ_R and $\delta_{R'}$ be the transition relations of $A_\varphi(\sigma, R)$ and $A_\varphi(\sigma, R')$, respectively, and let δ^*_R and $\delta^*_{R'}$ be their (labelled) reflexive and transitive closures. We prove by induction on the length of η that $\delta^*_{R'}(\{q_0\}, \eta) \subseteq \delta^*_R(\{q_0\}, \eta)$. The base case $\eta = \varepsilon$ is trivial. For the inductive case, note first that the transitions in the set $X = \{\, q \xrightarrow{\alpha(\boldsymbol{\rho}\sigma)} q' \mid q \xrightarrow{\alpha(\boldsymbol{\rho}) :\, g}\!\!\circ\; q' \in E \text{ and } \sigma \models g \,\}$ do not depend on R, R': indeed, R and R' only affect the self-loops added by *complete*. Let $\eta = \eta' \alpha(\boldsymbol{r})$. By the induction hypothesis, $\delta^*_{R'}(\{q_0\}, \eta') \subseteq \delta^*_R(\{q_0\}, \eta')$. Since $r \in res(\eta)$ and both $R \supseteq res(\eta)$ and $R' \supseteq res(\eta)$, it follows that $q \xrightarrow{\alpha(\boldsymbol{r})} q' \in complete(X, R)$ if and only if $q \xrightarrow{\alpha(\boldsymbol{r})} q' \in complete(X, R')$, which implies the thesis. □

Note that usage automata can be non-deterministic, in the sense that for all states q and input symbols $\alpha(\boldsymbol{\rho})$, the set of states:

$$\{\, q' \mid q \xrightarrow{\alpha(\boldsymbol{\rho}) :\, g}\!\!\circ\; q' \,\}$$

is not required to be a singleton. Given a trace η, we want that *all* the paths of the instances $A_\varphi(\sigma, R)$ comply with φ, i.e. they lead to a non-final state. Recall that we used final states to represent violations to the policy. An alternative approach would be the "default-deny" one, that allows for specifying the permitted usage patterns, instead of the denied ones. We could deal with this approach be regarding the final states as accepting. Both the default-deny and the default-allow approaches have known advantages and drawbacks. We think this form of diabolic non-determinism is particularly convenient when designing usage automata, since one can focus on the sequences of events that lead to violations, while neglecting those that do not affect the compliance to the policy. All the needed self-loops will be added by *complete*(X, R) in Def. 3.

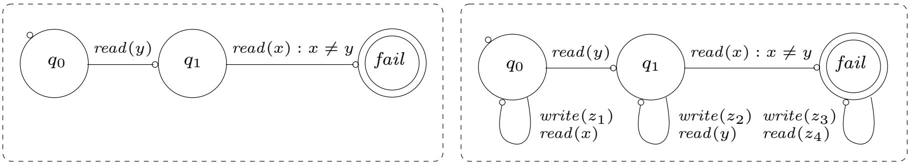


Fig. 3. A usage automaton (left), and a wrong attempt to expose its self-loops (right)

Example 3. Note that, in general, it is *not* possible to make all the self-loops explicit, so that $complete(X, R)$ adds no further self loops. Consider for instance the policy "a file x cannot be read if some other file $y \neq x$ has been read in the past". This is modelled by the usage automaton in Fig. 3, left, which recognizes e.g. the trace $read(f_0)write(f_0)read(f_1)$ as offending, and the trace $read(f_0)write(f_0)read(f_0)$ as permitted. The usage automaton depicted on the left has implicit self-loops: in order to correctly classify the above traces, it relies on $complete(X, R)$ in Def. 3, which adds the needed self-loops (e.g. those labelled $write(r)$ for all states and resources r). Any attempt to make all the self-loops explicit will fail, because it would require an unbounded number of variables in the edges whenever R is an infinite set. Consider e.g. the usage automaton in Fig. 3, right. For all the choices of σ, $complete(X, R)$ will add e.g. all the self-loops of q_0 labelled $write(r)$, for all $r \neq \sigma(z_1)$. □

According to Def. 3, to check $\eta \models \varphi$ we should instantiate a possibly *infinite* number of automata with finite states, one for each substitution $\sigma : var(\varphi) \to \mathsf{Res}$ (recall that Res can be an infinite set). Interestingly enough, the compliance of a trace with a usage automaton is a decidable property: the next lemma shows that checking $\eta \models \varphi$ can be decided through a *finite* set of finite state automata.

Lemma 2. *Let η be a trace, let φ be a usage automaton, let $res(\varphi)$ be the set of resources mentioned in the edges of φ (both in events and in guards), let $k = |var(\varphi)|$, and let $\#_1, \ldots, \#_k$ be arbitrary resources in $\mathsf{Res} \setminus (res(\eta) \cup res(\varphi))$, with $\#_i \neq \#_j$ for all $i \neq j$. Then, $\eta \models \varphi$ if and only if:*

$$\forall \sigma : var(\varphi) \to res(\eta) \cup res(\varphi) \cup \{\#_1, \ldots, \#_k\} \quad : \quad \eta \lhd A_\varphi(\sigma, res(\eta))$$

Proof. For the "only if" part, assume that $\eta \models \varphi$, i.e. that $\eta \lhd A_\varphi(\sigma', \mathsf{Res})$, for all $\sigma' : var(\varphi) \to \mathsf{Res}$. Since $res(\eta) \cup res(\varphi) \cup \{\#_1, \ldots, \#_k\} \subseteq \mathsf{Res}$, then it is also the case that $\eta \lhd A_\varphi(\sigma, \mathsf{Res})$. By Lemma 1, it follows that $\eta \lhd A_\varphi(\sigma, res(\eta))$.

For the "if" part, we prove the contrapositive. Assume that $\eta \not\models \varphi$, i.e. $\eta \not\lhd A_\varphi(\sigma', \mathsf{Res})$ for some $\sigma' : var(\varphi) \to \mathsf{Res}$. We prove that there exists some $\sigma : var(\varphi) \to res(\eta) \cup res(\varphi) \cup \{\#_1, \ldots, \#_k\}$ such that $\eta \not\lhd A_\varphi(\sigma, res(\eta))$. Let $var(\varphi) = \{x_1, \ldots, x_k\}$, and let $\sigma'(var(\varphi)) \setminus (res(\eta) \cup res(\varphi)) = \{r_1, \ldots, r_{k'}\}$, where $k' \leq k$ and $r_i \neq r_j$ for all $i \neq j$. Then, define σ as follows:

$$\sigma(x_j) = \begin{cases} \#_h & \text{if } \sigma'(x_j) = r_h \\ \sigma'(x_j) & \text{otherwise} \end{cases} \tag{1}$$

Let $\eta = \alpha_1(\boldsymbol{r}_1) \cdots \alpha_n(\boldsymbol{r}_n)$, and consider an offending run $q_0 \xrightarrow{\alpha_1(\boldsymbol{r}_1)} \cdots \xrightarrow{\alpha_n(\boldsymbol{r}_n)} q_n$ of $A_\varphi(\sigma', \mathsf{Res})$ on η. We now construct an offending run of $A_\varphi(\sigma, \mathit{res}(\eta))$ on η. Let $\delta_{A_\varphi(\sigma')} = \mathit{complete}(X', \mathsf{Res})$ and $\delta_{A_\varphi(\sigma)} = \mathit{complete}(X, \mathit{res}(\eta))$ be the transition sets of these automata, as given by Def. 3. We will show that, for all $i \in 1..n$:

$$\delta_{A_\varphi(\sigma')}(q_i, \alpha_i(\boldsymbol{r}_i)) \subseteq \delta_{A_\varphi(\sigma)}(q_i, \alpha_i(\boldsymbol{r}_i)) \tag{2}$$

Let $q' \in \delta_{A_\varphi(\sigma')}(q_i, \alpha_i(\boldsymbol{r}_i))$. Consider first the case that q' has not been added as a self-loop, i.e. $q' \in X'(q_i, \alpha_i(\boldsymbol{r}_i))$. Then, there exist $\boldsymbol{\rho}$ and g such that:

$$q_i \xrightarrow{\alpha_i(\boldsymbol{\rho})\colon g}\!\!\circ\; q' \in E \qquad \boldsymbol{\rho}\sigma' = \boldsymbol{r}_i \qquad \sigma' \models g$$

Since each component of $\boldsymbol{r}_i$ is in $\mathit{res}(\eta)$, then by (1) it follows that $\boldsymbol{\rho}\sigma = \boldsymbol{\rho}\sigma' = \boldsymbol{r}_i$. By induction on the structure of g, we prove that:

$$\sigma \models g \qquad \text{if and only if} \qquad \sigma' \models g \tag{3}$$

The base case $g = \mathit{true}$ is trivial. The other base case is when g has the form $\rho_0 = \rho_1$. One of the following four subcases hold.

- $\rho_0, \rho_1 \in \mathsf{Res}$. The thesis follows from $\rho_0\sigma = \rho_0 = \rho_0\sigma'$ and $\rho_1\sigma = \rho_1 = \rho_1\sigma'$.
- $\rho_0 \in \mathsf{Var}, \rho_1 \in \mathsf{Res}$. Then, $\sigma(\rho_1) = \rho_1 = \sigma'(\rho_1)$. Let $\rho_0 = x_a \in \mathit{var}(\varphi)$. By (1), we have that:

  $$\sigma(x_a) = \begin{cases} \#_{h_a} & \text{if } \sigma'(x_a) = r_{h_a} \qquad \text{(a1)} \\ \sigma'(x_a) & \text{otherwise} \qquad\qquad\; \text{(a2)} \end{cases}$$

  If $\sigma' \models \rho_0 = \rho_1$, then $\sigma'(x_a) = \rho_1$. We are in the case (a2), because $\rho_1 \in \mathit{res}(\varphi) \not\ni \#_{h_a}$. Therefore, $\sigma(x_a) = \sigma'(x_a) = \rho_1$, which proves $\sigma \models \rho_0 = \rho_1$.

  If $\sigma \models \rho_0 = \rho_1$, then $\sigma(x_a) = \rho_1$. Again, we are in the case (a2), because $\rho_1 \neq \#_{h_a}$. Therefore, $\sigma'(x_a) = \sigma(x_a) = \rho_1$, which proves $\sigma' \models \rho_0 = \rho_1$.
- $\rho_0 \in \mathsf{Res}, \rho_1 \in \mathsf{Var}$. Similar to the previous case.
- $\rho_0, \rho_1 \in \mathsf{Var}$. Assume $\rho_0 = x_a$ and $\rho_1 = x_b$, for some $a, b \in 1..k$ with $a \neq b$ (otherwise the thesis trivially follows). By (1), we have that:

  $$\sigma(x_a) = \begin{cases} \#_{h_a} & \text{if } \sigma'(x_a) = r_{h_a} \qquad \text{(a1)} \\ \sigma'(x_a) & \text{otherwise} \qquad\qquad\; \text{(a2)} \end{cases}$$

  $$\sigma(x_b) = \begin{cases} \#_{h_b} & \text{if } \sigma'(x_b) = r_{h_b} \qquad \text{(b1)} \\ \sigma'(x_b) & \text{otherwise} \qquad\qquad\; \text{(b2)} \end{cases}$$

  If $\sigma' \models \rho_0 = \rho_1$, then $\sigma'(x_a) = \sigma'(x_b)$. Let $r = \sigma'(x_a)$. If $r \in \mathit{res}(\eta) \cup \mathit{res}(\varphi)$, then we are in the case (a2,b2), and so $\sigma(x_a) = \sigma'(x_a) = \sigma'(x_b) = \sigma(x_b)$. Otherwise, if $r \notin \mathit{res}(\eta) \cup \mathit{res}(\varphi)$, then $r = r_{h_a} = r_{h_b}$. Since $r_1, \ldots, r_{k'}$ are pairwise distinct, then $h_a = h_b$, and so by (1), $\sigma(x_a) = \#_{h_a} = \#_{h_b} = \sigma(x_b)$. In both cases, we have proved that $\sigma \models \rho_0 = \rho_1$.

  Conversely, if $\sigma \models \rho_0 = \rho_1$, then $\sigma(x_a) = \sigma(x_b)$. Only the cases (a1,b1) and (a2,b2) are possible, because $\{\#_1, \ldots, \#_k\}$ is disjoint from $\mathit{res}(\eta) \cup \mathit{res}(\varphi)$. In the case (a1,b1), we have that $\sigma(x_a) = \#_{h_a} = \#_{h_b} = \sigma(x_b)$. Since $\#_1, \ldots, \#_k$ are pairwise distinct, then $h_a = h_b$, from which $\sigma'(x_a) = r_{h_a} = r_{h_b} = \sigma'(x_b)$. In the case (a2,b2), we have that $\sigma'(x_a) = \sigma(x_a) = \sigma(x_b) = \sigma'(x_b)$. In both cases, we have proved that $\sigma' \models \rho_0 = \rho_1$.

For the inductive case, the guard g can have either the form $\neg g_0$ or $g_0 \wedge g_1$, for some g_0 and g_1. If $g = \neg g_0$, then by the induction hypothesis of (3) it follows that $\sigma \models g_0$ if and only if $\sigma' \models g_0$, from which the thesis is straightforward. If $g = g_0 \wedge g_1$, the thesis follows directly from the induction hypothesis.

This concludes the proof of (3), which we have shown to imply $q' \in X'(q_i, \alpha_i(\boldsymbol{r}_i))$. Consider now the case $q' \in (\mathit{complete}(X', \mathsf{Res}) \setminus X')(q_i, \alpha_i(\boldsymbol{r}_i))$, i.e. $q' = q_i$ has been added as a self-loop. By Def. 3, we have that:

$$\nexists q \in Q \,:\, q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X'$$

We prove that $q' \in \delta_{A_\varphi(\sigma)}(q_i, \alpha_i(\boldsymbol{r}_i))$ by showing that the above implies:

$$\nexists q \in Q \,:\, q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X$$

We proceed contrapositively, i.e. we show:

$$\exists q \in Q \,:\, q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X \implies \exists q \in Q \,:\, q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X'$$

Assume that $q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X$. Then, there exist $\boldsymbol{\rho}$ and g such that:

$$q_i \xrightarrow{\alpha_i(\boldsymbol{\rho}):\, g} q \in E \qquad \boldsymbol{\rho}\sigma = \boldsymbol{r}_i \qquad \sigma \models g$$

Similarly to the previous case, this implies $\boldsymbol{\rho}\sigma' = \boldsymbol{r}_i$, and by (3) it also follows that $\sigma' \models g$. Therefore, $q_i \xrightarrow{\alpha_i(\boldsymbol{r}_i)} q \in X'$. Summing up, we have proved (2), from which the thesis directly follows. □

Following [16], we can characterize the set of policies recognized by our usage automata as a subset of safety properties. Usage automata recognize *properties*, because they define policies of the form $P = \{\, \eta \mid \eta \models \varphi \,\}$. They recognize *safety* properties because, by Def. 3, for all traces η and η', if $\eta\eta' \models \varphi$, then $\eta \models \varphi$.

We conclude this section by presenting some further relevant policies expressible through usage automata.

Example 4. Consider the policy requiring that, in a trace $\alpha(r_1)\alpha(r_2)\alpha(r_3)$, the three resources r_1, r_2, r_3 must be distinct. This is modelled by the usage automaton φ_3 in Fig. 4. A straightforward generalisation allows for keeping distinct k resources, by using a policy with arity $k - 1$.

Example 5. To justify the extra resources $\#_1, \ldots, \#_k$ used in Lemma 2, let φ be the usage automaton with a single edge $q_0 \xrightarrow{\alpha(x)\,:\, x \neq y} \mathit{fail}$. When a trace η

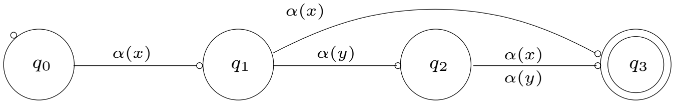


Fig. 4. The usage automaton φ_3 allows for keeping distinct three resources

contains some event $\alpha(r)$, there exists an instantiation $A_\varphi(\sigma, \mathit{res}(\eta))$ that recognizes η as offending, e.g. take $\sigma = \{x \mapsto r, y \mapsto r'\}$ for some $r' \neq r$. So, φ actually forbids any α actions. Consider now $\eta = \alpha(r_0)\,\beta(r_0)$. If Lemma 2 would not allow the range of σ to include any resource except r_0, then there would exist a single choice for σ, i.e. $\sigma_0 = \{x \mapsto r_0, y \mapsto r_0\}$. This would be wrong, because $\eta \lhd A_\varphi(\sigma_0, \{r_0\})$, while we know by Def. 3 that η violates φ. The resources $\#_1, \#_2$ allows for choosing e.g. $\sigma_1 = \{x \mapsto r_0, y \mapsto \#_1\}$, which yields $\eta \not\lhd A_\varphi(\sigma_1, \{r_0\})$, from which Lemma 2 correctly infers that $\eta \not\models \varphi$. □

Example 6. A classical security policy used in commercial and corporate business services is the Chinese Wall policy [7]. In such scenarios, it is usual to assume that all the objects which concern the same corporation are grouped together into a *company dataset*, e.g. $\mathit{bank}_A, \mathit{bank}_B, \mathit{oil}_A, \mathit{oil}_B$, etc. A *conflict of interest class* groups together all company datasets whose corporations are in competition, e.g. *Bank* containing $\mathit{bank}_A, \mathit{bank}_B$ and *Oil* containing $\mathit{oil}_A, \mathit{oil}_B$. The Chinese Wall policy then requires that accessing an object is only permitted in two cases. Either the object is in the same company dataset as an object already accessed, or the object belongs to a different conflict of interest class. E.g., the trace $\mathit{read}(\mathit{oil}_A, \mathit{Oil})\,\mathit{read}(\mathit{bank}_A, \mathit{Bank})\,\mathit{read}(\mathit{oil}_B, \mathit{Oil})$ violates the policy, because reading an object in the dataset oil_B is not permitted after having accessed oil_A, which is in the same conflict of interests class *Oil*. The Chinese Wall policy is specified by φ_{CW} in Fig. 5. The edge from q_0 to q_1 represents accessing the company dataset x in the conflict of interests class y. The edge leading from q_1 to the offending state q_2 means that a dataset x' different from x has been accessed in the same conflict of interests class y. □

Example 7. As a more substantial playground for studying the applicability of usage automata to express and enforce real-world policies, we consider a bulletin board system inspired by phpBB, a popular open-source Internet forum. A bulletin board consists of a set of *users* and a set of *forums*. Users can create new *topics* within forums. A topic represents a discussion, to be populated by *posts* inserted by users. Users may belong to three categories: guests, registered users, and, within these, moderators. We call regular users those who are not moderators. Each forum is tagged with a visibility level: `PUB` if everybody can read and write, `REG` if everybody can read and registered users can write, `REGH` if only registered users can read and write, `MOD` if everybody can read and moderators can write, `MODH` if only moderators can read and write. Additionally, we assume there exists a single administrator, which is the only user who can create new forums and delete them, set their visibility level, and lock/unlock them. Moderators can edit and delete the posts inserted by other users, can move topics

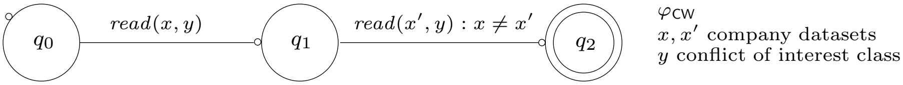


Fig. 5. A usage automaton φ_{CW} for the Chinese Wall policy

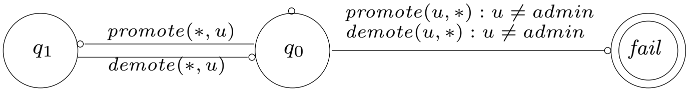


Fig. 6. Usage automaton for “only moderators can promote and demote other users”

through forums, can delete and lock/unlock topics. Both the administrator and the moderators can promote users to moderators, and vice-versa; the administrator cannot be demoted. Usage automata can be used to specify a substantial subset of the policies implemented (as local checks hard-wired in the code) by the server-side of phpBB, among which:

- only registered users can post to a `REG` or `REGH` forum; only moderators can post to a `MOD` or `MODH` forum.
- only registered users can browse a `REGH` forum; only moderators can browse a `MODH` forum.
- the session identifier used when interacting with the bulletin board must be the one obtained at login.
- a message can be edited/deleted by its author or by a moderator, only.
- a regular user cannot edit a message after it has been edited by a moderator.
- a regular user cannot delete a message after it has been replied.
- nobody can post on a locked topic.
- only moderators can promote/demote users and lock/unlock topics;
- only the administrator can lock/unlock forums.

Consider for instance the policy “only moderators can promote and demote other users”. This is modelled by the usage automaton in Fig. 6. The state q_0 models u being a regular user, while q_1, is for when u is a moderator. The first two transitions actually represent u being promoted and demoted. In the state q_0, u cannot promote/demote anybody, unless u is the administrator. For example, the trace $\eta = \mathit{promote}(\mathit{admin}, \mathit{alice})\mathit{promote}(\mathit{alice}, \mathit{bob})\,\mathit{demote}(\mathit{bob}, \mathit{alice})$ respects the policy, while $\eta\ \mathit{promote}(\mathit{alice}, \mathit{carl})$ violates it. Note that in Fig. 6 we use the wildcard $*$ to denote an arbitrary user. This has the same meaning of substituting a fresh variable for each occurrence of $*$. □

3 On the Expressive Power of Usage Automata

In the previous section we showed usage automata suitable for expressing a wide variety of policies, also from real-world scenarios. In this section we try to answer some interesting questions about their expressive power.

We start by investigating two restricted forms of usage automata. First, we consider usage automata without guards, and we show them less expressive than unrestricted ones. Then, we restrict the events $\alpha(\boldsymbol{\rho})$ on the edges of usage automata to be *monadic*, i.e. with $|\boldsymbol{\rho}| = 1$. Unlike in the previous case, this restriction does not affect the expressive power of usage automata.

Lemma 3. *Removing guards decreases the expressive power of usage automata.*

Proof. Recall from Ex. 1 the usage policy DIFF(1), preventing the action α to be fired on $k > 1$ distinct resource. We shall prove that there exists no usage automaton *without guards* that recognizes DIFF(1). By contradiction, assume that φ has no guards and recognizes DIFF(1). Let $\eta = \alpha(r)\alpha(r')$, where $r, r' \notin \mathit{res}(\varphi)$ and $r \neq r'$. Since $\eta \notin$ DIFF(1), by Lemma 2 it follows that $\eta \ntriangleleft A_\varphi(\sigma, \{r, r'\})$ for some $\sigma : \mathit{var}(\varphi) \to \{r, r'\} \cup \mathit{res}(\varphi) \cup \{\#_1, \ldots, \#_{|\mathit{var}(\varphi)|}\}$. Also, since $\alpha(r) \in$ DIFF(1), then $\alpha(r) \triangleleft A_\varphi(\sigma, \{r, r'\})$. Any run of $A_\varphi(\sigma, \{r, r'\})$ on η will then have the form:

$$q_0 \xrightarrow{\alpha(r)} q_1 \xrightarrow{\alpha(r')} q_2 \qquad \text{where } q_0, q_1 \notin F \text{ and } q_2 \in F$$

Since $q_1 \notin F$ and $q_2 \in F$, the transition from q_1 to q_2 cannot be a self-loop, and so there exists $q_1 \overset{\alpha(x)}{\multimap} q_2 \in E$ such that $\sigma(x) = r'$. Also, it must be $q_0 \neq q_1$. To show that, assume by contradiction that $q_0 = q_1$. Since $q_0 = q_1 \overset{\alpha(x)}{\multimap} q_2 \in E$, then the trace $\alpha(r)$ would violate φ, which cannot be the case by the hypothesis that φ recognizes DIFF(1). Thus, $q_0 \neq q_1$, and so there exists $q_0 \overset{\alpha(y)}{\multimap} q_1 \in E$ such that $\sigma(y) = r$. Consider now the trace $\eta' = \alpha(r)\alpha(r) \in$ DIFF(1). We have that $\eta' \ntriangleleft A_\varphi(\{x \mapsto r, y \mapsto r\}, \{r\})$, and so by Lemma 2 it follows that $\eta' \not\models \varphi$ – contradiction, because we assumed φ to recognize DIFF(1). □

We now consider the second restriction, i.e. only allowing for monadic events in the edges of usage automata. To show that this restriction does not affect their expressive power, we define a transformation that substitutes monadic events for polyadic ones in usage automata (Def. 4). We then show in Lemma 4 that the original and the transformed usage automata recognize the same policy. Of course, one needs to make monadic also the events in traces. To do that, we first define the set Act_1 of monadic actions as follows:

$$\mathsf{Act}_1 = \{\, \alpha^1, \ldots, \alpha^{|\alpha|} \mid \alpha \in \mathsf{Act} \,\}$$

Then, we define an injective transformation *slice* : $(\mathsf{Act} \times \mathsf{Res}^*)^* \to (\mathsf{Act}_1 \times \mathsf{Res})^*$ of traces as follows:

$$\mathit{slice}(\varepsilon) = \varepsilon \qquad \mathit{slice}(\eta\, \alpha(r_1, \ldots, r_k)) = \mathit{slice}(\eta)\, \alpha^1(r_1) \cdots \alpha^k(r_k)$$

In Def. 4 below we transform usage automata in order to make them recognize policies on traces of monadic events.

Definition 4. Slicing of polyadic events

Let $\varphi = \langle S, Q, q_0, F, E \rangle$ be a usage automaton. We define the usage automaton $\mathit{slice}(\varphi) = \langle S', Q', q_0, F, E' \rangle$ as follows:

$$S' = \{\, \alpha^1(\rho_1), \ldots, \alpha^k(\rho_k) \mid \alpha(\rho_1, \ldots, \rho_k) \in S \,\}$$

$$E' = \{\, q \overset{\alpha^1(\rho_1):\, g}{\multimap} q^1 \overset{\alpha^2(\rho_2)}{\multimap} \cdots q^{k-1} \overset{\alpha^k(\rho_k)}{\multimap} q' \mid q \overset{\alpha(\rho_1, \ldots, \rho_k):\, g}{\multimap} q' \in E \,\}$$

$$Q' = Q \cup \{\, q^i \mid q \overset{\alpha^i(\rho_i)}{\multimap} q^i \in E' \,\}$$

The following lemma states that polyadic events do not augment the expressive power of usage automata. Formally, it shows that making monadic the events of a usage automaton φ preserves the policy defined by φ, i.e. φ and $\mathit{slice}(\varphi)$ define the same policy (modulo the injective transformation of traces).

Lemma 4. *For all usage automata φ and for all traces η, $\eta \models \varphi$ if and only if $\mathit{slice}(\eta) \models \mathit{slice}(\varphi)$.*

Proof. Straightforward by induction on the length of η. □

We now turn our attention to the *arity* of usage automata, defined as $|\varphi| = |\mathit{var}(\varphi)|$. Unlike the arity of events, which we showed above unrelated to the expressive power, this is not the case for the arity of usage automata. Indeed, we will prove that each increment of the arity of usage automata corresponds to an increment of the usage policies they can recognize.

To show that, we exploit a proof technique provided by Lemma 5 below. Intuitively, it states that a usage automaton φ cannot distinguish among more than $|\varphi| + |\mathit{res}(\varphi)| + 1$ distinct resources. More formally, if $\eta \not\models \varphi$, then one can "deflate" to $|\varphi|+|\mathit{res}(\varphi)|+1$ the number of distinct resources in η, and still obtain a violation. Deflating to n the number of resources in η amounts to applying to η a substitution $\tau : \mathit{res}(\eta) \to \mathit{res}(\eta) \cup \{\#\}$ with $|\mathit{ran}(\tau)| = n$, where $\#$ is a distinguished resource not appearing in $\mathit{res}(\eta)$ and $\mathit{ran}(\tau) = \{\, x\tau \mid x \in \mathit{res}(\eta) \,\}$.

Lemma 5. *For all usage automata φ and for all traces η:*

$$(\forall \tau : \mathit{res}(\eta) \to \mathit{res}(\eta) \cup \{\#\} : |\mathit{ran}(\tau)| \leq |\varphi| + |\mathit{res}(\varphi)| + 1 \Longrightarrow \eta\tau \models \varphi) \Longrightarrow \eta \models \varphi$$

Proof. Let $n = |\varphi| + |\mathit{res}(\varphi)| + 1$. If $|\mathit{res}(\eta)| \leq n$, there is nothing to prove (it suffices to choose τ be the identity). Let then $|\mathit{res}(\eta)| = n' > n$. We proceed contrapositively. By contradiction, assume that $\eta \not\models \varphi$. We prove that there exists $\tau : \mathit{res}(\eta) \to \mathit{res}(\eta) \cup \{\#\}$ with $|\mathit{ran}(\tau)| \leq n$, such that $\eta\tau \not\models \varphi$. By Lemma 2, $\eta \not\models \varphi$ implies that there exists $\sigma : \mathit{var}(\varphi) \to \mathit{res}(\eta) \cup \mathit{res}(\varphi) \cup \{\#_1, \dots, \#_{|\varphi|}\}$ such that $\eta \ntrianglelefteq A_\varphi(\sigma)$. For all $r \in \mathit{res}(\eta)$, let τ be defined as follows:

$$\tau(r) = \begin{cases} r & \text{if } r \in \mathit{ran}(\sigma) \text{ or } r \in \mathit{res}(\varphi) \\ \# & \text{otherwise} \end{cases} \tag{4}$$

Note that $|\mathit{ran}(\tau)| \leq |\mathit{ran}(\sigma)| + |\mathit{res}(\varphi)| + 1 \leq |\mathit{var}(\varphi)| + |\mathit{res}(\varphi)| + 1 = n$. By induction on the length of η, we prove that:

$$\delta^*_{A_\varphi(\sigma)}(q_0, \eta) \subseteq \delta^*_{A_\varphi(\tau\circ\sigma)}(q_0, \eta\tau) \tag{5}$$

That is, each transition taken by $A_\varphi(\sigma)$ on η can be replayed on the trace $\eta\tau$. Since $\delta_{A_\varphi(\sigma)}(q_0, \eta) \in F$, from (5) it will follow that $\delta_{A_\varphi(\sigma)}(q_0, \eta\tau) \in F$, from which Lemma 2 implies the thesis $\eta\tau \not\models \varphi$.

We now prove (5). W.l.o.g. we assume φ has monadic events (see Lemma 4). The base case $\eta = \varepsilon$ is trivial. For the inductive case, let $\eta = \eta'\alpha(r)$. By the induction hypothesis, $\delta^*_{A_\varphi(\sigma)}(q_0, \eta') \subseteq \delta^*_{A_\varphi(\sigma)}(q_0, \eta'\tau)$. Let now $q \in \delta^*_{A_\varphi(\sigma)}(q_0, \eta')$, and let $q' \in \delta_{A_\varphi(\sigma)}(q, \alpha(r))$. Consider first the case that q' has not been added as a self-loop, i.e. $q' \in X(q, \alpha(r))$. Then, by Def. 3 it follows that:

$$\exists \rho, g : \qquad q \xrightarrow{\alpha(\rho) : g} q' \in E \qquad \rho\sigma = r \qquad \sigma \models g$$

To prove that $q' \in \delta_{A_\varphi(\sigma)}(q, \alpha(r\tau))$, just note that since $r = \rho\sigma$, then either $r \in \mathit{ran}(\sigma)$ or $r = \rho \in \mathit{res}(\varphi)$, and so by (4) it follows that $r\tau = r$.

Consider now the case $q' \in (\mathit{complete}(X, \mathit{res}(\eta)) \setminus X)(q, \alpha(r))$, i.e. $q' = q$ has been added as a self-loop. By Def. 3, we have that:

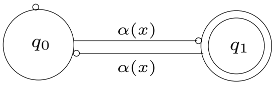
$$\nexists q'' \in Q \,:\, q \xrightarrow{\alpha(r)} q'' \in X$$

We prove that $q \in \delta_{A_\varphi(\sigma)}(q, \alpha(r\tau))$ by showing that the above implies:

$$\nexists q'' \in Q \,:\, q \xrightarrow{\alpha(r\tau)} q'' \in X$$

There are two cases, according to (4). If $r\tau = r$, then the thesis trivially follows. Otherwise, we have that $r\tau = \#$, and since # is a distinguished resource – in particular, different from any resource in $\mathit{ran}(\sigma)$ and $\mathit{res}(\varphi)$ – then $\alpha(\#)$ cannot be a label in X. Summing up, we have proved (5), which implies the thesis. □

Example 8. To justify the term "+1" in $|\mathit{ran}(\tau)| \leq |\varphi| + |\mathit{res}(\varphi)| + 1$, consider the usage automaton φ defined below, and the trace $\eta = \alpha(r_0)\alpha(r_1)$:



We have that $\varphi \not\models \eta$, e.g. $\eta \ntriangleleft A_\varphi(\sigma)$ with $\sigma = \{x \mapsto r_0\}$. Since $|\varphi| = 1$ and $\mathit{res}(\varphi) = 0$, Lemma 5 states that it is possible to obtain a violation by deflating η to 2 resources. Without the term "+1", we could have incorrectly deflated η to just 1 resource. Actually, all the possible "deflations" of η to only 1 resource – e.g. $\alpha(r_0)\alpha(r_0)$ and $\alpha(r_1)\alpha(r_1)$ – do obey φ, while $\eta \not\models \varphi$. □

We can now prove that the policy DIFF(k) of Ex. 1 cannot be expressed by a usage automaton with arity $k' < k$, in the case Res is an infinite set. By contradiction, assume that there exists φ with arity k' such that $\eta \models \varphi$ if and only if $\eta \in$ DIFF(k), for all η. W.l.o.g. we can assume that $\mathit{res}(\varphi) = \emptyset$. Let:

$$\eta_k = \alpha(r_0)\alpha(r_1) \cdots \alpha(r_k) \qquad \text{with } r_i \neq r_j \text{ for all } i \neq j \in 0..k \tag{6}$$

Since $k' < k$, we have that $\eta_k\tau \models \varphi$ for all $\tau : \mathit{res}(\eta) \to \mathit{res}(\eta) \cup \{\#\}$ such that $|\mathit{ran}(\tau)| \leq k' + 1$. By Lemma 5, this implies that $\eta_k \models \varphi$. But this is a contradiction, because by (6) it follows that $\eta_k \notin$ DIFF(k). Therefore, DIFF(k) cannot be expressed by any usage automaton with arity less then k.

4 Run-Time Enforcement of Usage Policies

In this section we discuss the feasibility of exploiting usage automata to implement run-time monitors for usage policies. The starting point is the observation that the specification of policy compliance of Def. 3 is not suitable for run-time monitors. Indeed, it requires the *whole* trace η, to decide whether $\eta \models \varphi$ or not. Since the trace of a running program will typically grow unbound, it is not feasible for a concrete run-time monitor to record it as a whole.

A realistic run-time monitor should only require a finite, bounded amount of memory to decide policy compliance. In particular, we aim at an execution monitor that does not need to record (any part of) the execution trace. Our monitor will just observe one event at a time, as it is fired at run-time, and use this event to update its (bounded) internal state.

We now define an enforcement mechanism for usage policies, that is suitable for run-time monitoring. Without loss of generality, we assume that events are monadic (see Lemma 4). The configurations of our mechanism have the form:

$$\mathcal{Q} = \{\sigma_0 \mapsto Q_0, \ldots, \sigma_k \mapsto Q_k\}$$

Intuitively, each $\sigma_i : \mathit{var}(\varphi) \to \mathsf{Res}$ represents a possible instantiation of φ into a finite state automaton $A_\varphi(\sigma_i)$. The set $Q_i \subseteq Q$ associated with σ_i represents the states reachable by the instantiation $A_\varphi(\sigma_i)$ upon the event trace seen so far. The number of instantiations recorded by the mechanism grows as new resources are discovered at run-time (yet, with the optimizations discussed below, it needs a bounded amount of space under the hypothesis that the number of resources "live" at any moment is bounded). The configuration $\mathcal{Q}$ of the mechanism is updated whenever an event is fired. The formal specification is in Def. 5.

Definition 5. Enforcement mechanism for usage automata

INPUT: a usage automaton $\varphi = \langle S, Q, q_0, F, E\rangle$ and a trace η.
OUTPUT: true if $\eta \models \varphi$, false otherwise.

1. *$\mathcal{Q} := \{\, \sigma \mapsto \{q_0\} \mid \sigma : \mathit{var}(\varphi) \to \{\#_1, \ldots, \#_p\}\,\}$, where $p = |\varphi|$ and $\#_1, \ldots, \#_p$ are arbitrary resources in $\mathsf{Res} \setminus (\mathit{res}(\eta) \cup \mathit{res}(\varphi))$.*
2. *while η is not empty, let $\eta = \alpha(r)\,\eta'$, do $\eta := \eta'$, and:*
   - *(a) if $r \notin \mathit{res}(\mathcal{Q})$, update $\mathcal{Q}$ as follows. For all σ occurring in $\mathcal{Q}$ and for all $\sigma' : \mathit{var}(\varphi) \to \mathit{res}(\mathcal{Q}) \cup \{r\}$ such that, for all $x \in \mathit{var}(\varphi)$, either $\sigma'(x) = \sigma(x)$ or $\sigma(x) = \#_j$:*

     $$\mathcal{Q} := \mathcal{Q}\,[\sigma' \mapsto \mathcal{Q}(\sigma)]$$

   - *(b) Let $\mathit{step}(q) = \{\, q' \mid q \in Q_i \wedge q \xrightarrow{\alpha(\rho):\, g} q' \in E \wedge \rho\sigma_i = r \wedge \sigma_i \models g \,\}$.*
     Let $\mathit{step}'(q) = (\text{if } \mathit{step}(q) = \emptyset \text{ then } \{q\} \text{ else } \mathit{step}(q))$.
     Then, for all $(\sigma_i \mapsto Q_i) \in \mathcal{Q}$, update $\mathcal{Q}$ as follows:

     $$\mathcal{Q} := \mathcal{Q}\,[\sigma_i \mapsto \textstyle\bigcup_{q \in Q_i} \mathit{step}'(q)]$$

3. *return true if, for all $(\sigma_i \mapsto Q_i) \in \mathcal{Q}$, $Q_i \cap F = \emptyset$. Otherwise, return false.*

The enforcement mechanism in Def. 5 actually defines a relation $\eta \vdash \varphi$, that holds whenever the above algorithm outputs *true* upon the inputs φ and η. The following lemma relates the "specification" of policy compliance in Def. 3 with its "implementation" provided by Def. 5. As expected, the two notions coincide.

Lemma 6. *For all traces η and usage automata φ, $\eta \models \varphi$ if and only if $\eta \vdash \varphi$.*

Proof. Let $\eta = \alpha_1(r_1) \cdots \alpha_n(r_n)$, and let $\mathcal{Q}_0 \xrightarrow{\alpha_1(r_1)} \cdots \xrightarrow{\alpha_n(r_n)} \mathcal{Q}_n$ be the sequence of configurations of the enforcement mechanisms upon input η, where $\mathcal{Q}_0$ is the initial configuration. By Lemma 2, $\eta \models \varphi$ if and only if $\eta \lhd A_\varphi(\sigma)$, for all $\sigma : \mathit{var}(\varphi) \to \mathit{res}(\eta) \cup \mathit{res}(\varphi) \cup \{\#_1, \ldots, \#_p\}$. Let η^i be the prefix of η containing exactly i events. We define the preorder $\leq$ as follows:

$$\begin{aligned} \sigma \leq \sigma' \quad \text{iff} \quad & \mathit{dom}(\sigma) = \mathit{dom}(\sigma') \\ \text{and} \quad & \forall x : \sigma(x) = \#_i \;\vee\; \sigma(x) = \sigma'(x) \\ \text{and} \quad & \forall x, y : \sigma(x) = \sigma(y) \iff \sigma'(x) = \sigma'(y) \end{aligned}$$

By induction on i, it is easy to prove that:

$$\sigma \leq \sigma \wedge \sigma \leq \sigma' \implies \mathcal{Q}_i(\sigma) = \mathcal{Q}_i(\sigma') \tag{7}$$

For all $i \in 0..n$, choose then σ^i be such that:

$$\sigma^i \in \max \{\, \sigma' \in \mathit{dom}(\mathcal{Q}_i) \mid \sigma' \leq \sigma \,\}$$

By (7) it follows that the image of $\mathcal{Q}_i$ on any maximum element of the set $\{\, \sigma' \in \mathit{dom}(\mathcal{Q}_i) \mid \sigma' \leq \sigma \,\}$ is the same. We shall then prove that, for all $i \in 0..n$:

$$\delta^*_{A_\varphi(\sigma)}(q_0, \eta^i) = \mathcal{Q}_i(\sigma^i) \tag{8}$$

The proof then proceeds by a straightforward induction on i. □

We now discuss some more concrete issues, building upon our experience with the implementation of a run-time monitor that enforces usage policies to Java programs [13]. A key issue is how to keep the size of the mechanism state small enough to make acceptable the overhead due to run-time monitoring. Some observations may help in achieving that.

First, observe that in real-world scenarios, the variables of usage automata will be typed, e.g. in the Chinese Wall policy of Ex. 6, the variables x, x' will have type "company dataset", while y will have type "conflict of interests class". Therefore, only those σ that respect types will need to be considered.

Second, the number of variables used in a policy can be reduced by means of *wildcards*. For instance, in [13] we used the wildcard $*$ to denote "any resource", and $-$ to denote "any resource different from the other resources mentioned in the policy". Since the number of possible instantiations grows exponentially in the number of variables, using wildcards may drastically reduce the size of the enforcement mechanism.

Third, some optimizations of the algorithm in Def. 5 – besides using smart data structures – are possible. For instance, there is no need to allocate in step

(a) a new instance $A_\varphi(\sigma)$, unless $A_\varphi(\sigma)$ can take a transition in step (b). Also, when a resource r is garbage-collected, we can discard all the instantiations $A_\varphi(\sigma)$ with $r \in \mathit{ran}(\sigma)$; it suffices to record in $\mathcal{Q}$ the states of $A_\varphi(\sigma)$ in a special entry $\sigma^\dagger \mapsto Q^\dagger$ for disposed objects.

5 Conclusions

We have studied a model for policies that control the usage of resources. Usage automata were originally introduced in [3]. The current work thoroughly improves and simplifies them. Usage automata can now have an arbitrary number of parameters, which augments their expressive power and allows for modelling significant real-world policies (see e.g. the Jalapa Project [13]). Also, the guards featured here allow for specifying policies that were not expressible through the negation operator ($\bar{x}$, meaning "any resource different from x") proposed in [3]. For instance, usage automata with guards can recognize DIFF(k) for all k, while those with negated variables cannot do that for $k > 1$.

The present work provides a deeper understanding of some foundational aspects of usage automata. From the point of view of expressivity, we have devised a proof technique (the "deflating lemma") through which showing that certain policies cannot be expressed by usage automata having less than a given number of variables. Also, we have proved that restricting the events to be monadic has no impact on the expressive power. From the point of view of implementability, we have proposed an enforcement mechanism to decide policy compliance. This mechanism only requires a bounded amount of memory.

Usage automata have been used in [2] to define a unified framework for the verification of history-based policies of programs. There, programs are terms in a λ-calculus enriched with primitives to create and use resources, and with a lexical mechanism to define the scope of usage policies (specified by usage automata). A type and effect analysis infers sound approximations of the set of run-time traces, and a model-checking algorithm verifies whether these traces comply with the policies at hand.

Related work. A lot of recent research is concerned with the study of usage policies and of their enforcement mechanisms.

A characterization of the class of policies enforceable through run-time monitoring systems is given in [16]. There, the policy language is that of *security automata*, a class of Büchi automata that recognize safety properties. To handle the case of parameters ranging over infinite domains (e.g. the formal parameters of a method), security automata resort to a countable set of states and input symbols. While this makes security automata suitable to provide a common semantic framework for policy languages (e.g. the semantics of our usage automata can be given in terms of a mapping into security automata) it also makes security automata hardly usable as an effective formalism for expressing policies. In [11] a characterization is given of the policies that can be enforced through program rewriting techniques. In [5] a kind of automata that can delete and insert events in the execution trace is considered. Polymer [6] is a language for specifying, composing and enforcing (global) security policies. In the lines of *edit automata* [5], a policy can intervene in the program trace, to insert or suppress some events. Policy composition can then be unsound, because the events inserted by a policy may interfere with those monitored by another policy. To cope with that, the programmer must explicitly fix the order in which policies are applied. The access control model of Java is enhanced in [15], by specifying fine-grained constraints on the execution of mobile code. A method invocation is denied when a certain condition on the dynamic state of the system is false. Security policies are modeled as process algebras in [14]. A custom Java Virtual Machine is used, with an execution monitor that traps system calls and fires them concurrently to the policy. When a trapped system call is not permitted by the policy, the execution monitor tries to force a corrective event – if possible – otherwise it aborts the system call. Unlike our usage automata, the models of [6,14,15] are Turing-equivalent, and so they are able to recognize more policies than ours. Yet, this flexibility comes at a cost. First, the process of deciding whether an action must be denied or not might loop forever. Second, non-trivial static optimizations are unfeasible, unlike in our approach. In particular, no static guarantee can be given about the compliance of a program with the imposed policy: run-time monitoring is then needed to enforce the policy, while our usage automata are model-checkable [4], and so may avoid this overhead.

Shallow history automata are investigated in [9]. These automata can keep track of the *set* of past access events, rather than the *sequence* of events. Although shallow history automata can express some interesting security properties, they are clearly less expressive than our usage automata.

In [12], policies are modelled as sets of permitted usage patterns, to be attached to resources upon creation. Our usage automata are not hard-wired to resources, yet they are *parametric* over resources. For instance, a usage automaton φ with variables x, y means that, for all the possible instantiations of x and y to actual resources, the obligation expressed by φ must be obeyed. This is particularly relevant in mobile code scenarios, where you need to impose constraints on how external programs access the resources created in your local environment, without being able to alter the code.

Acknowledgments. I thank Roberto Zunino for his insightful comments. Research partially supported by EU-FETPI Global Computing Project IST-2005-16004 Sensoria (Software Engineering for Service-Oriented Overlay Computers) and by PRIN Project SOFT (Tecniche Formali Orientate alla Sicurezza).

References

1. Abadi, M., Fournet, C.: Access control based on execution history. In: Proceedings of the 10th Annual Network and Distributed System Security Symposium, San Diego, California, USA, The Internet Society (2003)
2. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Local policies for resource usage analysis. To appear in ACM Tran. Programming Languages and Systems

3. Bartoletti, M., Degano, P., Ferrari, G.-L., Zunino, R.: Types and effects for resource usage analysis. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 32–47. Springer, Heidelberg (2007)
4. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Model checking usage policies. In: Proceedings of the 4th Trustworthy Global Computing, Barcelona, Spain. LNCS, vol. 5474, pp. 19–35. Springer, Heidelberg (2009)
5. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Proceedings of the Workshop on Foundations of Computer Security (FCS) (2002)
6. Bauer, L., Ligatti, J., Walker, D.: Composing security policies with Polymer. In: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI), Chicago, USA, pp. 305–314. ACM, New York (2005)
7. Brewer, D.F.C., Nash, M.J.: The Chinese Wall security policy. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy (1989)
8. Edjlali, G., Acharya, A., Chaudhary, V.: History-based access control for mobile code. In: Vitek, J. (ed.) Secure Internet Programming. LNCS, vol. 1603. Springer, Heidelberg (1999)
9. Fong, P.W.: Access control by tracking shallow execution history. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P 2004), Berkeley, CA, USA, May 9-12, pp. 43–55. IEEE Computer Society, Los Alamitos (2004)
10. Gong, L.: Inside Java 2 platform security: architecture, API design, and implementation. Addison-Wesley, Reading (1999)
11. Hamlen, K.W., Morrisett, J.G., Schneider, F.B.: Computability classes for enforcement mechanisms. ACM Trans. on Programming Languages and Systems 28(1), 175–205 (2006)
12. Igarashi, A., Kobayashi, N.: Resource usage analysis. In: Proceedings of the 29th Annual Symposium on Principles of Programming Languages (POPL), pp. 331–342. ACM, New York (2002)
13. Jalapa: Securing Java with Local Policies, `http://jalapa.sourceforge.net`
14. Martinelli, F., Mori, P.: Enhancing java security with history based access control. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 4677, pp. 135–159. Springer, Heidelberg (2007)
15. Pandey, R., Hashii, B.: Providing fine-grained access control for java programs. In: Guerraoui, R. (ed.) ECOOP 1999. LNCS, vol. 1628, pp. 449–473. Springer, Heidelberg (1999)
16. Schneider, F.B.: Enforceable security policies. ACM Trans. on Information and System Security 3(1), 30–50 (2000)
17. Skalka, C., Smith, S.: History effects and verification. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 107–128. Springer, Heidelberg (2004)

Static Detection of Logic Flaws in Service-Oriented Applications*

Chiara Bodei[1], Linda Brodo[2], and Roberto Bruni[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
{chiara,bruni}@di.unipi.it
[2] Dipartimento di Scienze dei Linguaggi, Università di Sassari, Italy
brodo@uniss.it

Abstract. Application or business logic, used in the development of services, has to do with the operations that define the application functionalities and not with the platform ones. Often security problems can be found at this level, because circumventing or misusing the required operations can lead to unexpected behaviour or to attacks, called *application logic attacks*. We investigate this issue, by using the CaSPiS calculus to model services, and by providing a Control Flow Analysis able to detect and prevent some possible misuses.

1 Introduction

More and more web surfers use applications based on web service technology for their transactions, such as bank operations or e-commerce purchases. The increasing availability of information exchange over e-services comes at the price of new security threats: new clever forms of attacks can come out at a rate that is growing with growth in usage.

Among the many different kinds of attacks that a malicious hacker can launch against web services applications, we here focus on the so-called *application logic attacks*, which are tailored to exploit the vulnerabilities of the specific functionalities of the application rather than the more general ones provided by the used platform, i.e. they violate application or business logic (see e.g., [11,19]). This logic represents the functions or the services that a particular site provides, in terms of the steps required to finalise a business goal, e.g. in an e-shop, the application logic can establish that customers' personal data necessary to complete an order must be provided only after the shopping basket is completed. Logic bugs in the application design may open dangerous loopholes that allow a user to do something that isn't allowed by the business, just by abusing or misusing the functions of the application, even without modifying them. Unfortunately, often security is not considered from the very beginning of the application development: the focus is on what the user is expected and allowed to do and not on the possible pathological usage scenarios that a goofy user may encounter

* Research supported by the EU FET-GC2 Project IST-2005-016004 Sensoria, by the Italian PRIN Project "SOFT" and by the Italian FIRB Project Tocai.it.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 70–87, 2009.

or a malicious user may exploit. For example, take a conference management system handling blind peer reviews: a malicious author could insert the names of “unfriendly” Program Committee members as co-authors of her/his paper to make them in conflict and exclude their opinions from the discussion phase. The fictitious co-authoring could then be removed in case the paper got accepted, without compromising the overall consistency of the conference management system and review process. Of course to prevent this misuse it could suffice to notify all authors about the submission as part of the application logic.

Logical vulnerabilities are subtle, application specific, and therefore difficult to detect. As usual, in the development of complex systems resorting to formal methods can be helpful. In particular, we try to transfer and adapt some techniques used in the field of network security (see e.g.,[8,10]). We develop a Control Flow Analysis for analysing the `close`-free fragment of CaSPiS [13,15], a process calculus recently introduced for modelling service oriented computing. The key features of CaSPiS are a disciplined, built-in management of long-running dyadic (and possibly nested) sessions between services and their clients, together with data-flow and orchestration primitives such as the pipeline operator and pattern-matching primitives that are suited, e.g., to deal with XML-like data typical of web service scenarios. The analysis statically approximates the behaviour of CaSPiS processes, in terms of the possible service and communication synchronisations. More precisely, what the analysis predicts encompasses everything that may happen, while what the analysis *does not* predict corresponds to something that cannot happen. The session mechanism is particularly valuable for the kind of analysis we use, because it guarantees that sibling sessions established between different instances of the same service and the corresponding clients do not interfere one with the other by leaking information, with two main consequences: first, our analysis can focus on each client-server conversation separately and second, we can focus on the application logic, neither having to commit on any specific implementation of sessioning over a certain platform nor worrying about the analysis of such a realisation.

The analysis we propose borrows some ideas from [7,20], because it exploits the similarity between the nesting hierarchies introduced by session primitives and the nesting hierarchies used in Ambients-like calculi. Furthermore to take care of malicious users, we modify the classical notion of Dolev-Yao attacker [17]: the attacker we are interested to model, that we call *malicious customer*, is an insider or, more precisely, an accredited user of a service that has no control of the communication channels, but that does not follow the intended rules of the application protocol, e.g., (s)he can cheat or introduce inconsistent data.

We apply our framework to an example inspired by a known logic-flawed application for e-commerce, where an abuse of functionality is possible, in which the attacker unexpectedly alters data, therefore modifying the application behaviour. The CyberOffice shopping cart [23] could be attacked by modifying the hidden field used for the price, within the web order form. The web page could be indeed downloaded normally, edited unexpectedly outside the browser and then submitted regularly with the prices set to any desired value, included zero or even a negative value. If no data-consistency control was performed by the server when the form was returned then the attack could be successful. Weak forms of validation could lead to similar problems, like when checking all item prices but not the total, or when checking goods price, but not the expedition costs. Intuitively, the information exchange between the customer C and the e-shop service S (and the data base DBI storing item prices) can be represented by the following informal protocol narration (steps from 1. to 4.), that we borrow from network security literature.

1. $C \rightarrow S: \quad ItemA$
2. $S \rightarrow DBI: Code, ItemA$
3. $DBI \rightarrow C: OrderForm(Code, ItemA, PriceA)$
4. $C \rightarrow S: \quad PaymentForm(Code, ItemA, PriceA, Name, Cc)$

...

Narration of the Protocol between Customer and E-shop Service

$4'. \quad C \rightarrow S: \quad PaymentForm'(Code, ItemA, FakedPriceA, Name, Cc)$

Attack on Fourth Step

The customer chooses an item $ItemA$, receives its price $PriceA$ inside an $OrderForm$ and can finalise the order by filling in a $PaymentForm$ with personal data like $Name$ and credit card information Cc. In the same form are reported: the transaction $Code$, the item and its price (for simplicity, we assume expedition expenses included). In case of pathological usage (step $4'$), the required information is added on a forged copy of the payment form, where the attacker has altered the price field, using the forged price $FakedPriceA$, instead of the one received from DBI within the $OrderForm$.

When modelled in CaSPiS, our analysis of the e-shop service is able to detect the possible price modification in harshly designed processes. The attack relies on the fact that S does not check that the third field of the received form has the correct value. This is because the application logic relies on step 4 to acquire personal data and credit card information of the customer that are considered as good enough credentials for establishing the "circle of trust" over the pending commercial transaction and it does not expect that misuses may still arise. Furthermore, to save on the number of exchanged messages, it delegates the DBI service (likely running on a separate, dedicated server) to communicate the price directly to the customer, so that S cannot perform any validation over it when the form is returned. A possible fix to the problem would consists in having the price information sent to S first and then redirected to the customer, so that it could be easy for the server to match the price included in $PaymentForm$ against the one received by DBI, as shown below.

$2''. \; S \rightarrow DBI: Code, ItemA$
$3''. \; DBI \rightarrow S: Code, ItemA, PriceA$
$4''. \quad S \rightarrow C: \quad OrderForm(Code, ItemA, PriceA)$
$5''. \quad C \rightarrow S: \quad PaymentForm(Code, ItemA, PriceA, Name, Cc)$

...

Alternative Narration

Related Work. Recently some works have faced the issue of security in the composition and verification of web applications. In [5] the authors propose security libraries, automatically analysed with ProVerif [6], using the WS-Security policies of [22]. WS ReliableMessaging is instead analysed in [24] with the AVISPA [3] verification toolkit, initially developed to check properties of cryptographic protocols. Recently, in [12], security properties have been considered in an extension of the π-calculus with session types and correspondence assertions [25]. In [1], behavioral types are used to statically approximate the behavior of processes. Building on [2,15], the type system [18] for CaSPiS is instead introduced to address the access control properties related to security levels. Safety properties of service behaviour is faced in [4]. Services can enforce security policies locally and can invoke other services respecting given security contracts. Formal reasoning about systems is obtained by means of dynamic and static semantics.

The focus of our interest is in formalising the logic used to develop an application and discovering its intrinsic weaknesses, due to a design practice that does not consider security as a first-class concern.

Plan of the Paper. In Section 2, we present the calculus. In Section 3, we introduce the Control Flow Analysis and the analysis of the malicious customer or attacker. In Section 4, we apply our framework to the above shown example of logic-flawed application for e-commerce. Section 5 concludes the paper. Proofs of theorems and lemmata presented throughout the paper are reported in [9].

2 The Calculus

We introduce here the fragment of CaSPiS that is sufficient to handle our modelling needs (i.e., without the constructs for handling session termination): it is essentially the one considered in [15]. Due to space limitation, we refer the interested reader to [13,14] for the motivation around CaSPiS design choices, its detailed description and many examples that witness its flexibility.

Syntax. Let $\mathcal{N} \ni n,\ n', ...$ be a countable set of names, that includes the set $\mathcal{N}_{srv} \ni s,\ s', ...$ of *service names* and the set $\mathcal{N}_{sess} \ni r,\ r', ...$ of *session names*, with $\mathcal{N}_{srv} \cap \mathcal{N}_{sess} = \emptyset$. We also let x range over variables (for service names and data). We distinguish here between *definition* occurrences and *use* occurrences of variables. A definition variable occurrence $?x$ is when x gets its binding value, while a use occurrence x is when the value has been bound. We assume that a variable cannot occur in both forms inside the same input, as in $(..., ?x, ..., x, ...)$. For the sake of simplicity, we let v range over values, i.e. names and use variables and $\tilde{v}$ on tuples. In the second part of the paper, we shall use a more general kind of input, including pattern matching. The syntax of CaSPiS is presented in Fig. 1, where the operators occur in decreasing order of precedence.

As usual, the empty summation is the nil process $\mathbf{0}$ (whose trailing is mostly omitted), parallel composition is denoted by $P|Q$ and restriction by $(\nu n)P$. The construct $r^p \rhd P$ indicates a generic session side with polarity p (taking values in $\{+, -\}$). Sessions are mostly intended as run-time syntax. In fact, differently

$P, Q ::=$		*processes*
	$\mid s.P$	service definition
	$\mid \overline{v}.P$	service invocation
	$\mid \Sigma_{i \in I} \pi_i P_i$	guarded sum
	$\mid r^p \rhd P$	session (considered as run-time syntax)
	$\mid P > (?\tilde{x})Q$	pipeline (written $P > \tilde{x} > Q$ in the literature)
	$\mid (\nu n)P$	restriction
	$\mid P \vert Q$	parallel composition
	$\mid !P$	replication
$p, q ::= + \vert -$		*polarities*
$\pi, \pi' ::=$		*action prefixes*
	$\mid (?\tilde{x})$	input
	$\mid \langle \tilde{v} \rangle$	output
	$\mid \langle \tilde{v} \rangle^{\uparrow}$	return

Fig. 1. Syntax of CaSPiS

from other languages that provide primitives for explicit session naming and creation, here all sessions are transparent to programmers as they can be built automatically, resulting in a more elegant and disciplined style of writing processes. A fresh session name r and two polarised session ends $r^- \rhd P$ and $r^+ \rhd Q$ are generated (on client and service sides, resp.) upon each service invocation $\overline{s}.P$ of the service $s.Q$. We say $r^- \rhd P$ is the dual session side of $r^+ \rhd Q$ and vice versa. As P and Q share a session, their I/O communications are directed toward the dual session side. We let p, q range over polarities and let $\overline{p}$ denote the opposite polarity of p, where $\overline{+} = -$ and $\overline{-} = +$. The prefix $\langle \tilde{v} \rangle^{\uparrow}$ is used to output values to the enclosing parent session and the pipe $P > (?\tilde{x})Q$ is a construct that spawns a fresh instance $Q[\tilde{v}/\tilde{x}]$ of Q on any value $\tilde{v}$ produced by P. Note that we use a slight modification of the pipeline $P > Q$ introduced in [13], similar to the variation considered in [15] and closer to Orc's *sequencing* [16]: the variables $\tilde{x}$ to be bound after pipeline synchronisation are included in a special input $(?\tilde{x})$, called *pipeline input* preceding the right branch process.

We assume processes are designed according to some typical well-formedness criteria: (i) the containment relation between session identifiers is acyclic (i.e. we cannot have processes like $r^p \rhd (P \vert r^p \rhd Q)$); (ii) for each session identifier r, each r^+ and r^- occurs once in the process and never in the scope of a dynamic operator; (iii) in any summation $\Sigma_i \pi_i$, all prefixes π_i are of one and the same kind (all outputs or all inputs or all returns).

Semantics. The *reduction semantics* of CaSPiS exploits a rather standard *structural congruence* $\equiv$ on processes, defined as the least congruence satisfying the clauses in Fig. 2. The binders for the calculus are $(\tilde{x}).P$ for $\tilde{x}$ in P and $(\nu n)P$ for n in P, with standard notions of free names (fn) and bound names (bn) of a process. Processes are considered equivalent up to the α-renaming of bound names. We present the operational semantics of the calculus by means of reduction contexts. The one-hole context $\mathbb{C}[\![\cdot]\!]$ is useful to insert a process P, the

- $(\mathcal{P}/_{\equiv}, |, \mathbf{0})$ is a commutative monoid;
- $!P \equiv P|\, !P$;
- $(\nu n)\mathbf{0} \equiv \mathbf{0}, \;\; (\nu n)(\nu n')P \equiv (\nu n')(\nu n)P, \;\; (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin \mathsf{fn}(P)$;
- $r^p \rhd (\nu n)P \equiv (\nu n)(r^p \rhd P)$ if $r \neq n$;
- $((\nu n)P) > (?x)Q \equiv (\nu n)(P > (?x)Q)$ if $n \notin \mathsf{fn}(Q)$;

Fig. 2. Structural Congruence Laws

result being denoted $\mathbb{C}[\![\, P \,]\!]$ (process P replaces the hole inside the context), into an arbitrary nesting of operators. Before describing the reduction rules, we need to fix some terminology. Let us call *dynamic operators* any service definition $s.[\![\cdot]\!]$, service invocation $\overline{s}.[\![\cdot]\!]$, prefix $\pi[\![\cdot]\!]$, left-sided pipeline $P > (?x)[\![\cdot]\!]$ and replication $![\![\cdot]\!]$. The remaining operators are called *static*. The contexts we are interested in are called *static*, and characterised by the fact that the hole occurs in an actively running position and it is ready to interact (e.g. it is not under a prefix): formally, we say that a context is *static* if its holes do not occur in the scope of a dynamic operator. Moreover, we say that a context is *session-immune* if its hole does not occur under a session operator, and *pipeline-immune* if its hole does not occur under a right-sided pipeline operator. In the following we let $\mathbb{C}[\![\cdot]\!]$ range over static contexts, $\mathbb{S}[\![\cdot]\!]$ over static session-immune contexts, and $\mathbb{P}[\![\cdot]\!]$ over contexts that are static, session-immune and pipeline-immune. Roughly, a static session-immune context $\mathbb{S}[\![\cdot]\!]$ can "intercept" concretion prefixes but not abstraction and return prefixes, while a static, session-immune and pipeline-immune context $\mathbb{P}[\![\cdot]\!]$ cannot "intercept" any prefix. Analogous definitions apply to the case of two-holes contexts $\mathbb{C}[\![\cdot,\cdot]\!]$. Below we let $\mathbb{C}_r[\![\cdot,\cdot]\!]$ be a context of the form $\mathbb{C}[\![\, r^p \rhd \mathbb{P}[\![\cdot]\!], r^{\overline{p}} \rhd \mathbb{S}[\![\cdot]\!] \,]\!]$ (for some $\mathbb{P}[\![\cdot]\!]$ and $\mathbb{S}[\![\cdot]\!]$), which helps us to characterise the most general situation in which intra-session communication can happen, and we write $\mathbb{S}_{r^p}[\![\cdot]\!]$ for a context of the form $r^p \rhd \mathbb{S}[\![\cdot]\!]$

The reduction rules of CaSPiS are given in Fig. 3, where we assume that r is fresh in Sync and that $|\tilde{x}| = |\tilde{v}|$. It can be shown that reductions preserve all well-formedness criteria mentioned above, in the sense that well-formed processes always reduce to well-formed processes.

Naming conventions. To distinguish among different occurrences of the same service, we assume to annotate each of them with a different index, as in $s_{@k}$. As a consequence, we can uniquely identify the fresh name used in case of synchronisation on the same service, e.g., when the synchronisation happens on the service s, on the occurrences $s_{@k}$ and $\overline{s}_{@m}$, then the session name is $r^p_{s@m:k}$. When unambiguous, we simply use s, $\overline{s}$ and r^p_s.

To distinguish among different pipeline constructs, we annotate each pipeline operator with a different label $l \in \mathcal{L}$, as in $>_l$. Also, we identify the left branch with a label l_0 and the right branch with l_1. The same annotation l_1 enriches the variables $\tilde{x}$ affected by the pipeline input in the right branch of the pipeline, as in $P >_l (?\tilde{x}^{l_1})Q$. Note that these annotations do not affect the semantics.

Furthermore, to simplify the definition of our Control Flow Analysis in Section 3, we discipline the α-renaming of *bound* values and variables. To do it in a simple and "implicit" way, we partition all the names used by a process into finitely

Sync	$\mathbb{C}[\![\,\overline{s}.P, s.Q\,]\!] \rightarrow (\nu r)\mathbb{C}[\![\, r^- \rhd P, r^+ \rhd Q\,]\!]$
S Sync	$\mathbb{C}_r[\![\,\langle\tilde{v}\rangle P + \sum_i \pi_i P_i, (?\tilde{x})Q + \sum_j \pi_j Q_j\,]\!] \rightarrow \mathbb{C}_r[\![\,P, Q[\tilde{v}/\tilde{x}]\,]\!]$
S Sync Ret	$\mathbb{C}_r[\![\,\mathbb{S}'_{r'^p}[\![\,\langle\tilde{v}\rangle^\uparrow P + \sum_i \pi_i P_i\,]\!], (?\tilde{x})Q + \sum_j \pi_j Q_j\,]\!] \rightarrow$ $\mathbb{C}_r[\![\,\mathbb{S}'_{r'^p}[\![\,P\,]\!], Q[\tilde{v}/\tilde{x}]\,]\!]$
P Sync	$\mathbb{C}[\![\,\mathbb{P}[\![\,\langle\tilde{v}\rangle P + \sum_i \pi_i P_i\,]\!] > (?\tilde{x})Q\,]\!] \rightarrow \mathbb{C}[\![\,Q[\tilde{v}/\tilde{x}] \mid (\mathbb{P}[\![\,P\,]\!] > (?\tilde{x})Q)\,]\!]$
P Sync Ret	$\mathbb{C}[\![\,\mathbb{P}[\![\,\mathbb{S}_{r^p}[\![\,\langle\tilde{v}\rangle P + \sum_i \pi_i P_i\,]\!]\,]\!] > (?\tilde{x})Q\,]\!] \rightarrow$ $\mathbb{C}[\![\,Q[\tilde{v}/\tilde{x}] \mid (\mathbb{P}[\![\,\mathbb{S}_{r^p}[\![\,P\,]\!]\,]\!] > (?\tilde{x})Q)\,]\!]$
Struct	$P \equiv P' \wedge P' \rightarrow Q' \wedge Q' \equiv Q \Rightarrow P \rightarrow Q$

Fig. 3. Reduction Semantics of CaSPiS

many equivalence classes and we use the names of the equivalence classes instead of the actual names. This partition works in such a way that names from the same equivalence class are assigned a common *canonical name* and consequently there are only finitely many canonical names in any execution of a given process. This is enforced by assigning the same canonical name to every name generated by the same restriction. The canonical name $\lfloor n \rfloor$ is for a name n; similarly $\lfloor x \rfloor$ is for a variable x. In this way, we statically maintain the identity of values and variables that might be lost by freely applying α-conversions. Hereafter, when unambiguous, we shall simply write n (resp. x) for $\lfloor n \rfloor$ (resp. $\lfloor x \rfloor$).

Examples. Consider the following simplified specification of a bank credit request service, where *req* is the service definition of the bank B.

$$\begin{aligned} B &\overset{def}{=} req_{@1}.(?y_{ba})\overline{val_{@3}}.\langle y_{ba}\rangle(?w_{ans})\langle w_{ans}\rangle^\uparrow \\ V &\overset{def}{=} val_{@4}.(?z_{ba})\langle Ans\rangle \end{aligned}$$

After the client invocation $\overline{req_{@1}}$, the bank waits for the client balance asset Ba (as input to $?y_{ba}$) that should be passed to the Validation service V, through the service invocation $\overline{val_{@3}}$. The validation answer Ans is sent to B (as input to $?w_{ans}$) and forwarded to the client. Thus a typical client C can be defined as

$$C \overset{def}{=} \overline{req_{@2}}.\langle Ba\rangle(?x_{ans})\langle x_{ans}\rangle^\uparrow$$

After the service invocation $\overline{req}$, the overall system $S \overset{def}{=} C|B|V$ becomes S',

$$\begin{aligned} S \rightarrow S' \overset{def}{=} (\nu r_{req@1:2})(\, & r^-_{req@1:2} \rhd \langle Ba\rangle(?x_{ans})\langle x_{ans}\rangle^\uparrow \mid \\ & r^+_{req@1:2} \rhd (?y_{ba})\overline{val_{@3}}.\langle y_{ba}\rangle(?w_{ans})\langle w_{ans}\rangle^\uparrow) \mid V \end{aligned}$$

where $r_{req@1:2}$ is the freshly generated session and where the client protocol is running on the left (the session side with negative polarity r^-) and the service protocol on the right (the session side with positive polarity r^+). The two protocols running on opposite sides of the same session can now exchange data, leading to S'':

$$\begin{aligned} S' \rightarrow S'' \overset{def}{=} (\nu r_{req@1:2})(\, & r^-_{req@1:2} \rhd (?x_{ans})\langle x_{ans}\rangle^\uparrow \mid \\ & r^+_{req@1:2} \rhd \overline{val_{@3}}.\langle Ba\rangle(?w_{ans})\langle w_{ans}\rangle^\uparrow) \mid V \end{aligned}$$

The computation continues as follows:

$$\begin{aligned}
S'' &\rightarrow (\nu r_{req@1:2})(r^{-}_{req@1:2} \triangleright (?x_{ans})\langle x_{ans}\rangle^{\uparrow} \mid \\
&\quad (\nu r_{val@3:4})(r^{+}_{req@1:2} \triangleright r^{-}_{val@3:4} \triangleright \langle Ba\rangle(?w_{ans})\langle w_{ans}\rangle^{\uparrow}) \mid r^{+}_{val@3:4} \triangleright (?z_{ba})\langle Ans\rangle)) \\
&\rightarrow (\nu r_{req@1:2})(r^{-}_{req@1:2} \triangleright (?x_{ans})\langle x_{ans}\rangle^{\uparrow} | \\
&\quad (\nu r_{val@3:4})(r^{+}_{req@1:2} \triangleright r^{-}_{val@3:4} \triangleright (?w_{ans})\langle w_{ans}\rangle^{\uparrow}) | r^{+}_{val@3:4} \triangleright \langle Ans\rangle)) \\
&\rightarrow (\nu r_{req@1:2})(r^{-}_{req@1:2} \triangleright (?x_{ans})\langle x_{ans}\rangle^{\uparrow} | \\
&\quad (\nu r_{val@3:4})(r^{+}_{req@1:2} \triangleright r^{-}_{val@3:4} \triangleright \langle Ans\rangle^{\uparrow}) | r^{+}_{val@3:4} \triangleright \mathbf{0})) \\
&\rightarrow (\nu r_{req@1:2})(r^{-}_{req@1:2} \triangleright \langle Ans\rangle^{\uparrow} | (\nu r_{val@3:4})(r^{+}_{req@1:2} \triangleright r^{-}_{val@3:4} \triangleright \mathbf{0} | r^{+}_{val@3:4} \triangleright \mathbf{0}))
\end{aligned}$$

Since the bank and validation service must typically handle more requests, one can use their replicated versions $!B$ and $!V$.

Now suppose several bank services $B_1, ..., B_n$ are available (together with suitable verification services $V_1, ..., V_m$, possibly shared by different banks), and that a client wants to contact them all and be notified by email about their answers Ans_j (with $j \in [1, m]$) exploiting a suitable service *email*. Then the client could be written as

$$(C_1 \mid \cdots \mid C_n) >_l (?x^{l_1}_{any-ans})\overline{email_{@0}}.\langle x_{any-ans}\rangle$$

where each C_i is the request to a specific bank service, i.e. it has the form $\overline{req_i}.\langle Ba\rangle(?x_i)\langle x_i\rangle^{\uparrow}$.

3 The Control Flow Analysis

We develop a Control Flow Analysis for the `close`-free fragment of CaSPiS, borrowing some ideas from [7,20]. Session primitives indeed introduce a nesting hierarchy that resembles the ones used in Ambients-like calculi. Our analysis uses the notion of enclosing scope, recording the current scope due to services, sessions or pipelines. We say that P is in the scope of $s_{@k}$ if $s_{@k}$ is the immediate enclosing service definition. Similarly for $\overline{s}_{@k}$, $r^{p}_{s@m:k}$, and for l_0 or l_1. The aim of the analysis is over-approximating all the possible behaviour of a CaSPiS process. In particular, our analysis keeps track of the possible contents of scopes, in terms of communication and service synchronizations. The result of analysing a process P is a pair $(\mathcal{I}, \mathcal{R})$, called *estimate* for P, that satisfies the judgements defined by the axioms and rules in the upper (lower, resp.) part of Table 1. The analysis is defined in the flavour of Flow Logic [21]. The first component $\mathcal{I}$ gives information on the contents of a scope. The second component $\mathcal{R}$ gives information about the set of values to which names can be bound. Moreover, let σ, σ' be scope identifiers, ranged over by $s_{@k}, \overline{s}_{@k}, r^{p}_{s@m:k}, l_0, l_1$.

To *validate* the correctness of a proposed estimate $(\mathcal{I}, \mathcal{R})$ we state a set of clauses operating upon judgements for analysing processes $\mathcal{I}, \mathcal{R} \models^{\sigma} P$. The judgement expresses that when P is enclosed within the scope identified by σ, then $(\mathcal{I}, \mathcal{R})$ correctly captures the behaviour of P, i.e. the estimate is valid also for all the states P', passed through a computation of P. More precisely:

- $\mathcal{I} : (\lfloor\mathcal{N}_{srv}\rfloor \cup \lfloor\mathcal{N}_{sess}\rfloor) \cup \mathcal{L} \rightarrow \wp(\lfloor\mathcal{N}_{srv}\rfloor \cup \lfloor\mathcal{N}_{sess}\rfloor \cup \mathcal{L}) \cup \lfloor\mathcal{P}ref\rfloor$, where $\lfloor S\rfloor$ is the set of canonical names in S, $\wp(S)$ stands for the power-set of the set S and $\lfloor\mathcal{P}ref\rfloor$ is the set of action and service prefixes, defined on canonical names. Here, $\sigma \in \mathcal{I}(\sigma')$ means that the scope identified by σ' may contain the one identified by σ; $\pi \in \mathcal{I}(\sigma')$ means that the action π may occur in the scope identified by σ'.
- $\mathcal{R} : \lfloor\mathcal{N}\rfloor \rightarrow \wp(\lfloor\mathcal{N}\rfloor)$ is the *abstract environment* that maps a variable to the set of names it can be bound to, i.e. if $v \in \mathcal{R}(n)$ then n may take the value v. We assume that for each free name n, we have that $n \in \mathcal{R}(n)$. Moreover, we write $\mathcal{R}(x_1, ..., x_n)$ as a shorthand for $\mathcal{R}(x_1), ..., \mathcal{R}(x_n)$. Without loss of generality, we suppose to have all the variables distinct.

Validation. Following [20], the analysis is specified in two phases. First, we check that $(\mathcal{I}, \mathcal{R})$ describes the initial process. This is done in the upper part of Table 1, where the clauses amount to a structural traversal of process syntax.

The clause for service definition checks that whenever a service $s_{@k}$ is defined in $s_{@k}.P$, then the relative hierarchy position w.r.t. the enclosing scope must be reflected in $\mathcal{I}$, i.e. $s_{@k} \in \mathcal{I}(\sigma)$. Furthermore, when inspecting the content P, the fact that the new enclosing scope is $s_{@k}$ is recorded, as reflected by the judgement $\mathcal{I}, \mathcal{R} \models^{s_{@k}} P$. Similarly for service invocation $\overline{x}_{@k}$: the only difference is that when x is a variable, the analysis checks for every actual value s that can be bound to x that $\overline{s}_{@k} \in \mathcal{I}(\sigma)$ and $\mathcal{I}, \mathcal{R} \models^{\overline{s}_{@k}} P$. The clauses for input, output and return check that the corresponding prefixes are included in $\mathcal{I}(\sigma)$ and that the analysis of the continuation processes hold as well. There is a special rule for pipeline input prefix, that allows us to distinguish it from the standard input one. Note that the current scope has the same identifier carried by the variables. Similarly, there is a rule for output prefixes occurring inside the scope of a left branch of a pipeline. The corresponding possible outputs are annotated with the label l_0. The rule for session, modelled as the ones for service, just checks that the the relative hierarchy position of the session identifier $r^p_{s@m:k}$ w.r.t. the enclosing scope must be reported in $\mathcal{I}$, i.e. $r^p_{s@m:k} \in \mathcal{I}(\sigma)$. It is used in analysing the possible continuations of the initial process.

All the clauses dealing with a compound process check that the analysis also holds for its immediate sub-processes. In particular, the analysis of $!P$ and that of $(\nu n)P$ are equal to the one of P. This is an obvious source of imprecision (in the sense of over-approximation). The clause for pipeline deserves a specific comment. It checks that whenever a pipeline $>_l$ is met, then the analysis of the left and the right branches is kept distinct by the introduction of two sub-indexes l_0 for the left one and l_1 for the right one. This allows us to predict possible communication over the two sides of the same pipeline. Furthermore, the analysis contents of the two scopes must be included in the enclosing scope identified by σ. This allows us to predict also the communications due to I/O synchronisations, involving prefixes occurring inside the scope of a pipeline.

In the second phase, we check that $(\mathcal{I}, \mathcal{R})$ also takes into account the dynamics of the process under analysis, i.e. the synchronizations due to communications, services and pipelines. This is expressed by the closure conditions in the lower part of Table 1 that mimic the semantics, by modelling, without exceeding the precision boundaries of the analysis, the semantic preconditions and the consequences of the possible actions. More precisely, preconditions check, in terms of $\mathcal{I}$, for the possible presence of the redexes necessary for actions to be performed. The conclusion imposes the additional requirements on $\mathcal{I}$ and $\mathcal{R}$, necessary to give a valid prediction of the analysed action. In the clause for *Service Synch*, we have to make sure that the precondition requirements hold, i.e.:

- there exists an occurrence of service definition: $s_{@k} \in \mathcal{I}(\sigma)$;
- there exists an occurrence of the corresponding invocation $\overline{s}_{@m} \in \mathcal{I}(\sigma')$;

If the precondition requirements are satisfied, then the conclusions of the clause express the consequences of performing the service synchronisation. In this case, we have that $\mathcal{I}$ must reflect that there may exist a session identified by $r^{+}_{s@m:k}$ inside σ and by $r^{-}_{s@m:k}$ inside σ' , such that the contents (scopes, prefixes) of $s_{@m:k}$ and of $\overline{s}_{@m}$ may also be inside $\mathcal{I}(r^{+}_{s@m:k})$ and $\mathcal{I}(r^{-}_{s@m:k})$, resp..

Similarly, in the clause for *I/O Synch*, if the following preconditions hold:

- there exists an occurrence of output in $\mathcal{I}(r^{p}_{s@m:k})$;
- there exists an occurrence of the corresponding input in the sibling session $\mathcal{I}(r^{\overline{p}}_{s@m:k})$.

then the values sent can be bound to the corresponding input variables: i.e., a possible communication is predicted here. Note that the rule correctly does not consider outputs in the form $\langle\tilde{v}\rangle^{l_0}$, because they possibly occur inside a left branch of a pipeline and therefore they are not available for I/O synchronisations.

The clause for *Ret Synch* is similar. The return prefix must be included in $r^{p}_{s@m:k}$, in turn included in $\mathcal{I}(r^{p'}_{s'@n:q})$, while the corresponding input must be included in $r^{\overline{p'}}_{s'@n:q}$, i.e. in the sibling session scope of the enclosing session.

In the clause for *Pipe I/O Synch*, the communication can be predicted, whenever the output and the pipeline input prefixes occur in the scope of the same session identifier (same side, too), and furthermore the output occurs in the left branch of a pipeline, while the pipeline input occurs in the right part of the same pipeline. Note that only pipeline input prefixes are considered here.

Similarly, in the clause for *Pipe Ret Synch*. The only difference is that the return prefix must occur in the session identified by $r^{p}_{s@m:k}$, included in the same scope that includes the corresponding pipeline input.

Example 1. Now, we can show how the analysis works on our running example:

$$\begin{aligned}
S &\overset{def}{=} B|V|C \\
B &\overset{def}{=} req_{@1}.(?y_{ba})\overline{val_{@3}}.\langle y_{ba}\rangle(?w_{ans})\langle w_{ans}\rangle^{\uparrow} \\
V &\overset{def}{=} val_{@4}.(?z_{ba})\langle Ans\rangle \\
C &\overset{def}{=} \overline{req_{@2}}.\langle Ba\rangle(?x_{ans})\langle x_{ans}\rangle^{\uparrow}
\end{aligned}$$

The main entries of the analysis are reported in Fig. 4, where $*$ identifies the ideal outermost scope in which the system top-level service scopes are. It is

Table 1. Analysis for CaSPiS Processes

$\mathcal{I},\mathcal{R} \models^{\sigma} s_{@k}.P$	iff $s_{@k} \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{s_{@k}} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} \overline{x}_{@k}.P$	iff $\forall s_{@m} \in \mathcal{R}(x) : \ \overline{s}_{@k} \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{\overline{s}_{@k}} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} (?\tilde{x}).P$	iff $(?\tilde{x}) \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{\sigma} P$
$\mathcal{I},\mathcal{R} \models^{l_1} (?\tilde{x}^{l_1}).P$	iff $(?\tilde{x}^{l}) \in \mathcal{I}(l_1) \ \wedge\ \mathcal{I},\mathcal{R} \models^{l_1} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} \langle\tilde{x}\rangle.P$	iff $\forall \tilde{v} \in \mathcal{R}(\tilde{x}) \ \langle\tilde{v}\rangle \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{\sigma} P$
$\mathcal{I},\mathcal{R} \models^{l_0} \langle\tilde{x}\rangle.P$	iff $\forall \tilde{v} \in \mathcal{R}(\tilde{x}) \ \langle\tilde{v}\rangle^{l_0} \in \mathcal{I}(l_0) \ \wedge\ \mathcal{I},\mathcal{R} \models^{l_0} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} \langle\tilde{x}\rangle^{\uparrow}.P$	iff $\forall \tilde{v} \in \mathcal{R}(x) \ \langle\tilde{v}\rangle^{\uparrow} \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{\sigma} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} \varSigma_{i\in I}\pi_i P_i$	iff $\forall i \in I : \mathcal{I},\mathcal{R} \models^{\sigma} \pi_i P_i$
$\mathcal{I},\mathcal{R} \models^{\sigma} P \mid Q$	iff $\mathcal{I},\mathcal{R} \models^{\sigma} P \ \wedge\ \mathcal{I},\mathcal{R} \models^{\sigma} Q$
$\mathcal{I},\mathcal{R} \models^{\sigma} !P$	iff $\mathcal{I},\mathcal{R} \models^{\sigma} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} (\nu n)P$	iff $\mathcal{I},\mathcal{R} \models^{\sigma} P$
$\mathcal{I},\mathcal{R} \models^{\sigma} P >_{l} (?\tilde{x}^{l_1})Q$	iff $l_0, l_1 \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{l_0} P \ \wedge\ \mathcal{I},\mathcal{R} \models^{l_1} (?\tilde{x}^{l})Q \ \wedge$ $\mathcal{I}(l_0), \mathcal{I}(l_1) \subseteq \mathcal{I}(\sigma)$
$\mathcal{I},\mathcal{R} \models^{\sigma} r^{p}_{s@m:k} \rhd P$	iff $r^{p}_{s@m:k} \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I},\mathcal{R} \models^{r^{p}_{s@m:k}} P$
(*Service Synch*)	$s_{@m} \in \mathcal{I}(\sigma) \wedge \overline{s}_{@k} \in \mathcal{I}(\sigma')$ $\Rightarrow r^{+}_{s@m:k} \in \mathcal{I}(\sigma) \ \wedge\ \mathcal{I}(s_{@m}) \subseteq \mathcal{I}(r^{+}_{s@m:k}) \ \wedge$ $r^{-}_{s@m:k} \in \mathcal{I}(\sigma') \ \wedge\ \mathcal{I}(\overline{s}_{@k}) \subseteq \mathcal{I}(r^{-}_{s@m:k})$
(*I/O Synch*)	$\langle\tilde{v}\rangle \in \mathcal{I}(r^{p}_{s@m:k}) \wedge (?\tilde{x}) \in \mathcal{I}(r^{\overline{p}}_{s@m:k})$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$
(*Ret Synch*)	$\langle\tilde{v}\rangle^{\uparrow} \in \mathcal{I}(r^{p}_{s@m:k}) \ \wedge\ r^{p}_{@m:k} \in \mathcal{I}(r^{p'}_{s'@n:q}) \wedge (?\tilde{x}) \in \mathcal{I}(r^{\overline{p'}}_{s'@n:q})$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$
(*Pipe I/O Synch*)	$\langle\tilde{v}\rangle^{l_0} \in \mathcal{I}(l_0) \ \wedge\ (?\tilde{x}^{l_1}) \in \mathcal{I}(l_1)$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$
(*Pipe Ret Synch*)	$\langle\tilde{v}\rangle^{\uparrow} \in \mathcal{I}(r^{p}_{s@m:k}) \ \wedge\ r^{p}_{s@m:k} \in \mathcal{I}(l_0)$ $\wedge\ (?\tilde{x}^{l_1}) \in \mathcal{I}(l_1)$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$

easy to check that $(\mathcal{I},\mathcal{R})$ is a valid estimate, i.e., that $\mathcal{I},\mathcal{R} \models^{*} S$, by following the two stages explained above: we just illustrate some steps. We have indeed that $\mathcal{I},\mathcal{R} \models^{*} S$ holds if $\mathcal{I},\mathcal{R} \models^{*} B$, $\mathcal{I},\mathcal{R} \models^{*} V$ and $\mathcal{I},\mathcal{R} \models^{*} C$. In particular, $\mathcal{I},\mathcal{R} \models^{*} A$ holds because $req_{@1} \in \mathcal{I}(*)$ and $\mathcal{I},\mathcal{R} \models^{*} C$ because $\overline{req_{@2}} \in \mathcal{I}(*)$. Now, by *(Service Synch)*, we have that $req_{@1}, \overline{req_{@2}} \in \mathcal{I}(*)$ implies that

$$\begin{array}{c} r^{+}_{req@1:2} \in \mathcal{I}(*) \wedge \mathcal{I}(req_{@1}) \subseteq \mathcal{I}(r^{+}_{req@1:2}) \wedge \\ r^{-}_{req@1:2} \in \mathcal{I}(*) \wedge \mathcal{I}(\overline{req_{@2}}) \subseteq \mathcal{I}(r^{-}_{req@1:2}) \end{array}$$

Furthermore, by *(I/O Synch)*, we have that $Ba \in \mathcal{R}(y_{ba})$, because

$$\langle Ba \rangle \in \mathcal{I}(r^{-}_{req@1:2}) \ \ (?y_{ba}) \in \mathcal{I}(r^{+}_{req@1:2})$$

Similarly, we can obtain that $\mathcal{R}(x_{any-ans})$ includes Ans_j, for each $j \in [1, m]$.

$$
\begin{array}{l}
\mathcal{I}(*) \ni req_{@1}, \overline{req_{@2}}, val_{@4} \\
\mathcal{I}(*) \ni r^{+}_{req@1:2}, r^{-}_{req@1:2}, r_{val@3:4} \\
\mathcal{I}(\overline{req_{@2}}), \mathcal{I}(r^{-}_{req@1:2}) \ni \langle Ba \rangle, (?x_{ans}), \langle x_{ans} \rangle^{\uparrow} \\
\mathcal{I}(req_{@1}), \mathcal{I}(r^{+}_{req@1:2}) \ni \overline{val_{@3}}, (?y_{ba}) \\
\mathcal{I}(\overline{val_{@3}}), \mathcal{I}(r^{-}_{val@3:4}) \ni \langle y_{ba} \rangle, (?w_{ba}), \langle w_{ba} \rangle^{\uparrow} \\
\mathcal{I}(val_{@4}), \mathcal{I}(r^{+}_{va@3:4}) \ni (?z_{ba}), \langle Ans \rangle \\
\mathcal{R}(y_{ba}) \ni Ba,\ \mathcal{R}(z_{ba}) \ni Ba \\
\mathcal{R}(w_{ans}) \ni Ans,\ \mathcal{R}(x_{ans}) \ni Ans
\end{array}
$$

Fig. 4. Some Entries of the Example Analysis

Semantic Correctness. Our analysis is correct w.r.t. the given semantics, i.e. a valid estimate enjoys the following subject reduction property.

Theorem 1. ***(Subject Reduction)***
If $P \rightarrow Q$ and $\mathcal{I}, \mathcal{R} \models^{\sigma} P$ then also $\mathcal{I}, \mathcal{R} \models^{\sigma} Q$.

This result depends on the fact that the analysis is invariant under the structural congruence, as stated below.

Lemma 1. ***(Invariance of Structural Congruence)*** *If $P \equiv Q$ and $\mathcal{I}, \mathcal{R} \models^{\sigma} P$ then also $\mathcal{I}, \mathcal{R} \models^{\sigma} Q$.*

The above results are handled and proved in the extended version of the calculus [9]. Currently, our analysis has not been implemented yet, but it could, e.g., along the lines of the one in [20].

Modelling the Malicious Customer. We need to model a new kind of attacker, different from the classical Dolev-Yao one [17], on which rely many systems for the verification of security protocols. A Dolev-Yao attacker acts on an open network, where principals exchange their messages. He/she can intercept and generate messages, provided that the necessary information is in his/her knowledge, which increases while interacting with the network. Our setting calls for a different model, because our attacker is an *accredited customer of a service* that has no control of the communication channels, apart from the ones established by the sessions in which he/she is involved. Nevertheless, our attacker, that we call *malicious customer* or **M**, does not necessarily follow the intended rules of the application protocol and can try to use the functions of the service in an unintended way, e.g., by sending messages in the right format, but with contents different from the expected ones. Under this regard, our attacker is less powerful of the Dolev-Yao one: **M** just tries to do everything that the application does not prevent him/her to do. More precisely, **M** has a knowledge made of all the public information and increased by the messages received from the service: the attacker can use this knowledge to produce messages to be sent to the server. We assume **M** is smart enough to send only messages in the expected format in each information exchange. We extend our framework in order to implicitly

Table 2. Analysis for CaSPiS Processes in the Presence of a Malicious Customer

$\mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} (?\tilde{x})^M.P$	iff $(?\tilde{x})^M \in \mathcal{I}(\sigma) \ \wedge \ \mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} P$
$\mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} (?\tilde{x}^l)^M.P$	iff $(?\tilde{x}^l)^M \in \mathcal{I}(\sigma) \ \wedge \ \mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} P$
$\mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} \langle \tilde{x} \rangle^M.P$	iff $\forall \tilde{v} \in \mathcal{R}(\tilde{x}) \ \langle \tilde{v} \rangle^M \in \mathcal{I}(\sigma) \ \wedge \ \mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} P$
$\mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} \langle \tilde{x} \rangle^{\uparrow M}.P$	iff $\forall \tilde{v} \in \mathcal{R}(\tilde{x}) \ \langle \tilde{v} \rangle^{\uparrow M} \in \mathcal{I}(\sigma) \ \wedge \ \mathcal{I}, \mathcal{R}, \mathcal{K} \models^{\sigma} P$
(*I/O Synch*)	$\langle \tilde{v} \rangle^a \in \mathcal{I}(r^p_{s@m:k}) \wedge (?\tilde{x})^b \in \mathcal{I}(r^{\overline{p}}_{s@m:k})$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$ $a = M \Rightarrow \forall \widetilde{v'} \in \mathcal{K} : \widetilde{v'} \in \mathcal{R}(\tilde{x})$ $b = M \Rightarrow \tilde{v} \in \mathcal{K}$
(*Pipe I/O Synch*)	$\langle \tilde{v} \rangle^{\uparrow a} \in \mathcal{I}(l_0) \ \wedge \ (?\tilde{x}^{l_1 b}) \in \mathcal{I}(l_1)$ $\Rightarrow \tilde{v} \in \mathcal{R}(\tilde{x})$ $a = M \Rightarrow \forall \widetilde{v'} \in \mathcal{K} : \widetilde{v'} \in \mathcal{R}(\tilde{x}^{l_1})$ $b = M \Rightarrow \tilde{v} \in \mathcal{K}$
(*Knowledge Rule* 1)	$v \in \mathcal{N}_{\mathsf{fn}} \Rightarrow v \in \mathcal{K}$
(*Knowledge Rule* 2)	$v_1, ..., v_k \in \mathcal{K} \Rightarrow \langle v_1, ..., v_k \rangle \in \mathcal{K}$
(*Knowledge Rule* 3)	$\langle v_1, ..., v_k \rangle \in \mathcal{K} \Rightarrow v_1, ..., v_k \in \mathcal{K}$

consider the possible behaviour of such an attacker or *malicious customer*. We statically approximate the malicious customer knowledge, by representing it by a new analysis component $\mathcal{K}$. Intuitively, the clauses acting on $\mathcal{K}$ implicitly take the attacker possible actions into account. The component $\mathcal{K}$ contains all the free names, all the messages that the customer can receive, and all the messages that can be computed from them, e.g. if v and v' belong to $\mathcal{K}$, then also the tuple (v, v') belongs to $\mathcal{K}$ and, vice versa, if (v, v') belongs to $\mathcal{K}$, then also v and v' belong to $\mathcal{K}$. Furthermore, all the messages in $\mathcal{K}$ can be sent by the customer. To distinguish the customer actions, we annotate the corresponding prefixes with M, as in π^M, and we use a and b both for M or the empty annotation ϵ. As a consequence, we need to slightly change the rules in Table 1, as shown in Table 2, where only the more significant rules are reported. Consider the rule (*I/O Synch*). Whenever the analysis predicts that a customer may send a certain message $\tilde{v}$, the analysis predicts that the same, possibly malicious, customer can also send every other message $\widetilde{v'}$, obtained by synthetising the information in $\mathcal{K}$, provided that it is in the same format of $\tilde{v}$. Whenever a customer may receive an input, then the analysis predicts that the same, possibly malicious, customer can also acquire the received message in $\mathcal{K}$.

Analysis at Work. In the following, we refer to the version of CaSPiS that includes pattern matching into the input construct, as defined by the following syntax (for a similar treatment see also [8]).

π, π' ::=	*Prefixes*	**E** ::=	*Terms*	**D** ::=	*Definition Terms*
$(D_1, ..., D_k)$	input	n	names	E	terms
$\langle E_1, ..., E_k \rangle$	output	x	variables	$?x$	definition variables
$\langle E_1, ..., E_k \rangle^{\uparrow}$	return				

Our patterns are tuples of definition terms $(D_1, \cdots, D_k)$ that have to be matched against tuples of terms $(E_1, \cdots, E_k)$, upon input. Note that, at run-time, each E_i is a closed term. Intuitively, the matching succeeds when the closed terms, say D_i, elementwise match to the corresponding terms E_i, and its effect is to bind the remaining terms E_j to the remaining variables. Actually, definition terms in $(D_1, \ldots, D_k)$ can be partitioned into closed terms to be matched and definition variables to be bound. We make the partition above explicit, by using the auxiliary functions $\mathbf{Term}(D_1, \ldots, D_k)$ and $\mathbf{Var}(D_1, \ldots, D_k)$. They work on the position of definition terms within the tuples in such a way that if $i \in \mathbf{Term}(D_1, \ldots, D_k)$, then D_i is a closed term, while if $i \in \mathbf{Var}(D_1, \ldots, D_k)$, then D_i is a definition variable. The new semantic rules for message exchange take pattern matching into account, e.g., the *(S Sync)* rule becomes:

$$\begin{array}{l} \mathbb{C}_r[\![\, \langle E_1, ..., E_k\rangle P + \sum_i \pi_i P_i, (D_1, ..., D_k)Q + \sum_j \pi_j Q_j \,]\!] \;\rightarrow\; \mathbb{C}_r[\![\, P, Q[\tilde{E}/\tilde{D}] \,]\!] \\ \text{if} \;\; \wedge_{i \in \mathbf{Term}(D_1,...,D_k)} \; E_i = D_i \end{array}$$

Suppose to have the following process

$$(\nu r)(r^- \rhd \langle A, M_A\rangle.P)|(r^+ \rhd (A, ?y_B).Q)$$

In the input tuple (A, y_B), we have that $\mathbf{Term}(A, ?y_B) = 1$ and $\mathbf{Var}(A, ?y_B) = 2$. Here the synchronisation succeeds, because the matching on the first term A succeeds. As a consequence, the second term y_B is bound to M_A in the continuation process $(\nu r)(r^- \rhd P)|(r^+ \rhd Q[M_A/y_B])$.

Extending the presented analysis in order to capture this kind of input construct is quite standard (see e.g. [8,10]). We recall the new CFA rules in Table 3, where we use $\mathcal{R}(D)$ as a shorthand for $\mathcal{R}(x)$ when $D = ?x$. For the sake of space, we only report some of them.

Table 3. Analysis for CaSPiS Processes

$\mathcal{I}, \mathcal{R} \models^\sigma (D_1, ..., D_k).P$ iff $(D_1, ..., D_k) \in \mathcal{I}(\sigma) \;\wedge\; \mathcal{I}, \mathcal{R} \models^\sigma P$ $\mathcal{I}, \mathcal{R} \models^\sigma \langle E_1, ..., E_k\rangle.P$ iff $\forall v_1, ..v_k : v_i \in R(E_i) \; \langle v_1, ..v_k\rangle \in \mathcal{I}(\sigma) \;\wedge\; \mathcal{I}, \mathcal{R} \models^\sigma P$
$(I/O\ Synch) \;\; \langle v_1, ..v_k\rangle \in \mathcal{I}(r^p_{s@m:k}) \wedge (D_1, ..., D_k) \in \mathcal{I}(r^{\overline{p}}_{s@m:k})$ $\wedge \bigwedge_{i \in \mathbf{Term}(D_1,...,D_k)} \; v_i = D_i$ $\Rightarrow \wedge_{j \in \mathbf{Var}(D_1,...,D_k)} \; v_j \in \mathcal{R}(D_j)$

We assume that, in each information exchange, a malicious customer sends messages in the expected format and successful w.r.t. the required pattern matching, as can stated by the rule (I/O $Synch$) in the presence of the Malicious Customer, modified to take the pattern matching into account.

$$
\begin{aligned}
(I/O\ Synch)\quad & \langle v_1, ..v_k\rangle^a \in \mathcal{I}(r^p_{s@m:k}) \wedge (D_1, ..., D_k)^b \in \mathcal{I}(r^{\overline{p}}_{s@m:k})\\
& \wedge \bigwedge_{i\in \mathbf{Term}(D_1,\dots,D_k)} v_i = D_i\\
& \Rightarrow \wedge_{j\in \mathbf{Var}(D_1,\dots,D_k)} v_j \in \mathcal{R}(D_j)\\
& ...\\
& a = M \Rightarrow \forall \langle v'_1, ..v'_k\rangle^a \in \mathcal{K} :\\
& \wedge \bigwedge_{i\in \mathbf{Term}(D_1,\dots,D_k)} v'_i = D_i\\
& \Rightarrow \wedge_{j\in \mathbf{Var}(D_1,\dots,D_k)} v'_j \in \mathcal{R}(D_j)
\end{aligned}
$$

Price Modification Example. We are now ready to model and analyse the example informally introduced in Section 1. The global system is composed by two processes put in parallel, the e-shop service S and the customer C. Also a data base DBI storing item prices is modeled:

$$(S \mid DBI) \mid C$$

Essentially, the e-shop S allows costumers to choose among several items $item_i$; when the costumer returns its selection, S asks the DBI service for the price of the selected item. In the first specification, shown below, the service S does not check if the form sent by the costumer contains the right price, i.e. the one computed by the DBI service, because the DBI sends the price directly to the client, without sending it also to the bank. For the sake of readability, we distinguish the part of the output tuples relative to the order form (payment form, resp.) by writing: $order\_form(...)$ ($payment\_form(...)$, resp.).

$$
\begin{aligned}
S \quad & = !selling.\ \textstyle\sum_i ((item_i)(\nu\ code)\\
& \quad (\overline{price\_db}\ \langle item_i\rangle\ (?x_{price_i})\langle order\_form(code, item_i, x_{price_i})\rangle^{\uparrow}\\
& \quad \mid\\
& \quad\ (ok, payment\_form(code, item_i, ?y_{price_i}, ?y_{name}, ?y_{cc})).PAY\ +\\
& \quad\ (no\_payment)))\\
DBI & = !price\_db\ \textstyle\sum_i ((item_i)\langle price_i\rangle)\\
\\
C \quad & = \overline{selling}.\ \langle item_i\rangle^M (order\_form(?z\_code, item_i, ?z_{price_i}))^M\\
& \qquad\qquad \langle ok, payment\_form(z\_code, item_i, z_{price_i}, name, cc)\rangle^M +\\
& \qquad\qquad \langle no\_payment\rangle^M
\end{aligned}
$$

In the second formulation, the service S' matches the price sent by the customer against the one provided by the DBI. The other processes are unmodified.

$$
\begin{aligned}
S' = !selling.\ \textstyle\sum_i ((item_i)(\nu\ code)\overline{price\_db}.\ \langle item_i\rangle\ (?x_{price_i})\langle x_{price_i}\rangle^{\uparrow}\ >_l\\
(?y^{l_1}_{price_i})\ \langle form(code, item_i, y_{price_i})\rangle\\
(ok, payment\_form(code, item_i, y_{price_i}, ?y_{name}, ?y_{cc}))) +\\
(no\_payment)
\end{aligned}
$$

The main entries of the analysis of the first formulation are reported in Fig 5. The variable $?y_{price}$, used by S, may be bound to any value the costumer sends, in particular to any possible faked price value. This depends on the fact that there is no pattern matching on the values received; more generally, no control on this part of input is made. Furthermore, note that since

$$\begin{array}{ll}
\mathcal{I}(selling), \mathcal{I}(r^{+}_{sell}) & \ni item_i, \overline{price\_db} \\
\mathcal{I}(\overline{price\_db}), I(r^{-}_{price}) & \ni \langle item_i \rangle, (?x_{price_i}), \langle order\_form(code, item_i, price_i) \rangle^{\uparrow}, \\
 & \quad (ok, payment\_form(code, item_i, price_i, name, cc)), \\
 & \quad (ok, payment\_form(code, item_i, faked\_price, name, cc)), \\
 & \quad (no\_payment) \\
\mathcal{I}(\overline{selling}), \mathcal{I}(r^{-}_{sell}) & \ni \langle item \rangle^{M}, (order\_form(code, item_i, price_i))^{M}, \\
 & \quad \langle ok, payment\_form(code, item_i, price_i, name, cc) \rangle^{M} \\
 & \quad \langle no\_payment \rangle^{M} \\
\mathcal{I}(price\_db), I(r^{+}_{price}) & \ni (item_i), \langle price_i \rangle \\
\mathcal{R}(x_{price_i}), \mathcal{R}(z_{price_i}) & \ni price_i \\
\mathcal{R}(y_{price_i}) & \ni price_i, faked\_price \\
\mathcal{K} & \ni price_i, faked\_price, payment\_form(code, item_i, price_i, name, cc), \\
 & \quad payment\_form(code, item_i, faked\_price, name, cc)
\end{array}$$

Fig. 5. Some Analysis Entries of $(S \mid DBI) \mid C$

$$\begin{array}{ll}
\mathcal{I}(selling), \mathcal{I}(r^{+}_{sell}) & \ni (item_i), l_0 \\
\mathcal{I}(l_0) & \ni \langle item_i \rangle, \overline{price\_db} \\
\mathcal{I}(\overline{price\_db}), I(r^{-}_{price}) & \ni \langle item_i \rangle (?x_{price}) \langle price \rangle^{\uparrow}, \\
\mathcal{I}(l_1) & \ni (?y^{l_1}_{price}), \langle form(code, item, price_i) \rangle, \\
 & \quad (ok, payment\_form(code, item_i, price_i, ?y_{name}, ?y_{cc})), \\
 & \quad (no\_payment) \\
\mathcal{R}(x_{price_i}) & \ni price_i \\
\mathcal{R}(y^{l_1}_{price_i}) & \ni price_i
\end{array}$$

Fig. 6. Some Analysis Entries of $(S' \mid DBI) \mid C$

$\langle ok, payment\_form(code, item_i, price_i, name, cc) \rangle^{M}$ belongs to $\mathcal{I}(\overline{selling})$ and $faked\_price \in \mathcal{K}$, then $\langle ok, payment\_form(code, item_i, faked\_price, name, cc) \rangle$ can synchronise with the input $(ok, payment\_form(code, item_i, price_i, name, cc))$.

In the second formulation, the problem does not arise, because: (i) the DBI sends the price to the bank that, in turn, forwards it to the client; (ii) the shop service checks if the price returned by the costumer matches against the one returned by the DBI component, thus avoiding the attack. The first fix has to do with the overall design of the service. The second has to do with input validation and, technically, is obtained by using pattern matching on the third value received in the payment form, that should match with the price $y^{l_1}_{price}$, as correctly predicted by the analysis of the second formulation, shown in Fig 6.

4 Conclusion

Often, component-based applications, i.e. those that are mostly implemented by connected existing applications, as web services, only limit the analysis of their component applications to the functionalities they actually use. A good practice could be to consider all the functionalities an application offers, before including it in the composition, in order to check all the possible inputs the overall web service can eventually provide and to keep trace where data-validation is applied or is missing.

We applied a Control Flow Analysis to an example inspired by a known logic-flawed application for e-commerce, specified in service oriented calculus as CaSPiS. Ours is a proof-of-concept work: we chose to detect the application misuse, by analysing the static approximation of the behaviour of the system. Our analysis can be easily specialised to capture specific properties of interest, e.g., data integrity in this example. We do not describe the specialisation process here, but only remark that the core of the analysis here introduced is preserved (for a similar process, see, e.g., the analyses for the LySa calculus in [8,10]). Our proposal shows that the chosen level of abstraction of services and of their features is suitable to investigate logic flaws. In fact, a calculus like CaSPiS abstractly expresses the key aspects of service oriented computing as primitives, without requiring any further codification, thus allowing us to focus on the security flaws that arise at the level of services, and to ignore those arising at the level of the underlying protocols. Indeed different forms of basic interactions are distinguished and regulated on their own (services are globally available, while ordinary I/O communications are context sensitive); and sessions are an implicit mechanism for enclosing the communications between a caller and its callee, avoiding external interferences.

References

1. Acciai, L., Boreale, M.: Type Abstractions of Name-Passing Processes. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 302–317. Springer, Heidelberg (2007)
2. Acciai, L., Boreale, M.: A Type System for Client Progress in a Service-Oriented Calculus. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 642–658. Springer, Heidelberg (2008)
3. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
4. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Semantics-Based Design for Secure Web Services. IEEE Transactions on Software Engineering 34(1), 33–49 (2008)
5. Bhargavan, K., Fournet, C., Gordon, A.D.: Verified Reference Implementations of WS-Security Protocols. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 88–106. Springer, Heidelberg (2006)
6. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: Computer Security Foundations Workshop (CSFW) (2001)
7. Bodei, C., Bracciali, A., Chiarugi, D.: Control Flow Analysis for Brane Calculi. ENTCS, vol. 227, pp. 59–75. Elsevier, Amsterdam (2009)
8. Bodei, C., Brodo, L., Degano, P., Gao, H.: Detecting and Preventing Type Flaws at Static Time. To appear in Journal of Computer Security (2009)
9. Bodei, C., Brodo, L., Bruni, R.: Static Detection of Logic Flaws in Service Applications. Technical Report, Dipartimento di Informatica, Università di Pisa (2009)

10. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static Validation of Security Protocols. Journal of Computer Security 13(3), 347–390 (2005)
11. Bond, M., Clulow, J.: Extending Security Protocol Analysis: New Challenges. ENTCS, vol. 125(1), pp. 13–24. Elsevier, Amsterdam (2005)
12. Bonelli, E., Compagnoni, A., Gunter, E.: Typechecking Safe Process Synchronization. In: Proc. Foundations of Global Ubiquitous Computing. ENTCS, vol. 138(1), pp. 3–22. Elsevier, Amsterdam (2005)
13. Boreale, M., Bruni, R., De Nicola, R., Loreti, M.: Sessions and Pipelines for Structured Service Programming. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 19–38. Springer, Heidelberg (2008)
14. Bruni, R.: Calculi for service-oriented computing. In: Proc. of 9th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Web Services (SFM 2009). LNCS, vol. 5569, pp. 1–41. Springer, Heidelberg (2009)
15. Bruni, R., Mezzina, L.G.: Types and Deadlock Freedom in a Calculus of Services, Sessions and Pipelines. In: Meseguer, J., Roşu, G. (eds.) AMAST 2008. LNCS, vol. 5140, pp. 100–115. Springer, Heidelberg (2008)
16. Kitchin, D., Cook, W.R., Misra, J.: A language for task orchestration and its semantic properties. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 477–491. Springer, Heidelberg (2006)
17. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. IEEE TIT, IT-29(12), 198–208 (1983)
18. Kolundzija, M.: Security Types for Sessions and Pipelines. In: Proc. of the 5th International Workshop on Web Services and Formal Methods (WS-FM 2008). LNCS, vol. 5387, pp. 175–189. Springer, Heidelberg (2009)
19. Nabi, F.: Secure business application logic for e-commerce systems. Computers & Security 24(3), 208–217 (2005)
20. Nielson, F., Riis Nielson, H., Priami, C., Schuch da Rosa, D.: Control Flow Analysis for BioAmbients. ENTCS, vol. 180(3), pp. 65–79. Elsevier, Amsterdam (2007)
21. Riis Nielson, H., Nielson, F.: Flow Logic: a multi-paradigmatic approach to static analysis. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 223–244. Springer, Heidelberg (2002)
22. OASIS Technical Commitee. Web Services Security (WS-Security) (2006)
23. Neohapsis Archives. Price modification possible in CyberOffice Shopping Cart, `http://archives.neohapsis.com/archives/bugtraq/2000-10/0011.html`
24. Backes, M., Mödersheim, S., Pfitzmann, B., Viganò, L.: Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 428–445. Springer, Heidelberg (2006)
25. Woo, T.Y.C., Lam, S.S.: A semantic model for authentication protocols. In: Proc. of IEEE Symposium on Security and Privacy (1993)

Improving the Semantics of Imperfect Security

Niklas Broberg and David Sands

Chalmers University of Technology, Sweden

Information flow policies that evolve over time (including, for example, declassification) are widely recognised as an essential ingredient in useable information flow control system. In previous work ([BS06a, BS06b]) we have shown one approach to such policies, *flow locks*, which is a very general and flexible system capable of encoding many other proposed approaches.

However, any such policy approach is only useful if we have a precise specification – a semantic model – of what we are trying to enforce. A semantic model gives us insight into what a policy actually guarantees, and defines the precise goals of any enforcement mechanism. Unfortunately, semantic models of declassification can be both inaccurate and difficult to understand. This was definitely the case for the flow locks system as presented in [BS06a, BS06b], and we have found that the main problem is one common to most proposed models to date. We will start by discussing the problem in general, and then go on to sketch its solution for the flow locks system specifically.

The Flow Sensitivity Problem. The most commonly used semantic definition of secure information flow defines security by comparing any two runs of a system in environments that only differ in their secrets. A system is secure or *noninterfering* if any two such runs are indistinguishable to an attacker. Many semantic models for declassification are built from adaptations of such a two-run noninterference condition.

Such adaptations are problematic for a number of reasons, but the most fundamental problem is that they lack a formal and intuitive attacker model. An attacker would not observe two runs and try to deduce something from their differences, which is what a two-run model inherently claims.

There are also technical problems arising from the use of a two-run system. Consider the first point in a run at which a declassification occurs. From this point onwards, two runs may very well produce different observable outputs. A semantic model for declassification must constrain the difference at the declassification point in some way, and further impose some constraint on the remainder of the computation. The prevailing approach (e.g. [MS04, EP05, EP03, AB05, Dam06, MR07, BCR08, LM08]) to give meaning to declassification is to reset the environments of the systems so as to restore the low-equivalence of environments at the point after a declassification. We refer to this as the *resetting approach* to declassification semantics.

The down-side of the resetting approach is that it is *flow insensitive*. This implies that the security of a program P containing a reachable subprogram Q requires that Q be secure independently of P. Another instance of the problem is that dead code can be viewed as semantically significant, so that a program will be rejected because of some insecure dead code. Note that flow insensitivity might be a perfectly reasonable property for a particular *enforcement* mechanism such as a type system – but in a sequential setting it has no place as a fundamental semantic requirement.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 88–91, 2009.

A Knowledge-based Approach. Comparing two runs is fairly intuitive for standard noninterference, but in the presence of policy changes such as the opening or closing of locks (in our work) or declassification (in other work) it can be hard to see how the semantic definition really relates to what we can say about an attacker.

One recent alternative to defining the meaning of declassification is Gradual Release [AS07], which uses a more explicit attacker model whereby one reasons about what an attacker learns about the initial inputs to a system as computation progresses.

The basic idea can be explained when considering the simple case of noninterference between an initial memory state, which is considered secret, and public outputs. The model assumes that the attacker knows the program itself P.

Suppose that the attacker has observed some (possibly empty) trace of public outputs t. In such a case the attacker can, at best, deduce that the possible initial state is one of the following: $K_1 = \{N \mid \text{Running } P \text{ on } N \text{ can yield trace } t\ \}$. Now suppose that after observing t the attacker observes the further output u. Then the attacker knowledge is $K_2 = \{N \mid \text{Running } P \text{ on } N \text{ can yield trace } t \text{ followed by } u\ \}$. For the program to be considered noninterfering, in all such cases we must have $K_1 = K_2$. The gradual release idea weakens noninterference by allowing $K_1 \neq K_2$, but only when u is explicitly labelled as a declassification event.

This style of definition is the key to our new flow lock semantics. The core challenge is then to determine what part of the knowledge must remain constant on observing the output u by viewing the trace from the perspective of the lock-state in effect at that time.

Flow locks: the basic idea. Flow locks are a simple mechanism for interfacing between information flow policies given to data, and the code that processes that data. Security policies are associated with the storage locations in a program. In general a *policy* p is a set of *clauses*, where each clause of the form $\Sigma \Rightarrow \alpha$ states the circumstances (Σ) under which actor α may view the data governed by this policy. Σ is a set of locks, and for α to view the data all the locks in Σ must be open.

A lock is a special variable in the sense that the only interaction between the program and the lock is via the instructions to *open* or *close* the lock. In this way locks can be seen as a purely compile-time entity used to specify the information flow policy. We say that x *is visible to* α *at* Δ if the policy of x contains $\Sigma \Rightarrow \alpha$ for some $\Sigma \subseteq \Delta$.

One aim of the flow locks approach is to provide a general language into which a variety of information flow mechanisms can be encoded, to let flow locks serve as a unifying framework for various policy mechanisms. As a simple example of such an encoding from [BS06a], consider a system with data marked with "high" or "low", and a single statement $\ell := \mathit{declassify}(h)$ taking high data in h and downgrading it to low variable ℓ. We can encode this using flow locks by letting low variables be marked with $\{\mathit{high}; \mathit{low}\}$ and high variables with $\{\mathit{Decl} \Rightarrow \mathit{high}; \mathit{low}\}$. The statement $\ell := \mathit{declassify}(h)$ can then be encoded with the following sequence of statements: $\texttt{open}\ \mathit{Decl}; \ell := h; \texttt{close}\ \mathit{Decl}$.

In common with many of the approaches cited above, the previous semantic security model for flow locks is defined using a resetting approach. All the problems discussed above are thus manifested in the old flow locks security model – no intuitive attacker model leading to an imperfect notion of knowledge gained, and a flow insensitive notion of security that rules out many perfectly secure programs for purely technical reasons.

A new semantics for Flow Locks. First we need a suitable attacker model. The model given in [AS07] is very simple, to match the simple declassification mechanism they consider. For flow locks we need a slightly more complex model, both since we deal with multiple actors, but also since flows are more fine-grained, so that a secret may be declassified for an actor through a series of steps.

To verify that a program is allowed, we need to validate the flows at each "level" that a secret may flow to, where a level corresponds to a certain set of locks guarding a location from a given actor. We note that these levels correspond to the points in the lattice $Actors \times \mathcal{P}(Locks)$, which leads us to our formal attacker model: An attacker A is a pair of an actor α and a set of locks $\varDelta$, formally $A = (\alpha, \varDelta) \in Actors \times \mathcal{P}(Locks)$. We refer to the lockstate component of an attacker as his *capability*, and say that a location x is visible to $A = (\alpha, \varDelta)$ iff x is visible to α at $\varDelta$.

The flows that we need to validate are the assignments, so each assignment must be registered in the "trace" that the program generates when run. An output u in the trace is visible to attacker A if the variable being assigned to is, and an *A-observable trace* is the restriction of a full trace to only include the outputs visible to A.

To reason about attacker knowledge we also need to be able to focus on the parts of a memory which are visible to a given attacker. Given an attacker A, we say a memory L is A-low if $dom(L) = \{x \mid A \text{ can see } x\}$. We say that two memories M and N are A-equivalent, written $M \sim_A N$ if their A-low projections are identical – i.e. they agree on all variables that A can see. The knowledge gained by an attacker A from observing a sequence of outputs $\vec{w}$ of a program c starting with a A-low memory L is then defined to be the set of all possible starting memories that could have lead to that observation:

$$k_A(\vec{w}, c, L) = \{M | M \sim_A L, \text{running } c \text{ on } M \text{ can yield A-visible trace } \vec{w}\}$$

Intuitively, for a program to be flow lock secure we must consider the perspective of each possible attacker A, and how his knowledge of the initial memory evolves as he observes successive outputs. The requirement for each output thus observed is that knowledge of the initial memory only increases if the attacker's inherent capabilities are weaker than the program lockstate in effect at the time of the output. The intuition here is that an attacker whose capability includes the program lock state in effect should already be able to see the locations used when computing the value that is output. Thus no knowledge should be gained by such an attacker.

For convenience we introduce the notion of a *run*, which is just an output trace together with the lockstate in effect at the time of the last output in the sequence:

$$\begin{aligned} Run_A(\varSigma, c, L) = \{(\vec{w}w, \varOmega) | M \sim_A L,& \\ &\text{running } c \text{ with starting memory } M \text{ and starting lockstate } \varSigma \text{ yields} \\ &A\text{-visible output trace } \vec{w}w \text{ where } \varOmega \text{ is the lockstate in effect at output } w\} \end{aligned}$$

Finally we can define our security requirement in terms of runs. A program c is said to be $\varSigma$-flow lock secure, written $FLS(\varSigma, c)$ iff for all attackers $A = (\alpha, \varDelta)$, all A-low memories L, and all runs $(\vec{w}w, \varOmega) \in Run_A(\varSigma, c, L)$ such that $\varOmega \subseteq \varDelta$ we have

$$k_A(\vec{w}w, c, L) = k_A(\vec{w}, c, L)$$

This definition directly captures the intuition that we started out with. An attacker whose capabilities includes the current lockstate in effect at the time of the output should learn nothing new when observing that output. Attackers who do not fulfill this criterion have no constraint on what they may learn at this step. But note that this cannot lead to unchecked flows because we quantify over *all* attackers including, in particular, those with sufficient capabilities.

Further reading. This extended abstract accompanies an invited talk, and what we describe here is still work in progress. The slides and latest version of a full paper are available at: `http://www.cs.chalmers.se/~dave/ARSPA-WITS09`

References

[AB05] Almeida Matos, A., Boudol, G.: On declassification and the non-disclosure policy. In: Proc. IEEE Computer Security Foundations Workshop, June 2005, pp. 226–240 (2005)

[AS07] Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: Proc. IEEE Symp. on Security and Privacy, May 2007, pp. 207–221 (2007)

[BCR08] Barthe, G., Cavadini, S., Rezk, T.: Tractable enforcement of declassification policies. In: Proc. IEEE Computer Security Foundations Symposium (2008)

[BS06a] Broberg, N., Sands, D.: Flow locks: Towards a core calculus for dynamic flow policies. In: Sestoft, P. (ed.) ESOP 2006. LNCS, vol. 3924, pp. 180–196. Springer, Heidelberg (2006)

[BS06b] Broberg, N., Sands, D.: Flow locks: Towards a core calculus for dynamic flow policies. Technical report, Chalmers University of Technology and Göteborgs University (May 2006); Extended version of [BS06a]

[Dam06] Dam, M.: Decidability and proof systems for language-based noninterference relations. In: Proc. ACM Symp. on Principles of Programming Languages (2006)

[EP03] Echahed, R., Prost, F.: Handling harmless interference. Technical Report 82, Laboratoire Leibniz, IMAG (June 2003)

[EP05] Echahed, R., Prost, F.: Security policy in a declarative style. In: Proceedings of the 7th International Conference on Principles and Practice of Declarative Programming (PPDP 2005), Lisboa, Portugal (July 2005)

[LM08] Lux, A., Mantel, H.: Who can declassify? In: Preproceedings of the Workshop on Formal Aspects in Security and Trust (FAST) (2008)

[MR07] Mantel, H., Reinhard, A.: Controlling the what and where of declassification in language-based security. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 141–156. Springer, Heidelberg (2007)

[MS04] Mantel, H., Sands, D.: Controlled declassification based on intransitive noninterference. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 129–145. Springer, Heidelberg (2004)

Analysing PKCS#11 Key Management APIs with Unbounded Fresh Data

Sibylle Fröschle[1] and Graham Steel[2]

[1] School of Informatics, University of Oldenburg, Germany
[2] LSV, ENS Cachan & CNRS & INRIA, France

Abstract. We extend Delaune, Kremer and Steel's framework for analysis of PKCS#11-based APIs from bounded to unbounded fresh data. We achieve this by: formally defining the notion of an *attribute policy*; showing that a well-designed API should have a certain class of policy we call *complete*; showing that APIs with complete policies may be safely abstracted to APIs where the attributes are fixed; and proving that these *static* APIs can be analysed in a small bounded model such that security properties will hold for the unbounded case. We automate analysis in our framework using the SAT-based security protocol model checker SATMC. We show that a symmetric key management subset of the Eracom PKCS#11 API, used in their ProtectServer product, preserves the secrecy of sensitive keys for unbounded numbers of fresh keys and *handles*, i.e. pointers to keys. We also show that this API is not robust: if an encryption key is lost to the intruder, SATMC finds an attack whereby all the keys may be compromised.

1 Introduction

RSA Laboratories standard PKCS#11 defines Cryptoki, a general purpose API for cryptographic devices such as smartcard security tokens and cryptographic hardware security modules (HSMs) [8]. It has been widely adopted in industry. As well as providing cryptographic functionality, PKCS#11 is also supposed to enforce certain security properties. In particular, it is stated in the standard that even if the device is connected to an untrusted machine where the operating system and device drivers might be compromised, keys marked "sensitive" cannot be compromised [8, §7]. However, in 2003, Clulow presented a number of attacks, i.e. sequences of valid PKCS#11 commands, which result in sensitive keys being revealed in the clear [2]. A typical one is the so-called 'key separation attack'. The name refers to the fact that a key may have conflicting roles. Clulow gives the example of a key configured for decryption of ciphertexts, and for 'wrapping', i.e. encryption of other keys for secure transport. To determine the value of a sensitive key, the attacker calls the 'wrap' command and then the 'decrypt' command, as shown in Figure 1. Here (and subsequently) $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$ is what we call a *handle binding*, i.e. a function application modelling the fact that n_1 is a *handle* for (or pointer to) key k_1 on the device. The symmetric encryption of k_1 under key k_2 is represented by the term $\mathsf{senc}(\mathsf{k}_1, \mathsf{k}_2)$. We model commands

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 92–106, 2009.

by giving the inputs on the left of the arrow, and the output on the right. The result of the attack is to reveal the value of k_1 in the clear.

Delaune, Kremer and Steel have proposed a formal model for the operation of the PKCS#11 API [5], which together with a model checker allowed various configurations of the API to be examined and several new attacks to be found. However, their model requires a bound on the number of freshly generated names, used to model new keys and handles. This means that when the model is shown to be secure up to the bound, we cannot be sure that there is not an attack on a larger model of the API. Moreover, they were only able to find secure configurations by severely restricting the functionality of the interface - in effect restricting the update of long-term keys to an operation requiring the device to be connected to a trusted machine. In this paper, we motivate a restriction to a certain class of APIs, showing how APIs outside this class are likely to be problematic because of the wrapping and unwrapping operations intrinsic to the API. We prove theorems showing that if there is an attack on an API in our class in the unbounded model, then there is an attack in a particular bounded model of a size suitable for treatment with a model checker. We use version 3.0 of the SAT-based model checker for security protocols SATMC [1] to carry out experiments on these APIs.

The rest of the paper is organised as follows. We first define our formal model (§2). We make explicit the idea of an *attribute policy*, which specifies what roles a key may have, and insist that API rules give rise to a certain class of attribute policies, called *complete* policies (§3). We explain why this is a reasonable demand, and show that an API with a complete policy can be mapped to an equivalent (with respect to security) API with a static policy, i.e. one where the attributes are non-mutable. We then show how security analysis of APIs with static policies for unbounded numbers of keys and handles can be performed by model checking carefully designed small bounded models (§4). Finally, we prove by model checking the secrecy of sensitive keys for a small PKCS#11 based API, for unbounded numbers of keys and handles (§5). This API is a subset of the implementation of PKCS#11 used by Eracom/SafeNet in their ProtectServer product [7]. We also show how the loss of a key with the encrypt attribute set can lead to the loss of all the keys, thanks to an attack found by SATMC. We show how to modify the API to prevent the attack. Thus finally we are able to prove some robust guarantees of security for a functional subset of PKCS#11, albeit in our symbolic model. We conclude in section 6.

2 Formal Model

In Cryptoki's logical view a "cryptographic token" (or simply "token") is a device that stores *objects* and can perform cryptographic functions with these objects. For the subset of Cryptoki functions that we will model, the objects will always be cryptographic *keys*. Each object will be referenced by a *handle*, which (like a file handle) can be thought of as a pointer to the object. Several instances of the same object are possible. *Attributes* are characteristics that distinguish a

Initial knowledge: The intruder knows $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$ and $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$. The name n_2 has the attributes wrap and decrypt set whereas n_1 has the attribute sensitive and extract set.

Trace:

Wrap: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \rightarrow \mathsf{senc}(\mathsf{k}_1, \mathsf{k}_2)$
SDecrypt: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{senc}(\mathsf{k}_1, \mathsf{k}_2) \rightarrow \mathsf{k}_1$

Fig. 1. Decrypt/Wrap attack

particular instance of an object. They enable or restrict the way functions can be applied to the instance of the object. For example, the instance of a key can only be used to decrypt a ciphertext (provided by the application programmer) if the attribute decrypt is set. The combinations of attributes that an instance of an object is allowed to have, and how the attributes can be set and unset, depends on the particular configuration of the token. We will model this by defining the concept of an *attribute policy*. The API commands and their semantics will as usual be modelled by a *rewriting system* over a *term algebra*.

Our model has two main differences to that of Delaune, Kremer and Steel (DKS) [5]. First, justified by results therein [5, Theorem 1] we will work with a *typed* term algebra, silently insisting that all APIs considered are *well-moded* with respect to our types. Second, our concept of defining our APIs relative to a *attribute policy* is new. Following DKS [5] we will model handles by nonces, and the reference of an object k by a nonce n by the function application $h(n, k)$, where h is a symbol unknown to the attacker. However, to avoid ambiguity in informal discussions, we will call such a function application a *handle binding* rather than just a *handle* as in [5].

2.1 Term Algebra

We consider the following types: Cipher, Key, Nonce, and HBinding where Key and Nonce are atomic types. We assume four countably infinite sets: a set $\mathcal{N}$ of ground terms of type Nonce; a set $\mathcal{X}_{\mathcal{N}}$ of variables of type Nonce; a set $\mathcal{K}$ of ground terms of type Key; and a set $\mathcal{X}_{\mathcal{K}}$ of variables of type Key. All other terms are built up from the atomic terms by the following operators:

- encryption $\mathsf{enc} : \mathsf{Key} \times \mathsf{Key} \rightarrow \mathsf{Cipher}$, and
- handle binding $h : \mathsf{Nonce} \times \mathsf{Key} \rightarrow \mathsf{HBinding}$.

We denote the set of *terms* by $\mathcal{T}$, and the set of *ground terms*, i.e., terms that do not contain any variables, by $\mathcal{GT}$. We define the set of *template terms*, denoted by $\mathcal{TT}$, to be the set of terms that do not contain any subterm of $\mathcal{N} \cup \mathcal{K}$. Thus, a template term is a term that does not contain any ground nonce or key but may use variables to represent them. We define *key terms* as $\mathcal{T}_{\mathcal{K}} = \mathcal{K} \cup \mathcal{X}_{\mathcal{K}}$, and *nonce terms* as $\mathcal{T}_{\mathcal{N}} = \mathcal{N} \cup \mathcal{X}_{\mathcal{N}}$ respectively. Given a term $t \in \mathcal{T}$, let *nonces*(t) be the set of nonce terms occurring in t. We extend this notation to sets of terms in the obvious way.

We also consider a *finite* set $\mathcal{A}$ of predicate symbols, disjoint from the other symbols, which we call *attributes*. In this paper we work with the following restricted set of attributes:

$$\mathcal{A} = \{\mathsf{encrypt}, \mathsf{decrypt}, \mathsf{wrap}, \mathsf{unwrap}\}.$$

The set of *attribute terms* is defined as

$$\mathcal{AT} = \{att(n) \mid att \in \mathcal{A} \;\&\; n \in \mathcal{T}_{\mathcal{N}}\}.$$

Attribute terms will be interpreted as propositions. A *literal* is an expression a or $\neg a$ where $a \in \mathcal{AT}$. *Ground attribute terms* and *literals*, and *template attribute terms* and *literals* are defined analogously to above. We denote template literals by $\mathcal{TL}$.

2.2 Attribute Policies and Attribute States

An *attribute valuation* is a partial function $a : \mathcal{A} \to \{\bot, \top\}$. If an attribute valuation is total then it is an *object state*. We write $\mathcal{S}$ for the set of object states. An *attribute policy* is a finite directed graph $P = (S_P, \to_P)$ where $S_P \subseteq \mathcal{S}$ is the set of *allowable object states*, and $\to_P \subseteq S_P \times S_P$ is the set of *allowable transitions* between the object states.

A *token state* is a partial function $A : \mathcal{N} \to \mathcal{S}$ which assigns an object state to each nonce in $dom(A)$, and thus to the keys represented by the nonces in $dom(A)$. Given an attribute policy P, we say A *conforms to* P iff $ran(A) \subseteq S_P$. Every token state induces a valuation V_A for the set of attribute terms over $dom(A)$, defined by:

$$V_A(att(n)) = \begin{cases} \top & \text{if } A(n)(att) = \top \\ \bot & \text{otherwise,} \end{cases} \qquad V_A(\neg att(n)) = \begin{cases} \top & \text{if } A(n)(att) = \bot \\ \bot & \text{otherwise.} \end{cases}$$

2.3 API Rewrite Systems

We model two types of API commands: *key management commands* and the command `SetAttributeValue`, which allows the API programmer to manipulate the token state. The actions of the intruder will be modelled by a fixed set of *intruder rules*.

Syntax and informal semantics. Key management commands may generate new objects, in which case the token state will be extended by assigning an object state to each new object. The execution of a key management command may be subject to whether the objects referenced have certain attributes set or unset. Thus, formally, such commands are described by *key management rules* of the form

$$R : \quad T \;;\; L \xrightarrow{\mathsf{new}\,\tilde{x}} T' \;;\; A_{new}$$

where $T \subseteq \mathcal{TT}$ is a set of template terms, $L \subseteq \mathcal{TL}$ is a set of template literals such that $vars(L) \subseteq vars(T)$, $\tilde{x} \subseteq \mathcal{X}_\mathcal{N} \cup \mathcal{X}_\mathcal{K}$ is a set of key and nonce variables such that $\tilde{x} \cap vars(T) = \emptyset$, $T' \subseteq \mathcal{TT}$ is a set of template terms such that $vars(T') \subseteq vars(T) \cup \tilde{x}$, and A_{new} is a template token state with $dom(A_{new}) = nonces(\tilde{x})$.

The `SetAttributeValue` command is modelled by the parameterized *attribute rule* $\mathsf{set}(n, a)$ where $n \in \mathcal{N}$, and a is an attribute valuation. Whether an instance of this rule can indeed be applied at a given token state will depend on the attribute policy.

An *API rewrite system* is a pair $(P, \mathcal{R})$ where P is an attribute policy and $\mathcal{R}$ is a set of *key management rules* that conform to P in the following way: every rule that generates a new handle is parameterized by an object state $a \in S_P$ such that the new handle is assigned object state a in A_{new}. Note that the attribute rule $\mathsf{set}(n, a)$ is assumed to be part of all APIs modelled here and does not need to be specified explicitly.

Example 1. As an example consider the rewrite system $(P, \mathcal{R})$ where $\mathcal{R}$ is given by the rules in Figure 2, and P is defined by:

- $S_P = \{\{\mathsf{encrypt}, \mathsf{decrypt}\}, \{\mathsf{wrap}, \mathsf{unwrap}\}\}$, and
- $\rightarrow_P = (S_P, S_P \times S_P)$.

As in this example we often specify the allowed object states concisely as sets of attributes rather than functions. Note that due to our concept of attribute policy we do not need to specify rules for setting or unsetting attributes.

Intruder capability. The following two rules represent the deduction capabilities of the intruder. We will denote the intruder theory by $\mathcal{I}$.

$$\mathsf{I\text{-}SEncrypt}: \quad \mathsf{k_e},\ \mathsf{k} \rightarrow \mathsf{senc}(\mathsf{k}, \mathsf{k_e})$$

$$\mathsf{I\text{-}SDecrypt}: \quad \mathsf{senc}(\mathsf{k}, \mathsf{k_e}),\ \mathsf{k_e} \rightarrow \mathsf{k}$$

Note that the intruder rules can be considered to be in the same format as the key management rules. We will make use of this when defining the semantics.

$$\mathsf{Wrap}: \ \mathsf{h}(\mathsf{n_w}, \mathsf{k_w}), \mathsf{h}(\mathsf{n}, \mathsf{k});\ \mathsf{wrap}(\mathsf{n_w}) \rightarrow \mathsf{senc}(\mathsf{k}, \mathsf{k_w})$$

For all $a \in S_P$:

$$\mathsf{Unwrap(a)}: \quad \mathsf{h}(\mathsf{n_w}, \mathsf{k_w}), \mathsf{senc}(\mathsf{k}, \mathsf{k_w});\ \mathsf{unwrap}(\mathsf{n_w}) \xrightarrow{\mathsf{new\ n}} \mathsf{h}(\mathsf{n}, \mathsf{k});\ \{\mathsf{n} \mapsto \mathsf{a}\}$$

$$\mathsf{KeyGenerate(a)}: \quad \xrightarrow{\mathsf{new\ n,k}} \mathsf{h}(\mathsf{n}, \mathsf{k});\ \{\mathsf{n} \mapsto \mathsf{a}\}$$

$$\mathsf{SEncrypt}: \quad \mathsf{h}(\mathsf{n_e}, \mathsf{k_e}),\ \mathsf{k};\ \mathsf{encrypt}(\mathsf{n_e}) \rightarrow \mathsf{senc}(\mathsf{k}, \mathsf{k_e})$$

$$\mathsf{SDecrypt}: \quad \mathsf{h}(\mathsf{n_e}, \mathsf{k_e}), \mathsf{senc}(\mathsf{k}, \mathsf{k_e});\ \mathsf{decrypt}(\mathsf{n_e}) \rightarrow \mathsf{k}$$

Fig. 2. PKCS#11 symmetric key management subset relative to an attribute policy P

Semantics. Let $(P, \mathcal{R})$ be an API rewrite system. A *state* of $(P, \mathcal{R})$ is a pair (Q, A) where $Q \subseteq \mathcal{GT}$ is a set of ground terms, and A is a token state such that $dom(A) = nonces(Q)$ and A conforms to P. Given a state (Q, A) and a rule $R \in \mathcal{R} \cup \mathcal{I}$ of the form $T; L \xrightarrow{\mathsf{new}\,\tilde{x}} T', A_{new}$, we say R *can be applied* at (Q, A) under substitution θ if

1. θ is grounding for R and assigns to the variables in $\tilde{x}$ distinct nonces and keys that do not occur in Q,
2. $T\theta \subseteq Q$, and
3. $V_A(L\theta) = \top$.

The *successor state* of (Q, A) under R, θ is then defined as (Q', A') where $Q' = Q \cup T'\theta$, and $A' = A \cup A_{new}\theta$. This gives rise to a transition $(Q, A) \rightsquigarrow_{R,\theta} (Q', A')$.

Given a state (Q, A) and an instance of the attribute rule R of the form $\mathsf{set}(n, a)$, we say R *can be applied* at (Q, A) if

1. $n \in nonces(Q)$, and
2. $A(n) \to_P a'$, where a' is defined by

$$a'(att) = \begin{cases} a(att) & \text{if } att \in dom(a), \\ A(n)(att) & \text{otherwise.} \end{cases}$$

The *successor state* of (Q, A) under R is then (Q, A') where A' is defined by $A'(n) = a'$, and $A'(n') = A(n')$ for all $n' \in dom(A)$ such that $n' \neq n$. This gives rise to a transition $(Q, A) \rightsquigarrow_R (Q', A')$.

We write $(Q, A) \rightsquigarrow (Q', A')$ when there is some transition from (Q, A) to (Q', A'). We write $\rightsquigarrow^*$ for the reflexive and transitive closure of $\rightsquigarrow$. We call *derivation* a sequence of rule applications $D = (S_0, A_0) \rightsquigarrow (S_1, A_1) \cdots \rightsquigarrow (S_n, A_n)$. We call (S_0, A_0) the initial state of D and (S_n, A_n) the final state of D.

Queries. A *query* is a pair (T, L) where T is a set of template terms and L is a set of template literals. A state (Q, A) satisfies a query (T, L) iff there exists a substitution θ grounding for (T, L) and such that $T\theta \subseteq Q$ and $V_A(L\theta) = \top$. A derivation D satisfies a query (T, L) relative to an initial state (T_0, L_0) iff the initial state of D satisfies (T_0, L_0) and the final state of D satisfies (T, L).

3 Towards Static Attribute Policies

An attribute policy is called *static* if it rules out any change of object state. Formally, an attribute policy $P = (S, \to)$ is static if $\to\, = \emptyset$. It is clear that, in general, a given set of API rules is much easier to analyse and verify when the associated attribute policy is static: in an informal analysis one does not need to consider any side effects that may arise from moving between allowable object states; in a formal analysis one does not need to consider any attribute set/unset rules; thereby the state space is reduced and remains monotonic. Even though static attribute policies may seem very restrictive, we will argue that well-designed attribute policies should satisfy a criterion we call *completeness*, and that complete attribute policies may be safely abstracted to static attribute policies.

Due to limitation of space we will only consider API rewrite systems whose rewrite rules are positive in that all the tests of attributes are positive. In a future version of this paper we will show how our results can be carried over to all APIs.

API rewrite systems in positive form. A rule $T; L \xrightarrow{\mathsf{new}\,\tilde{x}} T', A_{new}$ is *positive* if all literals $l \in L$ are positive. An API rewrite system $(P, \mathcal{R})$ is *in positive form* if all the rules in $\mathcal{R}$ are positive.

Given an API rewrite system in positive form we adopt the following simplifications. An object state $a : \mathcal{A} \to \{\top, \bot\}$ can be viewed as the set of attributes $\{att \mid a(att) = \top\}$, and a token state $A : \mathcal{T}_{\mathcal{N}} \to \mathcal{S}$ as the set of attribute terms $\{att(n) \mid n \in dom(A) \;\&\; att \in A(n)\}$. The valuation induced by an attribute state A, V_A, then simplifies to $V_A(att(n)) = \top$ if and only if $att(n) \in A$. (We never need to consider valuations of negative literals.)

One could adopt similar conventions for API rewrite systems in general. However, due to our restriction to positive form we have: for two object states $a \subseteq a'$ the object state a' enables at least as many commands as a, and similarly for token states.

Complete attribute policies. In the following let $\mathcal{R}$ be a set of positive API rules. Assume we wish to obtain a secure attribute policy for the commands modelled by $\mathcal{R}$. Typically we will start out with an idea of which attributes are conflicting. For example, the attack in Figure 1 tells us that no object should ever have both wrap and decrypt set. In this way we can induce the set of allowable object states. It is clear from previously discovered attacks, however, that defining a safe transition relation between allowable states is non-trivial. For example, one might try to prevent the attack in Figure 1 by disallowing the attributes wrap and decrypt from being set on the same handle (which is illustrated by the policy of Example 1). The attack in Figure 3 shows that this will not suffice. To address this, one might decide to declare wrap and decrypt as 'sticky' attributes, i.e. attributes which cannot be unset. However, the fact that

Initial knowledge: The intruder knows $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$ and $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$. The handle n_2 has the attribute wrap set.

Trace:

Wrap: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \to \mathsf{senc}(\mathsf{k}_1, \mathsf{k}_2)$

Unset wrap: $\mathsf{set}(n_2, [\mathsf{wrap} \mapsto \bot])$

Set decrypt: $\mathsf{set}(n_2, [\mathsf{decrypt} \mapsto \top])$

SDecrypt: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{senc}(\mathsf{k}_1, \mathsf{k}_2) \to \mathsf{k}_1$

Fig. 3. Decrypt/Wrap attack II [9]

the intruder can generally wrap and unwrap keys in order to obtain multiple handles for the same key means the attack can still be performed [5, Fig. 4].

For the rest of this paper, we will consider a restricted class of policies:

Definition 1. *An attribute policy $P = (S, \rightarrow)$ is* complete *if P consists of a collection of disjoint, disconnected cliques, and for each clique C, $c_0, c_1 \in C \Rightarrow c_0 \cup c_1 \in C$.*

The intuition behind this is that we do not expect to be able to prevent the intruder from making copies of an object via the import and export mechanisms, as in certain known attacks [5, Fig. 4]. This means that for policy $P = (S, \rightarrow)$, $\forall s_1, s_2, s_3 \in S$, if $s_1 \rightarrow s_2$ and $s_1 \rightarrow s_3$ are in P, then by copying a handle in state s_1, the intruder can obtain what is effectively a handle for an object in state $s_2 \cup s_3$. A well designed policy should take this into account. We further require the transition relation to be symmetric. We believe that the results in this paper could be extended to relax this restriction, but observe that our current definition has the advantage of giving a simple and intuitive rule for attribute policy design. Of course not all complete policies will be secure: consider a trivial policy in which all object states, including conflicting ones, are connected (such as the policy of Example 1). However, complete policies are amenable to analysis, as we will now show.

End point abstractions. Let $P = (S, \rightarrow)$ be a complete attribute policy. We call the object states of S that are maximal in S with respect to set inclusion *end points* of P, denoted by $\varepsilon(P)$. Such object states are end points in the following sense: once an attacker has reached an end point for an object he does not gain anything by leaving it: any object state that can be reached from an end point will have less enabling power.

P naturally gives rise to a static attribute policy where the allowable object states are taken to be the end points of P. Formally, we define the *end point abstraction of* P, denoted by $EP(P)$, to be the attribute policy $(\varepsilon(P), \emptyset)$. In Theorem 1 below we prove that $EP(P)$ provides a sound and complete abstraction of P.

Given $a \in S$, define $\varepsilon(a)$ to be the uniquely given end point e such that $a \subseteq e$. Given an object state A, define $\varepsilon(A)$ to be the object state that results when replacing every map $[n \mapsto a] \in A$ by $[n \mapsto \varepsilon(a)]$. If a rule $R \in \mathcal{R}$ generates a new handle with object state a then $\varepsilon(R)$ is the rule that results from R by replacing a by $\varepsilon(a)$. (Recall that $\varepsilon(R) \in \mathcal{R}$ by definition of API rewrite systems.)

Proposition 1. *1. For all standard rules R, we have:*
if $(Q_1, A_1) \rightsquigarrow_R (Q_2, A_2)$ then $(Q_1, \varepsilon(A_1)) \rightsquigarrow_{\varepsilon(R)} (Q_2, \varepsilon(A_2))$.
2. For all attribute rule instances R, we have:
if $(Q_1, A_1) \rightsquigarrow_R (Q_2, A_2)$ then $\varepsilon(A_1) = \varepsilon(A_2)$ (and $Q_1 = Q_2$ as usual).

Proof. (1) is a consequence of the definition of $\varepsilon(a)$ and our restriction to positive rules. (2) follows since by definition of complete attribute policies if $a_1 \rightarrow a_2$ is a transition in P then $\varepsilon(a_1) = \varepsilon(a_2)$.

Theorem 1. *If there is a derivation under P from (Q_0, A_0) to (Q_m, A_m) then there is a derivation under $EP(P)$ from $(Q_0, \varepsilon(A_0)$ to $(Q_m, \varepsilon(A_m))$.*

Conversely, if there is a derivation under $EP(P)$ from (Q_0, A_0) to (Q_m, A_m) then there is a derivation under P from (Q_0, A_0) to (Q_m, A_m).

Proof. To prove the first part assume a derivation D under P. By Prop. 1 we can transform D into a derivation under $EP(P)$ in the following way: replace each state (Q_i, A_i) occurring in D by $(Q_i, \varepsilon(A_i))$, and remove all attribute rule transitions from D. The converse direction is immediate since $EP(P)$ is a subgraph of P.

Altogether our results mean that for our class of APIs it suffices to analyse security under a static attribute policy. As we will demonstrate in the next section, together with further insights, this will lead us to reducing two variants of the PKCS#11 API to a bounded model.

4 Reducing APIs with Static Attribute Policies to Bounded Models

We consider the symmetric key fragment of the PKCS#11 key management API as modelled in Figure 2, and the Eracom version of the API to be introduced in this section. We show that for both these rewrite systems, when we assume a static attribute policy, it is sufficient to work with a bounded number of freshly generated values.

In the following we assume an API rewrite system $(P, \mathcal{R})$ where $P = (\mathcal{E}, \emptyset)$ is any static attribute policy and $\mathcal{R}$ will be specified in each paragraph. Derivations will always be assumed to start from a given initial state (Q_0, A_0). Since we work with static attribute policies only, the object state of every handle stays constant:

Proposition 2. *Let $(Q_0, A_0) \rightsquigarrow (Q_1, A_1) \cdots \rightsquigarrow (Q_m, A_m)$ be a derivation. Let n be a nonce, and i such that n is generated by the ith transition. Then for all $j \in [i, m]$, $A_j(n)$ is defined and $A_i(n) = A_j(n)$.*

Justified by this, in the context of a derivation as above, for every nonce n occurring in D we write $A(n)$ for the one object state it can assume. Also note that we no longer need to consider any attribute rule instances.

Atom substitutions. One key insight behind our reductions is that we can eliminate freshly generated keys and handles whenever we are able to replace them by already existing keys and handles that provide the same functionality. Formally, we will require the concept of *atom substitutions*, following [6]. We use *Atoms* to denote the set of ground atomic terms, i.e., $\mathcal{N} \cup \mathcal{K}$.

An *atom substitution* is a partial function $\delta : Atoms \rightarrow Atoms$ that respects the type of the atoms. We extend atom substitutions to sets, relations, etc. in the usual way. Given a token state A, we say atom substitution δ, is *defined for* A if for all $n_1, n_2 \in dom(A)$ we have:

1. $n_1 \mapsto n_2 \in \delta$ & $n_2 \notin dom(\delta) \implies A(n_1) = A(n_2)$, and
2. $n_1 \mapsto n, n_2 \mapsto n \in \delta$ for some $n \implies A(n_1) = A(n_2)$.

Proposition 3. *If A is a token state and δ is an atom substitution defined for A then we have:*

1. *$A\,\delta$ is a token state with $dom(A\,\delta) = dom(A)\,\delta$, and*
2. *for all literals l, if $V_A(l)$ is defined then $V_{A\delta}(l\delta)$ is defined and $V_A(l) = V_{A\delta}(l\delta)$.*

The following two propositions show that atom substitutions preserve rule applications as well as queries in a natural way. Both propositions are routine to prove by Prop. 3 and inspection of the definitions.

Proposition 4. *Let R be a rule of the form $T; L \xrightarrow{\mathsf{new}\,\tilde{x}} T', A_{new}$. If $(Q, A) \rightsquigarrow_{R,\theta} (Q', A')$ and δ is an atom substitution defined for A such that $\theta(\tilde{x}) \cap (dom(\delta) \cup ran(\delta)) = \emptyset$ then we have $(Q, A)\delta \rightsquigarrow_{R,\theta\delta} (Q', A')\delta$.*

Proposition 5. *If a state (Q, A) satisfies a query (T, L) by a substitution θ, and δ is an atom substitution defined for A, then $(Q, A)\delta$ satisfies (T, L) by $\theta\delta$.*

PKCS#11 rewrite system. Consider the symmetric key fragment of the standard PKCS#11 API as modelled in Figure 2. There are only two rules that generate fresh values: KeyGenerate and Wrap.

It is intuitive that a pair k, n generated by KeyGenerate with, say, the object state of n set to ε provides the same functionality as any other pair k', n' generated by KeyGenerate with n' set to the same object state ε. Thus, it is plausible that to check whether a query is satisfied we need to consider at most one instance of KeyGenerate for each allowable object state. The following proposition gives us the means to successively delete instances of KeyGenerate from any derivation until we arrive at a derivation that is bounded in this way.

Proposition 6. *Assume a derivation*

$$(Q_0, A_0) \cdots \rightsquigarrow_{R_i} (Q_i, A_i) \cdots \rightsquigarrow_{R_j} (Q_j, A_j) \cdots \rightsquigarrow_{R_n} (Q_n, A_n)$$

such that for some n_i, n_j, k_i, k_j,

1. *R_i is an instance of* KeyGenerate *producing $h(n_i, k_i)$,*
2. *R_j is an instance of* KeyGenerate *producing $h(n_j, k_j)$, and*
3. *$A(n_i) = A(n_j)$.*

Then

$$\begin{aligned}(Q_0, A_0) \cdots \rightsquigarrow_{R_i} (Q_i, A_i) \cdots \rightsquigarrow_{R_{j-1}} (Q_{j-1}, A_{j-1}) \rightsquigarrow_{R_{j+1}\delta} (Q_{j+1}, A_{j+1})\delta \cdots \\ \cdots \rightsquigarrow_{R_n\delta} (Q_n, A_n)\delta\end{aligned}$$

is a derivation, where $\delta = [n_j \mapsto n_i, k_j \mapsto k_i]$.

Proof. This follows by Prop. 4 when considering that $(Q_{j-1}, A_{j-1}) = (Q_j, A_j)\delta$, and that δ satisfies the conditions of Prop. 4 with respect to each transition $\rightsquigarrow_{R_m}$, $m \geq j$.

It can be read from the rules that, given a key k, whenever two handles for k, say n_1 and n_2, have the same object state then they can be employed in exactly the same way. Hence, it is plausible that for each key we need at most one handle per allowable object state. The following proposition gives us the means to eliminate instances of Unwrap until we arrive at a derivation that is reduced in this way.

Proposition 7. *Assume a derivation*

$$(Q_0, A_0) \cdots \rightsquigarrow_{R_i} (Q_i, A_i) \cdots \rightsquigarrow_{R_n} (Q_n, A_n)$$

such that for some n_1*,* n_2*,* k

1. $h(n_1, k) \in Q_{i-1}$ *, and*
2. R_i *is an instance of* Unwrap *producing* $h(n_2, k)$ *with* $A(n_1) = A(n_2)$*.*

$$(Q_0, A_0) \cdots \rightsquigarrow_{R_{i-1}} (Q_{i-1}, A_{i-1}) \rightsquigarrow_{R_{i+1}\delta} (Q_{i+1}, A_{i+1})\delta \cdots \rightsquigarrow_{R_n\delta} (Q_n, A_n)\delta$$

is a derivation, where $\delta = [n_2 \mapsto n_1]$*.*

Proof. This follows similarly to Prop. 6.

Altogether we obtain:

Lemma 1. *If there is a derivation of a query* (T, L) *then there is a derivation* D' *of the same query such that, in* D'*, the following holds:*

1. *for all pairs* k_1, n_1 *and* k_2, n_2 *generated by* KeyGenerate*,* $k_1 \neq k_2$ *implies* $A(n_1) \neq A(n_2)$*;*
2. *for every key* k *and all handles* n_1*,* n_2 *with bindings* $h(n_1, k)$*,* $h(n_2, k)$*,* $n_1 \neq n_2$ *implies* $A(n_1) \neq A(n_2)$*.*

Proof. The lemma easily follows by successively applying the above two propositions and because by Prop. 5 the transformations preserve the query.

The lemma implies that for the PKCS#11 rewrite rules we can reduce the static model to a bounded model. We also make use of the fact that in a model with no disequalities on keys, an attacker is as powerful when given one key (without a handle) in his initial knowledge as when given many such keys.

Theorem 2. *In the PKCS#11 rewrite system with attribute policy* $(\mathcal{E}, \emptyset)$*, if there is a derivation of a query* (T, L) *then there is a derivation of the same query using*

1. *at most* $1 + |\mathcal{E}|$ *keys: at most* 1 *key in* Q_0*, and at most* $|\mathcal{E}|$ *keys generated by* KeyGenerate*, and*
2. *at most* $|\mathcal{E}| \times (1 + |\mathcal{E}|)$ *handles.*

For all $e \in \mathcal{E}$:

$$\mathsf{KeyGenerate(e)} : \xrightarrow{\mathsf{new\ n,k}} \mathsf{h(n,k); e(n)}$$

$$\begin{array}{l}\mathsf{Wrap(e)} : \\ \mathsf{h(n_w, k_w), h(n,k);\ wrap(n_w), e(n)} \xrightarrow{\mathsf{new\ k_m}} \mathsf{enc(k, k_w), enc(m_k, k_w), hmac_{k_m}(k, e)}\end{array}$$

$$\begin{array}{l}\mathsf{Unwrap(e)} : \\ \mathsf{h(n_w, k_w), enc(k, k_w), enc(k_m, k_w), hmac_{k_m}(k, e);\ unwrap(k_w)} \xrightarrow{\mathsf{new\ n}} \mathsf{h(n,k);\ e(n)}\end{array}$$

$$\begin{array}{ll}\mathsf{SEncrypt} : & \mathsf{h(n_e, k_e), k;\ encrypt(n_e)} \rightarrow \mathsf{enc(k, k_e)} \\ \mathsf{SDecrypt} : & \mathsf{h(n_e, k_e), enc(k, k_e);\ decrypt(k_e)} \rightarrow \mathsf{k}\end{array}$$

Fig. 4. Symmetric Key Management subset of the Eracom PKCS#11 API. $e(n)$ is a short-hand for "n is in object state e".

Eracom rewrite system. Consider the API rules given in Figure 4, derived from the symmetric key management commands of the Eracom ProtectServer [7]. Here, an HMAC is used to bind the attributes to the wrapped key, to prevent an attacker from re-importing several copies of a key with different attributes. Formally, we have the new type mac and the function symbol $\mathsf{hmac} : \mathsf{Key} \times \mathsf{Key} \times att \rightarrow \mathsf{mac}$. Although the proof of Lemma 1 carries over to this set of rules, Theorem 2 no longer holds, since the wrap rule gives an additional way of generating fresh keys (the key to be used as MAC-key). However, we can recover the result with a simple abstraction: if there is an attack, then there is an attack where the MAC-key generated by the wrap command is constant:

Proposition 8. *Given a derivation under the rules of Figure 4, we can map in the obvious way to a derivation under rules that are like those of Figure 4 apart from that the wrap rule always uses a constant mac-key* $\mathsf{m_K}$.

This gives us an abstraction, not an equivalence. It may in theory lead to false attacks, but it is sound for proofs since in our queries we have no disequalities on keys. In this model, we have:

Theorem 3. *In the* abstracted *Eracom rewrite system with attribute policy* $(\mathcal{E}, \emptyset)$*, if there is a derivation of a query* (T, L) *then there is a derivation of the same query using*

1. *at most* $1 + |\mathcal{E}| + 1$ *keys:*
   at most 1 *key in* Q_0*,*
   at most $|\mathcal{E}|$ *keys generated by* KeyGenerate*, and*
   at most 1 *key used by* Wrap*; and*
2. *at most* $|\mathcal{E}| \times (2 + |\mathcal{E}|)$ *handles.*

In fact we can recover an exact version (i.e. without the abstraction) of Theorem 2 for the Eracom rewrite system involving a slightly larger number of keys and handles, but due to lack of space we leave this for a future longer version of the paper.

5 Experiments

Having established the validity of model checking a small bounded model of our Eracom-based PKCS#11 API (Fig. 4), we can now investigate security properties for unbounded keys and handles. We assume an endpoint attribute policy $P = (\{ed, wu\}, \emptyset)$, where ed represents encrypt and decrypt and wu represents wrap and unwrap. All keys are assumed to be sensitive. SATMC includes the usual rules for encryption and decryption by known keys. First we investigate the stated property required of PKCS#11 in the specification [8, §7].

Definition 2. *A* known key *is a key* k *such that the intruder knows the plaintext value* k *and the intruder has a handle* $\mathsf{h}(\mathsf{n}, \mathsf{k})$.

Property 1. If an intruder starts with no known keys, he cannot obtain any known keys.

In the API of Figure 4, this property is verified by SATMC in 0.4 seconds. A further desirable property of such an API is that if a session key (i.e. an encryption/decryption key) is lost to the intruder by some means beyond the scope of the model, then no further keys are compromised.

Property 2. If an intruder starts with a known key k_i with handle $\mathsf{h}(\mathsf{n}_\mathsf{i}, \mathsf{k}_\mathsf{i})$, and $\mathsf{ed}(\mathsf{n}_\mathsf{i})$ is true, then he cannot obtain any further known keys.

Unfortunately this does not hold for the Eracom-based API. An attack is found by SATMC in 0.4 seconds. We give the attack in Figure 5. The intruder first wraps his key k_i, then fakes an HMAC for it using k_i as the MAC key. This allows him to re-import k_i as a wrap/unwrap key. One way to prevent this attack is to add the wrapping key inside the HMAC (see Figure 6). SATMC verifies this property in about 0.5 seconds.

Full details of our model checking experiments, including all relevant model files, are available at `http://www.lsv.ens-cachan.fr/~steel/pkcs11/`.

Initial knowledge: Handles $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, and $\mathsf{h}(\mathsf{n}_\mathsf{i}, \mathsf{k}_\mathsf{i})$. Key k_i. Attributes $\mathsf{ed}(\mathsf{n}_1), \mathsf{wu}(\mathsf{n}_2), \mathsf{ed}(\mathsf{n}_\mathsf{i})$. The HMAC key is k_3.

Trace:

Wrap: (ed)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, $\mathsf{h}(\mathsf{n}_\mathsf{i}, \mathsf{k}_\mathsf{i}) \rightarrow \mathsf{senc}(\mathsf{k}_\mathsf{i}, \mathsf{k}_2)$, $\mathsf{senc}(\mathsf{k}_3, \mathsf{k}_2)$, $\mathsf{hmac}_{\mathsf{k}_3}(\mathsf{k}_\mathsf{i}, \mathsf{ed})$
Unwrap: (wu)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, $\mathsf{senc}(\mathsf{k}_\mathsf{i}, \mathsf{k}_2)$, $\mathsf{senc}(\mathsf{k}_\mathsf{i}, \mathsf{k}_2)$, $\mathsf{hmac}_{\mathsf{k}_\mathsf{i}}(\mathsf{k}_\mathsf{i}, \mathsf{wu}) \rightarrow \mathsf{h}(\mathsf{n}_2, \mathsf{k}_\mathsf{i})$
Wrap: (ed)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_\mathsf{i})$, $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \rightarrow \mathsf{senc}(\mathsf{k}_1, \mathsf{k}_\mathsf{i})$, $\mathsf{senc}(\mathsf{k}_3, \mathsf{k}_\mathsf{i})$, $\mathsf{hmac}_{\mathsf{k}_3}(\mathsf{k}_1, \mathsf{ed})$
Decrypt:	k_i, $\mathsf{senc}(\mathsf{k}_1, \mathsf{k}_\mathsf{i}) \rightarrow \mathsf{k}_1$

Fig. 5. Lost session key attack

$\mathsf{Wrap:}$

$$\mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\mathsf{h}(\mathsf{x}_2,\mathsf{y}_2);\ \mathsf{wrap}(\mathsf{x}_1),\mathsf{A}(\mathsf{x}_2) \xrightarrow{\mathsf{new}\ \mathsf{m}_\mathsf{k}} \mathsf{enc}(\mathsf{y}_2,\mathsf{y}_1),\mathsf{enc}(\mathsf{m}_\mathsf{k},\mathsf{y}_1),\mathsf{hmac}_{\mathsf{m}_\mathsf{k}}(\mathsf{y}_2,\mathsf{A},\mathsf{y}_1)$$

$\mathsf{Unwrap:}$

$$\mathsf{h}(\mathsf{x}_1,\mathsf{y}_2),\mathsf{enc}(\mathsf{y}_1,\mathsf{y}_2),\mathsf{enc}(\mathsf{x}_\mathsf{m},\mathsf{y}_2),\mathsf{hmac}_{\mathsf{x}_\mathsf{m}}(\mathsf{y}_1,\mathsf{A},\mathsf{y}_2);\ \mathsf{unwrap}(\mathsf{x}_1) \xrightarrow{\mathsf{new}\ \mathsf{n}_1} \mathsf{h}(\mathsf{n}_1,\mathsf{y}_1);\ \mathsf{A}(\mathsf{n}_1)$$

Fig. 6. Revised Wrap/Unwrap Mechanism for the Eracom API

6 Conclusions

We have presented our framework for analysing PKCS#11 based APIs in an unbounded model. We described an attack on a version of the API used by Eracom, discovered using our model and the model checker SATMC. We suggested a fix and proved the secrecy of sensitive keys for the fixed version of the API, for unbounded numbers of keys, handles and command calls. An extension to asymmetric cryptography is our first priority, and then experiments with further proprietary implementations of PKCS#11.

We have explained how our work extends previous work by Delaune, Kremer and Steel [5]. There have been other attempts to analyse PKCS#11, but these were also for bounded models [9,10], and included further approximations such as monotonic global state. With our endpoint abstractions, we also obtain a monotonic global state, however we have formally justified this in the presence of a complete attribute policy. In fact, our analysis suggests that robust attribute policies can be reduced to static ones. There have also been proofs of security for other APIs, such as revised versions of the IBM Common Cryptographic Architecture, again in bounded models [3,4]. We plan to adapt our method to this API.

References

1. Armando, A., Compagna, L.: SAT-based model-checking for security protocols analysis. Int. J. Inf. Sec. 7(1), 3–32 (2008), `http://www.ai-lab.it/satmc`; Currently developed under the AVANTSSAR project, `http://www.avantssar.eu`
2. Clulow, J.: On the security of PKCS#11. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 411–425. Springer, Heidelberg (2003)
3. Cortier, V., Keighren, G., Steel, G.: Automatic analysis of the security of XOR-based key management schemes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 538–552. Springer, Heidelberg (2007)
4. Courant, J., Monin, J.-F.: Defending the bank with a proof assistant. In: Proceedings of the 6th International Workshop on Issues in the Theory of Security (WITS 2006), Vienna, Austria, March 2006, pp. 87–98 (2006)
5. Delaune, S., Kremer, S., Steel, G.: Formal analysis of PKCS#11. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008), Pittsburgh, PA, USA, pp. 331–344. IEEE Computer Society Press, Los Alamitos (2008)

6. Fröschle, S.: The insecurity problem: Tackling unbounded data. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF 2007), Venice, Italy, pp. 370–384. IEEE Computer Society Press, Los Alamitos (2007)
7. Krhovják, J.: PKCS #11 based APIs. Talk given at the Analysis of Security APIs Workshop (ASA-1) (July 2007), http://homepages.inf.ed.ac.uk/gsteel/asa/slides/
8. RSA Security Inc., v2.20. PKCS #11: Cryptographic Token Interface Standard (June 2004)
9. Tsalapati, E.: Analysis of PKCS#11 using AVISPA tools. Master's thesis, University of Edinburgh (2007)
10. Youn, P.: The analysis of cryptographic APIs using the theorem prover Otter. Master's thesis, Massachusetts Institute of Technology (2004)

Transformations between Cryptographic Protocols*

Joshua D. Guttman

The MITRE Corporation

Abstract. A transformation F between protocols associates the messages sent and received by participants in a protocol Π_1 with messages sent and received in some Π_2. Transformations are useful for modeling protocol design, protocol composition, and the services that protocols provide.

A protocol transformation determines a map from partial behaviors $\mathbb{A}_1$ of Π_1—which we call "skeletons"—to skeletons $F(\mathbb{A}_1)$ of Π_2. Good transformations should act as functors, preserving homomorphisms (information-preserving maps) from one Π_1-skeleton to another. Thus, if $H\colon \mathbb{A}_1 \mapsto \mathbb{A}_2$ is a homomorphism between Π_1-skeletons, then there should be a homomorphism $F(H)\colon F(\mathbb{A}_1) \mapsto F(\mathbb{A}_2)$ between their images in Π_2.

We illustrate protocol transformation by examples, and show that our definition ensures that transformations act as functors.

1 Introduction

A protocol transformation F from a protocol Π_1 to a protocol Π_2 maps message transmissions and receptions of roles of Π_1 to transmissions and receptions of roles of Π_2. F may be used to show how Π_1 and Π_2 exhibit one of a number of relations, among them:

- Π_1 may be a choreography—typically defined without cryptography—which the cryptography protocol Π_2 is intended to implement faithfully, despite the presence of malicious parties;
- Π_1 may be a stage in designing the fuller protocol Π_2;
- Π_1 may be a simplification of an implemented protocol Π_2 [12]; or
- Π_2 may be a composition of Π_1 with other protocols [2,8,10].

In this paper, our goals are to give:

1. a definition of transformations flexible enough for these purposes, which requires us to avoid giving an excessively syntactic criterion;
2. examples to show how it accommodates these purposes; and
3. three key results ensuring that protocol transformations are well-behaved.

* Supported by the MITRE-Sponsored Research program.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 107–123, 2009.

Skeletons and the homomorphisms between skeletons are central to our approach. We have previously used them to develop an algorithm for protocol analysis [6]; to prove this algorithm covers all the possibilities [5]; and to give a criterion ensuring that a protocol composition will respect the security goals achieved by the protocols being combined [8].

A *skeleton* $\mathbb{A}$ *describes* the behavior of the regular (non-adversarial) participants in some set of executions. It contains some regular behavior; the executions $\mathbb{A}$ describes must also contain this behavior. Transmission and reception events are related by a partial order expressing causal precedence. The skeleton may stipulate that some keys are assumed uncompromised, and that some keys or other (nonce-like) values are freshly generated. Some skeletons are *realized* in the sense that they contain all the regular behavior needed for a complete execution. This means that for every message that the skeleton says was delivered, either this message should itself have been previously transmitted, or else the adversary can synthesize it using its own creations and messages that were previously sent. Naturally, adversary behavior must respect the freshness and noncompromise assumptions of the skeleton. A protocol transformation F determines a map lifting skeletons $\mathbb{A}$ of Π_1 to skeletons $F(\mathbb{A})$ of Π_2.

A *homomorphism* is an information-preserving map $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$ between skeletons of one protocol. The executions that a skeleton $\mathbb{A}_0$ *describes* are all the *realized* skeletons $\mathbb{A}_r$ such that, for some homomorphism H, $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_r$. An information-preserving map $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$ cannot increase (loosen) the set of executions described; instead, H's target $\mathbb{A}_1$ should describe a subset of the executions described by its source $\mathbb{A}_0$. This follows from the fact that the composition of homomorphisms is a homomorphism.

The three theorems of this paper state:

1. A transformation acts as a functor (Thm. 1). If H is a homomorphism $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$ on skeletons of Π_1, F should determine a (unique) homomorphism between their images in Π_2, i.e. $F(H)\colon F(\mathbb{A}_0) \mapsto F(\mathbb{A}_1)$.
2. F, regarded as a functor on homomorphisms, does not introduce new ones (Thm. 2). If $G\colon F(\mathbb{A}_0) \mapsto F(\mathbb{A}_1)$ is a homomorphism on Π_2 skeletons that are the images of Π_1 skeletons $\mathbb{A}_0, \mathbb{A}_1$, then $G = F(H)$ for some $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$.
3. If $\mathbb{B}$ is any Π_2 skeleton, then there is a (unique) maximum $\mathbb{A}_1$ such that $F(\mathbb{A}_1)$ may be mapped injectively into $\mathbb{B}$ (Thm. 3). $\mathbb{A}_1$ defines the maximum Π_1 content that is contained in $\mathbb{B}$.

In all of these results, uniqueness is to within isomorphism.

Our criterion on transformations F involves, besides a few bookkeeping constraints, three main requirements. First, suppose a parameter of a role ρ_1 of Π_1 *originates* on its image $F(\rho_1)$. This means that it is transmitted without having previously been received. Then this parameter should also have originated on ρ_1. Second, is an analogous condition on parameters of a role ρ_1 of Π_1 that are simply transmitted on its image $F(\rho_1)$, rather than being used only as keys for encryption or decryption. Finally, suppose that two roles ρ_0, ρ_1 of Π_1 have a substitution instance s in common. Then their images $F(\rho_0), F(\rho_1)$ in Π_2

should have in common the image $F(s)$. This condition is necessary to ensure that $F(\rho_i)$ does not commit to one branch of a branching behavior while ρ_0 and ρ_1 are uncommitted.

Section 2 presents a variety of examples. In Section 3, we pause to summarize the current strand space treatment of the algebra of messages, skeletons, and homomorphisms, following [6,8]. Section 4 presents the main definition and shows that $F(\mathbb{A})$ is well determined, and Section 5 shows that homomorphisms are preserved. Section 6 mentions some key related work and concludes.

2 Some Examples

We discuss here several variants of a transaction in which a principal A answers a question asked by another principal B. First, in the choreography description [1], Fig. 1, A starts by sending a message saying that he is ready; B responds with a query message containing the parameter Q, representing the question; and then A branches. Either A decides to answer the question affirmatively, and sends the constant *yes*, or else A decides to answer the question negatively, and sends the constant *no*.

The local behaviors of the individual parties are of four kinds. Two of them, the answerer A's behaviors C_1, C_2, are shown in Fig. 2. We refer to the local sequence of actions of one principal in one run of the protocol—possibly incomplete—as a *strand*. The questioner B's strands C_3, C_4 are dual, interchanging message sends and receives. In C_1, the final message transmission is the affirmative form *yes* while in C_2, it is *no*. In either case, the query can carry any value of the parameter Q. A strand which is only partly complete—because only the first step or the first two steps have occurred—is the same whether it is a C_1 behavior or a C_2 behavior. The execution has not yet committed to one branch or the other if it has not yet performed the last step. When one observes only

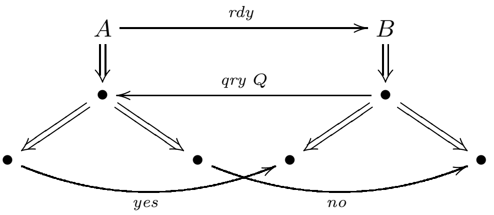


Fig. 1. A Question-Answering Choreography

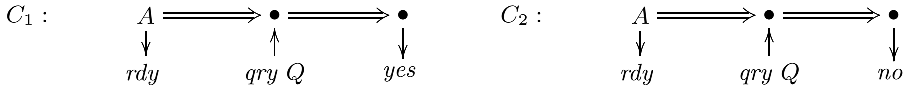


Fig. 2. Question-Answering Choreography: A's Parametric Strands

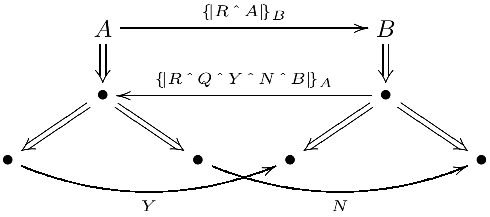


Fig. 3. Question-Answering Cryptoprotocol

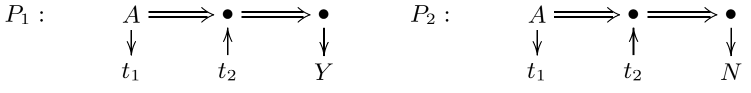
the first two nodes of any of these behaviors, one cannot distinguish whether the last node will be affirmative or negative.

The matching behaviors C_3, C_4 are not shown since they are simply the duals.

A may later bill B a fee for this service. With so valuable a service, there are of course security considerations. A may not want to disclose to an adversary sniffing the network the answer to B's question; B may not want to disclose what question he is asking; and both parties may wish to ensure that the adversary cannot alter B's question Q to a different question Q', or alter A's answer from affirmative to negative or the reverse. To defeat these (standard) attacks, we may implement our choreography using a cryptographic protocol (Fig 3). Here, each principal is assumed to know a public (asymmetric) encryption key to which only the partner holds the matching decryption key; we write t encrypted in a form that only principal P can read as $\{|t|\}_P$. The parameters R, Y, N are nonces, i.e. randomly chosen bitstrings that are extremely unlikely to be used more than once. The last message is unencrypted. A transmits the first of the two nonces Y and N created by B to indicate the answer *yes*, and the second to indicate the answer *no*. This conveys the answer to B, while conveying nothing whatever to any principal that did not prepare $\{|R\hat{\ }Q\hat{\ }Y\hat{\ }N\hat{\ }B|\}_A$ and cannot decrypt it. The nonces themselves are perfectly arbitrary.

Each local behavior of the principal A is summarized in one of the strands in Fig. 4. Each one has the parameters A, B, R, Q, Y, N, and the *instances* of each strand are generated by substituting each possible name, nonce, or question for these parameters, respecting types. Again, the strands P_3, P_4 of B are dual.

Looking at the strands of Figs. 2, 4, we can see that we could perform a mapping in either direction. One could map the transmission and reception events of the cryptoprotocol strands to send and receive events in the choreography, discarding superfluous parameters. In this scheme, each j^{th} node of P_i would map to



$$t_1 = \{|R\hat{\ }A|\}_B \qquad t_2 = \{|R\hat{\ }Q\hat{\ }Y\hat{\ }N\hat{\ }B|\}_A$$

Fig. 4. Question-Answering Cryptoprotocol: Parametric Strands

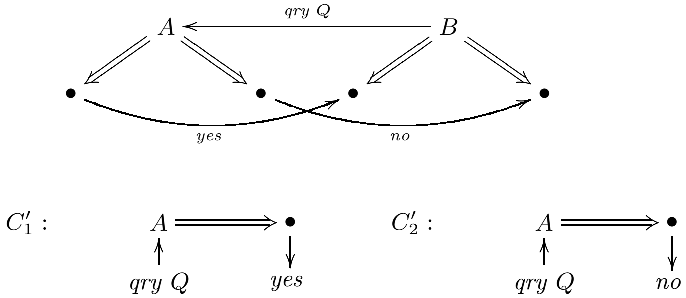


Fig. 5. A Question-Answering Service and Its Strands

the j^{th} node of C_i. Alternatively, one could map the events of the choreography to events on strands of the cryptoprotocol, allowing the additional parameters to be freely chosen. In this scheme, each j^{th} node of C_i would map to the j^{th} node of P_i.

In either case, we will map nodes on answerer strands at one level to nodes on answerer strands at the other level, and questioner nodes to questioner nodes at the other level. We will map affirmative nodes at one level to affirmative nodes at the other, and likewise for negative nodes. Along each strand, each successive transmission or reception must be associated with the corresponding transmission or reception of the strand at the other level. Thus, our only choice is whether to map from P_i to the C_i, or vice versa.

This example is, however, peculiar in that the two protocols have corresponding strands of the same length. Consider next the choreography of Fig. 5. It is more natural than the choreography of Fig. 1, since the questioner acts as client and the answerer acts as server; the *rdy* message at the beginning of Fig. 1 inverts the client-server roles. However, in Fig. 5, we can no longer map the j^{th} node of C'_i to the j^{th} node of P_i. Instead, it corresponds to the $j+1^{\text{st}}$ node of P_i. For this same reason, we cannot speak of a transformation in the other direction; since the $C_i \mapsto P_i$ mapping is not surjective, the $P_i \mapsto C_i$ mapping would not be total.

Moreover, we can also think of protocol transformations where both source and target protocols involve cryptography. For instance, we can regard the cryptoprotocol of Fig. 3 as derived from the choreography of Fig. 5 by a succession of steps that progressively introduce suitable cryptography. As a first step, we may introduce the first readying message of Fig. 3, with the nonce R returned piggybacked with the question Q. We include these two elements within an encryption so that an adversary cannot reassociate the nonce R with a different question Q'. Thus, we obtain the intermediate form given in Fig. 6. The first two messages provide an "outgoing test" [6] that allows A to authenticate this question Q as having been submitted by B. If the private decryption keys of A, B are uncompromised, then only B could liberate R from $\{\!|R\,\hat{}\,A|\!\}_B$, and moreover no third party could liberate R from any other message $\{\!|R\,\hat{}\,Q'\,\hat{}\,B|\!\}_A$. Thus, no third party could have repackaged R with the Q that A actually received.

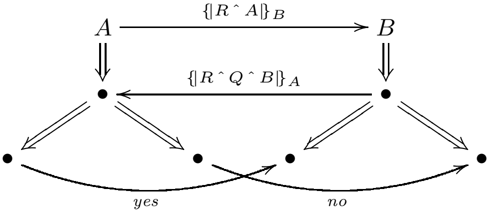

$\{\!|R \,\hat{}\, A|\!\}_B$
A
B
$\{\!|R \,\hat{}\, Q \,\hat{}\, B|\!\}_A$
yes
no

Fig. 6. From Choreography to Cryptoprotocol, Step 1

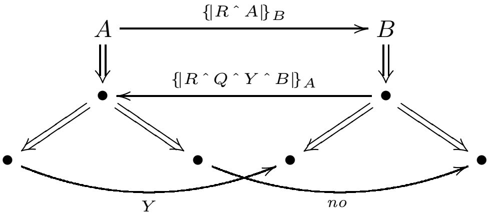

$\{\!|R \,\hat{}\, A|\!\}_B$
A
B
$\{\!|R \,\hat{}\, Q \,\hat{}\, Y \,\hat{}\, B|\!\}_A$
Y
no

Fig. 7. From Choreography to Cryptoprotocol, Step 2

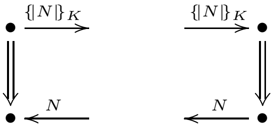


Fig. 8. The Outgoing (Nonce) Authentication Test

Using the outgoing test idea again, B can supply a new nonce Y within the second message $\{\!|R \,\hat{}\, Q \,\hat{}\, Y \,\hat{}\, B|\!\}_A$; A may liberate this nonce to indicate an affirmative answer to Q (Fig 7). Repeating this idea with another nonce N for the negative case completes the protocol design process for Fig. 3.

Indeed, we can express the abstract schema of an "outgoing test" [6], in which a nonce is transmitted in encrypted form and received back later outside of that encryption, as a sort of germ protocol (Fig. 8). It allows the sender to infer that either the decryption key is compromised, or else a regular participant has received the encrypted unit and transformed its content. If we perform a renaming here, mapping N to R and K to $\mathsf{pubk}(B)$, then we can map the left strand to the first two nodes on the left of Fig. 6 and the right strand to the first two nodes on the right. This appears to show Fig. 6 as a sort of "join" of the choreography of Fig. 5 with an instance of the outgoing test protocol. We can now repeat this process with a different renaming of Fig. 8 to produce Fig. 7 as a successive join, producing the final cryptoprotocol in a similar third step. These maps appear to characterize how the protocol of Fig. 4 achieves its goals.

3 Messages, Protocols, Skeletons

In this section, we provide an overview of the current strand space framework; cf. [8] or the extended version of [6] for more detail. Let $\mathfrak{A}_0$ be an algebra equipped with some operators and a set of homomorphisms $\eta\colon \mathfrak{A}_0 \to \mathfrak{A}_0$. We call members of $\mathfrak{A}_0$ *atoms.*

For the sake of definiteness, we will assume here that $\mathfrak{A}_0$ is the disjoint union of infinite sets of *nonces*, *atomic keys*, *names*, and *texts.* The operator $\mathsf{sk}(a)$ maps names to (atomic) signature keys, and K^{-1} maps an asymmetric atomic key to its inverse, and a symmetric atomic key to itself. Homomorphisms η are maps that respect sorts, and act homomorphically on $\mathsf{sk}(a)$ and K^{-1}.

Let X be an infinite set disjoint from $\mathfrak{A}_0$; its members—called *indeterminates*—act like unsorted variables. $\mathfrak{A}$ is freely generated from $\mathfrak{A}_0 \cup X$ by two operations: encryption $\{\!|t_0|\!\}_{t_1}$ and tagged concatenation $\mathit{tag}\ t_0\,\hat{}\,t_1$, where the tags tag are drawn from some set TAG. For a distinguished tag nil, we write $\mathit{nil}\ \ t_0\,\hat{}\,t_1$ as $t_0\,\hat{}\,t_1$ with no tag. In $\{\!|t_0|\!\}_{t_1}$, a non-atomic key t_1 is a symmetric key. Members of $\mathfrak{A}$ are called *messages.*

A homomorphism $\alpha = (\eta, \chi)\colon \mathfrak{A} \to \mathfrak{A}$ consists of a homomorphism η on atoms and a function $\chi\colon X \to \mathfrak{A}$. It is defined for all $t \in \mathfrak{A}$ by the conditions:

$$\begin{array}{llr} \alpha(a) = \eta(a), & \text{if } a \in \mathfrak{A}_0 & \alpha(\{\!|t_0|\!\}_{t_1}) = \{\!|\alpha(t_0)|\!\}_{\alpha(t_1)} \\ \alpha(x) = \chi(x), & \text{if } x \in X & \alpha(\mathit{tag}\ t_0\,\hat{}\,t_1) = \mathit{tag}\ \alpha(t_0)\,\hat{}\,\alpha(t_1) \end{array}$$

Thus, atoms serve as typed variables, replaceable only by other values of the same sort, while indeterminates x are untyped. Indeterminates x serve as blank slots, to be filled by any $\chi(x) \in \mathfrak{A}$. Indeterminates and atoms are jointly *parameters.*

This $\mathfrak{A}$ has the most general unifier property, which we will rely on. That is, suppose that for $v, w \in \mathfrak{A}$, there exist α, β such that $\alpha(v) = \beta(w)$. Then there are α_0, β_0, such that $\alpha_0(v) = \beta_0(w)$, and whenever $\alpha(v) = \beta(w)$, then α and β are of the forms $\gamma \circ \alpha_0$ and $\gamma \circ \beta_0$.

Messages are abstract syntax trees in the usual way:

1. Let ℓ and r be the partial functions such that for $t = \{\!|t_1|\!\}_{t_2}$ or $t = \mathit{tag}\ t_1\,\hat{}\,t_2$, $\ell(t) = t_1$ and $r(t) = t_2$; and for $t \in \mathfrak{A}_0$, ℓ and r are undefined.
2. A *path* p is a sequence in $\{\ell, r\}^*$. We regard p as a partial function, where $\langle\rangle = \mathsf{Id}$ and $\mathsf{cons}(f, p) = p \circ f$. When the rhs is defined, we have: 1. $\langle\rangle(t) = t$; 2. $\mathsf{cons}(\ell, p)(t) = p(\ell(t))$; and 3. $\mathsf{cons}(r, p)(t) = p(r(t))$.
3. *p traverses a key edge* in t if $p_1(t)$ is an encryption, where $p = p_1{}^\frown\langle r\rangle{}^\frown p_2$.
4. *p traverses a member of S* if $p_1(t) \in S$, where $p = p_1{}^\frown p_2$ and $p_2 \neq \langle\rangle$.
5. *t_0 is an ingredient of t*, written $t_0 \sqsubseteq t$, if $t_0 = p(t)$ for some p that does not traverse a key edge in t.
6. *t_0 appears in t*, written $t_0 \ll t$, if $t_0 = p(t)$ for some p.

Strands and origination. A single local session of a protocol at a single principal is a *strand*, containing a linearly ordered sequence of transmissions and receptions that we call *nodes.* In Figs. 2, 4, and 5, the vertical columns of nodes connected by double arrows $\Rightarrow$ are strands.

We write $s \downarrow i$ for the i^{th} node on strand s, using 1-based indexing. We write $n \Rightarrow m$ when n, m are successive nodes on the same strand, i.e. when for some s, i, $n = s \downarrow i$ and $m = s \downarrow i + 1$. We write $\mathsf{msg}(n)$ for the message sent or received on the node n.

A message t_0 *originates* at a node n_1 if (1) n_1 is a transmission node; (2) $t_0 \sqsubseteq \mathsf{msg}(n_1)$; and (3) whenever $n_0 \Rightarrow^+ n_1$, $t_0 \not\sqsubseteq \mathsf{msg}(n_0)$.

Thus, t_0 originates when it was transmitted without having been either received or transmitted previously on the same strand. Values assumed to originate only on one node in an execution—*uniquely originating* values—formalize the idea of freshly chosen, unguessable values. Values assumed to originate nowhere may be used to encrypt or decrypt, but are never sent as message ingredients. They are called *non-originating* values. For a non-originating value K, $K \not\sqsubseteq t$ for any transmitted message t. However, $K \ll \{|t_0|\}_K \sqsubseteq t$ possibly, which is why we distinguish $\sqsubseteq$ from $\ll$. See [11,6] for more details.

In the tree model of messages, to apply a homomorphism, we walk through, copying the tree, but inserting $\alpha(a)$ every time an atom a is encountered, and inserting $\alpha(x)$ every time that an indeterminate x is encountered.

Protocols. A *protocol* Π is a finite set of strands, representing the roles of the protocol. Four of the roles of the yes-no cryptoprotocol are the strands shown in Fig. 4. Their instances result by replacing A, B, R, Q, Y, N, etc., by any names, nonce, question, etc. The fifth role is the *listener* role $\mathsf{Lsn}[y]$ with a single reception node in which y is received. The instances of $\mathsf{Lsn}[y]$ are used to document that values are available without cryptographic protection. For instance, $\mathsf{Lsn}[K]$ would document that K is compromised. Every protocol contains the role $\mathsf{Lsn}[y]$.

Indeterminates represent syntactically unconstrained messages received from protocol peers, or passed down as parameters from higher-level protocols. Thus, we require an indeterminate to be received as an ingredient before appearing in any other way:

If n_1 is a node on $\rho \in \Pi$, with an indeterminate $x \ll \mathsf{msg}(n_1)$,
then $\exists n_0$, $n_0 \Rightarrow^* n_1$, where n_0 is a reception node and $x \sqsubseteq \mathsf{msg}(n_0)$.

A principal executing a role such as P_3 in Fig. 4 may be partway through its run; for instance, it may have executed the first reception event and first transmission, without "yet" having executed the final reception event. Thus, it can not yet be distinguished from the first two nodes of an instance of role P_4. When $n_1 \Rightarrow n_2$ is the beginning of a strand $\alpha(P_3)$, we also regard it as an instance of the role P_4, since nothing that has happened shows that it is not:

Definition 1. *Node n is a* role node *of Π if $n = \rho \downarrow j$ lies on some $\rho \in \Pi$.*

Let $\rho \downarrow j$ be a role node of Π. Node m_j is an instance *of $\rho \downarrow j$ if, for some homomorphism α, the strand of m_j, up to m_j, takes the form: $\alpha(\rho \downarrow 1) \Rightarrow \ldots \Rightarrow \alpha(\rho \downarrow j) = m_j$.* □

That is, messages and their directions—transmission or reception—must agree up to node j. However, any remainders of the two strands beyond node j are unconstrained. They need not be compatible. When a protocol allows a principal to decide between different behaviors after step j, based on the message contents of their run, then this definition represents branching [7,9]. At step j, one doesn't yet know which branch will be taken.

If s is a strand, then $t_0 \ll s$ iff for some j, $t_0 \ll \mathsf{msg}(s \downarrow j)$; and similarly $t_0 \sqsubseteq s$ iff for some j, $t_0 \sqsubseteq \mathsf{msg}(s \downarrow j)$.

Skeletons. A *skeleton* $\mathbb{A}$ consists of (possibly partially executed) role instances, i.e. a finite set of nodes, $\mathsf{nodes}(\mathbb{A})$, with two additional kinds of information:

1. A partial ordering $\preceq_{\mathbb{A}}$ on $\mathsf{nodes}(\mathbb{A})$;
2. Finite sets $\mathsf{unique}_{\mathbb{A}}, \mathsf{non}_{\mathbb{A}}$ of atomic values assumed uniquely originating and respectively non-originating in $\mathbb{A}$.

$\mathsf{nodes}(\mathbb{A})$ and $\preceq_{\mathbb{A}}$ must respect the strand order, i.e. if $n_1 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \Rightarrow n_1$, then $n_0 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \preceq_{\mathbb{A}} n_1$. If $a \in \mathsf{non}_{\mathbb{A}}$, then a must originate nowhere in $\mathsf{nodes}(\mathbb{A})$, though a or a^{-1} may be the key encrypting some $e \ll \mathsf{msg}(n)$ for $n \in \mathsf{nodes}(\mathbb{A})$. If $a \in \mathsf{unique}_{\mathbb{A}}$, then a must originate at most once in $\mathsf{nodes}(\mathbb{A})$.

$\mathbb{A}$ is *realized* if it is a possible run without additional activity of *regular* participants; i.e., for every reception node n, the adversary can construct $\mathsf{msg}(n)$ via the Dolev-Yao adversary actions,[1] using as inputs:

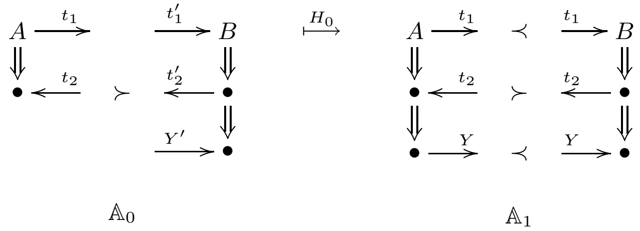
1. all messages $\mathsf{msg}(m)$ where $m \prec_{\mathbb{A}} n$ and m is a transmission node;
2. any atomic values a such that $a \notin (\mathsf{non}_{\mathbb{A}} \cup \mathsf{unique}_{\mathbb{A}})$, or such that $a \in \mathsf{unique}_{\mathbb{A}}$ but a originates nowhere in $\mathbb{A}$.

Two skeletons $\mathbb{A}_0, \mathbb{A}_1$ are shown in Fig. 9. They are skeletons of the protocol shown in Fig. 7, i.e. the second step in the derivation of the Question-Answering Cryptoprotocol. Of these skeletons, only $\mathbb{A}_1$ is realized. It is a realized skeleton because every message to be received may be derived by the adversary from messages previously sent. In fact, every message received was itself previously sent: i.e. the trivial adversary derivation suffices. However, $\mathbb{A}_0$ is not realized. Y' is not derivable from its predecessors in $\mathbb{A}_0$, since the adversary can not use K_A^{-1}, nor re-originate Y'.

The initiator strand in $\mathbb{A}_0$ is only of height 2, rather than 3, so that it has not yet committed to branch to the affirmative or negative answer. Thus, it is a common instance of both initiator strands of its protocol. Whether we regard the not-yet-occurred third node as an affirmative one or a negative node makes no difference. Indeed, the map on skeletons that replaces the first two nodes of an affirmative strand by the first two nodes of a negative strand is an isomorphism.

Given any skeleton $\mathbb{A}_0$, assume that $\psi \colon \mathsf{nodes}_{\mathbb{A}_0} \to \mathsf{nodes}_{\mathbb{A}_1}$. $[\psi, \alpha]$ is an equivalence class of pairs ψ', α'. A pair ψ', α' is in $[\psi, \alpha]$ if (1) $\psi' = \psi$; and (2) $\alpha(a) = \alpha'(a)$, for every a such that $a \ll \mathsf{msg}(n)$ for any $n \in \mathsf{nodes}(\mathbb{A}_0)$. This definition for $[\psi, \alpha]$ implies that the action of α on atoms not mentioned in $\mathbb{A}_0$ is irrelevant.

[1] The Dolev-Yao adversary actions are: concatenating messages and separating the pieces of a concatenation; encrypting a given plaintext using a given key; and decrypting a given ciphertext using the matching decryption key.



$$t_1 = \{\!|R\,\hat{}\,A|\!\}_B \qquad t_1' = \{\!|R'\,\hat{}\,A|\!\}_B$$
$$t_2 = \{\!|R\,\hat{}\,Q\,\hat{}\,Y\,\hat{}\,B|\!\}_A \qquad t_2' = \{\!|R'\,\hat{}\,Q'\,\hat{}\,Y'\,\hat{}\,B|\!\}_A$$

$$\mathsf{unique}_{\mathbb{A}_0} = \{Y'\} \qquad \mathsf{unique}_{\mathbb{A}_1} = \{Y\}$$
$$\mathsf{non}_{\mathbb{A}_0} = \{K_A^{-1}\} \qquad \mathsf{non}_{\mathbb{A}_1} = \{K_A^{-1}, K_B^{-1}\}$$

Fig. 9. Homomorphism $H_0 = [\psi_0, \alpha_0] \colon \mathbb{A}_0 \mapsto \mathbb{A}_1$

Definition 2. *Let α be a homomorphism on messages of $\mathfrak{A}$, and $\psi\colon \mathsf{nodes}_{\mathbb{A}_0} \to \mathsf{nodes}_{\mathbb{A}_1}$, for skeletons $\mathbb{A}_0, \mathbb{A}_1$. $H = [\psi, \alpha]$ is a* skeleton homomorphism, *written $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$, if:*

1*a*. *For all $n \in \mathbb{A}_0$, $\mathsf{msg}(\psi(n)) = \alpha(\mathsf{msg}(n))$*
1*b*. *$\psi(n)$ is a transmission node iff n is a transmission node;*
1*c*. *ψ acts strand-by-strand; i.e.*

$$\forall s\ \exists s'\ \forall j\ .\ s \downarrow j \in \mathsf{nodes}(\mathbb{A}) \quad \textit{implies} \quad \psi(s \downarrow j) = s' \downarrow j;$$

2. *$n \preceq_{\mathbb{A}_0} m$ implies $\psi(n) \preceq_{\mathbb{A}_1} \psi(m)$;*
3. *$\alpha(\mathsf{non}_{\mathbb{A}_0}) \subseteq \mathsf{non}_{\mathbb{A}_1}$;*

4*a*. *$\alpha(\mathsf{unique}_{\mathbb{A}_0}) \subseteq \mathsf{unique}_{\mathbb{A}_1}$;*
4*b*. *If a originates at $n \in \mathsf{nodes}_{\mathbb{A}_0}$ for $a \in \mathsf{unique}_{\mathbb{A}_0}$, then $\alpha(a)$ originates at $\psi(n)$.*

H is an isomorphism if (as usual) for some $G\colon \mathbb{A}_1 \mapsto \mathbb{A}_0$, $G \circ H = Id_{\mathbb{A}_0}$.

We sometimes write $H(n)$ for $\psi(n)$ or $H(a)$ for $\alpha(a)$, when $H = [\psi, \alpha]$. As an example of a homomorphism between skeletons, consider the map H_0 between skeletons shown in Fig. 9.

In H_0, $\alpha_0(R') = R$, $\alpha_0(Q') = Q$, and $\alpha_0(Y') = Y$; the remaining parameters are unchanged. The function ψ_0 on nodes maps successive nodes on the left strand of $\mathbb{A}_0$ to their counterparts on the left strand of $\mathbb{A}_1$; it maps nodes on the right strand of $\mathbb{A}_0$ to their counterparts on the right strand of $\mathbb{A}_1$. It also enriches the ordering. Specifically, the topmost nodes are ordered in $\mathbb{A}_1$ but not in $\mathbb{A}_0$; and the second node of the left strand precedes the last node of the right strand in $\mathbb{A}_1$, although they are not ordered in $\mathbb{A}_0$. Moreover, $\alpha_0(\mathsf{unique}_{\mathbb{A}_0}) = \mathsf{unique}_{\mathbb{A}_1}$, although $\alpha_0(\mathsf{non}_{\mathbb{A}_0}) \subsetneq \mathsf{non}_{\mathbb{A}_1}$.

When, for a homomorphism $H = [\psi, \alpha] \colon \mathbb{A} \mapsto \mathbb{A}'$, ψ is an injective function from nodes of $\mathbb{A}$ to nodes of $\mathbb{A}'$, we call H a *node-injective homomorphism.*

We proved in [6] that if $H\colon \mathbb{A} \mapsto \mathbb{A}'$ and $G\colon \mathbb{A}' \mapsto \mathbb{A}$ are node-injective, then $\mathbb{A}$ and $\mathbb{A}'$ are isomorphic. Thus, we write $\mathbb{A} \leq_N \mathbb{A}'$ to mean that for some node-injective H, $H\colon \mathbb{A} \mapsto \mathbb{A}'$. Regarding $\leq_N$ as a relation on skeletons factored by isomorphism, it is a well-founded partial order. In fact, there are only finitely many isomorphism classes below the isomorphism class of any skeleton $\mathbb{A}$.

4 Transformations

We define our notion of transformation via five clauses. Clauses 1–3 are simple bookkeeping requirements. Clause 4 says that a transformation respects origination and $\sqsubseteq$.

Clause 5 says that a transformation respects the instances of roles in the sense of Def. 1. Essentially, Clause 5 says that the transformed protocol does not commit to one branch any earlier than the source, whenever the source roles can branch.

Definition 3. *Suppose that Π_1 and Π_2 are protocols, and F is a map from the strands $\rho_1 \in \Pi_1$ to pairs $F(\rho_1) = \rho_2, g$, where $\rho_2 \in \Pi_2$ and $g\colon \mathbb{N} \to \mathbb{N}$ is a function on natural numbers.*

F is a protocol transformation *iff, for all $\rho_1 \in \Pi_1$, letting $F(\rho_1) = \rho_2, g$:*

1. *g is an increasing function, i.e. $i < j$ implies $g(i) < g(j)$.*
2. *$g(\mathsf{length}(\rho_1)) \leq \mathsf{length}(\rho_2)$.*
3. *$\rho_1 \downarrow j$ is a transmission node [resp. a reception node] iff $\rho_2 \downarrow g(j)$ is a transmission node [resp. a reception node].*
4. *For each role node $n = \rho_1 \downarrow i$, and each parameter a such that $a \ll \mathsf{msg}(n)$:*
   - *(a) If a originates on $\rho_2 \downarrow k$, with $k \leq g(j)$, then a originates on some $\rho_1 \downarrow j'$ with $j' \leq j$; and*
   - *(b) If $a \sqsubseteq \mathsf{msg}(\rho_2 \downarrow k)$, with $k \leq g(j)$, then $a \sqsubseteq \mathsf{msg}(\rho_1 \downarrow j')$ with $j' \leq j$.*
5. *Suppose for some ρ_1, ρ_1' and α, α', and all $j \leq k$,*

   $$\alpha(\rho_1 \downarrow j) = \alpha'(\rho_1' \downarrow j).$$

   Let $F(\rho_1') = \rho_2', g'$. For every $i \leq k$, $g(i) = g'(i)$; and for every $j \leq g(k)$,

   $$\alpha(\rho_2 \downarrow j) = \alpha'(\rho_2' \downarrow j).$$

Node n is an image of *m iff for some ρ_1, α, j, letting $F(\rho_1) = \rho_2, g$, we have $m = \alpha(\rho_1 \downarrow j)$ and $n = \alpha(\rho_2 \downarrow g(j))$.* □

We use the indefinite article, *an* image of m, because there are many substitutions α' such that $m = \alpha'(\rho_1 \downarrow j)$, i.e. all those that differ only for parameters not appearing in $\rho_1 \downarrow j$. For instance, in Fig. 5, the parameters of $C_2' \downarrow 1$ are A, B, Q, so this node is unaffected by the action of α' on R, Y, N, and unaffected by the choice of the encryption keys $\mathsf{pubk}(A), \mathsf{pubk}(B)$. However, different instances of $P_2 \downarrow 2$ in Fig. 4 result as these parameters vary.

According to this definition, we have numerous transformations among the examples given in Section 2. The two-step choreography service of Fig. 5 may be transformed into any of the other choreographies and protocols of Figs. 1–7. Each j^{th} node of the source is mapped to the $j+1^{\text{th}}$ node of the target, so that the functions $g(j) = j+1$. The first "ready" message is not in the range of the transformations.

Indeed, an alternate transformation maps the affirmative strands of Fig. 5 to negative strands of the targets; this represents an inversion of the convention about the meanings of the outcomes. Indeed, the definition also allows the "nonsensical" maps that send affirmative initiator strands to negative responder strands, and negative initiator strands to affirmative responder strands. These nonsensical transformations, however, have the property that the image of a well-formed choreography execution is not a realized skeleton of the crypto protocol.

All of the three-step protocols have transformations to all of the others, which in some cases introduce additional parameters and cryptographic structure; in the reverse cases, the transformation "forgets" parameters. If P_1 and P_1 sent their first messages in incompatible forms, then Clause 5 would imply that the map from the choreography in Fig. 1 to the resulting cryptoprotocol would not be a transformation. It would force commitment to one branch too early. This corresponds to the (pointless) service in which A allows B to ask a question to which the answer is *yes*, and then answers *yes*, or allows B to ask a question to which the answer is *no*, and then answers *no*.

Clause 4 prohibits the map from Fig. 7 to a variant of Fig. 4 in which the public or private key of A is included in one of the messages as part of its plaintext. It is a parameter of the source, while not being an ingredient in the plaintext of the messages, and Clause 4(b) requires that this be preserved under transformation. Likewise, a variant of Fig. 4 in which the responder originates R in message 2 without having received it in message 1 would violate Clause 4(a).

When a transformation introduces parameters, then nodes of the source protocol have many possible images under the transformation. Naturally some choices may be unnecessarily constraining, when they select values for the newly introduced parameters that equal other parameters unnecessarily.

Suppose that X is a set of parameters, e.g. the parameters appearing in $\rho_1 \downarrow j$. Some parameters of $\rho_2 \downarrow g(j)$ may not be in X, for instance R if $\rho_2 = P_2$ and $j = 1$. Among substitutions α' that agree with α on X, some are unnecessarily specific, for instance those that map two parameters $\notin X$ to the same value. Selecting $\alpha'(Y) \neq \alpha'(N)$ would be more general than forcing $\alpha'(Y) = \alpha'(N)$. Likewise, forcing a parameter not in X to agree with one in X discards generality, e.g. forcing $\alpha'(Y) = \alpha'(Q)$.

Let α_0 be a substitution that agrees with α on X, and where $\alpha_0(x) = \alpha_0(x')$ implies $x, x' \in X$ or $x = x'$. Then α_0 is maximally general among substitutions that agree with α on X, in the following sense. If α_1 is any substitution agreeing with α on X, then any $\beta \circ \alpha_1 = \gamma \circ \alpha_0$ for exactly one γ. As a consequence, if α_1 also satisfies the same property as α_0, then α_0 and α_1 differ by a renaming.

Thus, for any set of parameters X, α_0 is *universal for* X if $\alpha_0(x) = \alpha_0(x')$ implies $x, x' \in X$ or $x = x'$. Such an α is however unique only up to isomorphism, and the same will remain true for our functions to lift skeletons $\mathbb{A}$. That is, $F(\mathbb{A})$ is determined only to within isomorphism.

In some circumstances (cf. e.g. [13]), it would be desirable to relax the order preserving requirement of Def. 3, Clause 1. In this case, letting

$$g^+(k) = \max_{1 \le i \le k} g(i),$$

we would rewrite Clause 2 as $g^+(\mathsf{length}(\rho_1)) \le \mathsf{length}(\rho_2)$. Similarly, the occurrences of $g(j)$ in Clause 4(a) and (b) would become $g^+(j)$, and the quantification over all $i \le g(k)$ in Clause 5 becomes $i \le g^+(k)$. The main results of Section 5 are not substantially changed.

A *strand of* $\mathbb{A}$ is any s where $n \in \mathsf{nodes}(\mathbb{A})$, for at least s's first node $n = s \downarrow 1$.

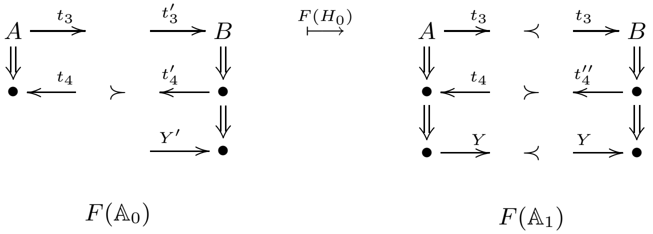
Definition 4. *Let F transform Π_1 to Π_2, and let $\mathbb{A}$ be a Π_1-skeleton.*

1. *A Π_2-skeleton $\mathbb{B}$ is an F-*image *of $\mathbb{A}$ iff there is a bijection φ between strands of $\mathbb{A}$ and strands of $\mathbb{B}$ such that:*
   (a) For every strand s of $\mathbb{A}$, letting $s = \alpha(\rho_1)$ and $F(\rho_1) = \rho_2, g$:
   $$\varphi(s) = \alpha(\rho_2) \quad \textit{and} \quad \mathsf{height}_{\mathbb{B}}(\varphi(s)) = g(\mathsf{height}_{\mathbb{A}}(s));$$
   (b) If $s \downarrow j \preceq_{\mathbb{A}} s' \downarrow k$, then $(\varphi(s) \downarrow g(j)) \preceq_{\mathbb{B}} (\varphi(s') \downarrow g'(k))$, where
   $$\begin{array}{ll} s = \alpha(\rho_1) & F(\rho_1) = \rho_2, g \\ s' = \alpha'(\rho_1') & F(\rho_1') = \rho_2', g'. \end{array}$$
   (c) Uniquely originating and non-originating values are preserved:
   $$\mathsf{unique}_{\mathbb{A}} \subseteq \mathsf{unique}_{\mathbb{B}} \textit{ and } \mathsf{non}_{\mathbb{A}} \subseteq \mathsf{non}_{\mathbb{B}}$$
2. *We write $F(\mathbb{A}) = \mathbb{B}$ if and only if $\mathbb{B} \le_N \mathbb{B}'$, for every F-image $\mathbb{B}'$ of $\mathbb{A}$.*

Lemma 1. *Let F transform Π_1 to Π_2, and let $\mathbb{A}$ be a Π_1-skeleton. There is—to within isomorphism—a unique $\mathbb{B}$ such that $F(\mathbb{A}) = \mathbb{B}$.*

Proof sketch. We may regard the strands that contribute nodes to $\mathbb{A}$ as a family $\{\alpha_i(\rho_1^i)\}_{i \in I}$, where for each a representation has been chosen, as an instance of a particular role ρ_1^i under a particular substitution α_i. In some cases, more than one choice of role ρ_1^i may be compatible with the nodes that actually appear to $\mathbb{A}$, which may be only an initial segment of the whole role. Definition 3, Clause 5 ensures that this choice cannot affect the outcome. Moreover, α_i may freely vary for all parameters that do not appear in nodes of this segment of ρ_1^i. By the discussion above, we may require that the α_i are chosen such that:

1. for atoms or indeterminates v not appearing in ρ_1^i, $\alpha_i(v) = \alpha_j(w)$ implies $w = v$ and $i = j$;
2. for atoms a not appearing in ρ_1^i, $\alpha_i(a) \notin \mathsf{unique}_{\mathbb{A}} \cup \mathsf{non}_{\mathbb{A}}$;
3. for indeterminates x not appearing in ρ_1^i, $\alpha_i(x)$ is an indeterminate.



$$t_3 = \{\!|R\,\hat{}\,A|\!\}_B \qquad t'_3 = \{\!|R'\,\hat{}\,A|\!\}_B$$

$$t_4 = \{\!|R\,\hat{}\,Q\,\hat{}\,Y\,\hat{}\,N\,\hat{}\,B|\!\}_A \qquad t'_4 = \{\!|R'\,\hat{}\,Q'\,\hat{}\,Y'\,\hat{}\,N'\,\hat{}\,B|\!\}_A$$

$$t''_4 = \{\!|R\,\hat{}\,Q\,\hat{}\,Y\,\hat{}\,N'\,\hat{}\,B|\!\}_A$$

$$\mathsf{unique}_{F(\mathbb{A}_0)} = \{Y'\} \qquad \mathsf{unique}_{F(\mathbb{A}_1)} = \{Y\}$$

$$\mathsf{non}_{F(\mathbb{A}_0)} = \{K_A^{-1}\} \qquad \mathsf{non}_{F(\mathbb{A}_1)} = \{K_A^{-1}, K_B^{-1}\}$$

Fig. 10. Lifted Homomorphism $F(H_0)\colon F(\mathbb{A}_0) \mapsto F(\mathbb{A}_1)$

We now construct $\mathbb{B}$ by (a) taking images of each strand $\alpha_i(\rho_1^i)$ using $\alpha_i(\rho_2^i)$ up to height $g(k)$, where k is the height of $\alpha_i(\rho_1^i)$ in $\mathbb{A}$ and $F(\rho_1^i) = \rho_2^i, g$; (b) selecting $\preceq_{\mathbb{B}}$ to be a minimal extension of $\varphi(\preceq_{\mathbb{A}})$ compatible with the strand ordering in $\mathbb{B}$; and (c,d) letting $\mathsf{non}_{\mathbb{B}} = \mathsf{non}_{\mathbb{A}}$ and $\mathsf{unique}_{\mathbb{B}} = \mathsf{unique}_{\mathbb{A}}$.

$\mathbb{B}$ is a Π_2 skeleton: The conditions on the nodes and ordering are immediate from the definitions. Moreover, Definition 3, Clause 4 ensures that any node in $\mathbb{B}$ in which $a \in \mathsf{non}_{\mathbb{B}}$ appears as an ingredient is an image of a node in $\mathbb{A}$ in which it already appeared as an ingredient. Thus, the condition on non-origination is satisfied in $\mathbb{B}$ because it was satisfied in $\mathbb{A}$. Similarly, Definition 3, Clause 4 ensures that any node in $\mathbb{B}$ in which $a \in \mathsf{unique}_{\mathbb{B}}$ originates is an image of a node in $\mathbb{A}$ in which it already originated. Thus, the condition on unique origination is satisfied in $\mathbb{B}$ because it was satisfied in $\mathbb{A}$.

Moreover, by the choice of the substitutions α_i, $\mathbb{B}$ has a nodewise injective homomorphism to any other image of $\mathbb{A}$. □

As an example, in Fig. 10, we provide the images of the skeletons $\mathbb{A}_0, \mathbb{A}_1$ from Fig. 9. The new parameters N, N' have been chosen to be different from each other, and different from any other value appearing in the skeletons. Despite the fact that the other primed parameters disappear from $\mathbb{A}_1$, N' remains; generality would have been lost if it were forced to agree with N. Hence, $F(\mathbb{A}_1)$ is not realized, although the result of the substitution $N' \mapsto N$ would yield a realized skeleton.

5 Transformations and Homomorphisms

There are three main facts that show that our notion of transformation is reasonable, and these concern the way that transformations relate the homomorphisms among Π_1-skeletons with those among Π_2-skeletons. The first shows that transformations lift each Π_1-homomorphism to a Π_2-homomorphism which is unique

$$\begin{tikzcd} \mathbb{A}_0 \arrow[r, mapsto, "H"] \arrow[d, "\varphi_0"'] & \mathbb{A}_1 \arrow[d, "\varphi_1"] \\ F(\mathbb{A}_0) \arrow[r, mapsto, "G"] & F(\mathbb{A}_1) \end{tikzcd}$$

Fig. 11. Homomorphisms H, G

to within isomorphism. We write this G subsequently as $F(H)$. See Fig. 11. An example appears in Fig. 10.

Theorem 1. *Let F transform Π_1 to Π_2, and suppose $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$ is a homomorphism on Π_1-skeletons. There is a homomorphism G on Π_2-skeletons $G\colon F(\mathbb{A}_0) \mapsto F(\mathbb{A}_1)$ such that (letting φ_i satisfy the conditions of Def. 4 for $\mathbb{A}_i$):*

1. *$\varphi_1(H(n)) = G(\varphi_0(n))$ for every $n \in \mathsf{nodes}(\mathbb{A}_0)$; and*
2. *$G(v) = H(v)$ for every atom or indeterminate v appearing in $\mathbb{A}_0$.*

Moreover, if G and G' both satisfy this property, then they differ by an isomorphism I, i.e. $G' = I \circ G$. G is node-injective iff H is.

Proof sketch. Suppose that the homomorphism $H = [\psi_1, \beta]$. We may regard the strands contributing nodes to $\mathbb{A}_0$ as a family $\alpha_i(\rho_1^i)$ in such as way that the strands of $\mathbb{A}_1$ in the image of $\mathbb{A}_0$ under ψ are all of the form $(\beta \circ \alpha_i)(\rho_1^i)$. $\mathbb{A}_1$ may restrict which roles are chosen more tightly than $\mathbb{A}_0$, since they may have greater height in $\mathbb{A}_1$. Let β be general for parameters not appearing in $\mathbb{A}_0$.

Thus, we define $G = [\psi_2, \beta]$, where $\psi_2 = \phi_1 \circ \psi_1 \circ \phi_0^{-1}$. G is a homomorphism because each strand $\alpha_i(\rho_2^i)$ in $F(\mathbb{A}_0)$ is mapped by ψ_2 to $\beta \circ \alpha_i(\rho_2^i)$ as desired, and the origination constraints follow as in Lemma 1.

If $G' = [\psi_2', \beta']$ also satisfies the conditions, then let ϕ_1' a function such that, for $n \in \mathsf{nodes}_{\mathbb{A}_1}$, $\phi_1'(n)$ is (1) $\phi_1(n)$ if n is not in the range of ψ_1; and (2) $\psi_2'(\phi_0(m))$ for any m such that $\psi_1(m) = n$, otherwise. By the uniqueness condition on ϕ_1, $\phi_1' \circ (\phi_1)^{-1}$ is the desired isomorphism on $F(\mathbb{A}_1)$.

The node-injectiveness property is immediate from the definition of ψ_2 using the bijections ϕ_1, ϕ_0^{-1}. □

Retaining the notation of Fig. 11, we can also go in the other direction:

Theorem 2. *Let F transform Π_1 to Π_2, and suppose $G\colon F(\mathbb{A}_0) \mapsto F(\mathbb{A}_1)$. $G = F(H)$, for some $H\colon \mathbb{A}_0 \mapsto \mathbb{A}_1$. H is unique to within isomorphism.*

Proof sketch. Let $G = [\psi_2, \beta]$. We define $\psi_1 = \phi_1^{-1} \circ \psi_2 \circ \phi_0$. □

Finally, each Π_2-skeleton $\mathbb{B}$ has a decomposition into a maximal part of the form $F(\mathbb{A})$, to which a node-injective mapping is applied.

Theorem 3. *Let F transform Π_1 to Π_2, and let $\mathbb{B}$ be a Π_2-skeleton. Let*

$$S = \{F(\mathbb{A}_0)\colon \mathbb{A}_0 \textit{ is a } \Pi_1\textit{-skeleton and } F(\mathbb{A}_0) \leq_N \mathbb{B}\}.$$

Then S has a $\leq_N$-maximum member, i.e. there is a $\mathbb{A}_1$ such that $F(\mathbb{A}_1) \in S$ and for every $\mathbb{B}_0 \in S$, $\mathbb{B}_0 \leq_N F(\mathbb{A}_1)$.

Proof sketch. For each strand s with nodes in $\mathsf{nodes}_{\mathbb{B}}$, consider the nodes in $\mathbb{B}$ that equal the nodes of any $F(\alpha_i(\rho_1^i))$. For each s, choose a ρ_1^i that maximizes the k such that s is of the form $F(\alpha_i(\rho_1^i))$ up to $g(k)$. Let $\mathbb{B}_1$ be the subskeleton of $\mathbb{B}$ where:

1. $\mathsf{nodes}_{\mathbb{B}_1}$ consists of the nodes of $\mathbb{B}$ of height less than this $g(k)$ for each s;
2. $\preceq_{\mathbb{B}_1}$ is the weakest order generated from the strand orderings and $\preceq_{\mathbb{B}}$ restricted to nodes of the form $s \downarrow g(j)$;
3. $\mathsf{non}_{\mathbb{B}_1}$ is the subset of $\mathsf{non}_{\mathbb{B}}$ which are parameters appearing in some $\alpha_i(\rho_1^i)$, and likewise for $\mathsf{unique}_{\mathbb{B}_1}$.

This $\mathbb{B}_1$ is of the form $F(\mathbb{A}_1)$. □

6 Conclusion

In this paper we have simply provided a definition, and shown that the definition is well-behaved. Namely, we show that the definition of transformation fits the skeletons-and-homomorphisms framework of strand spaces. However, this in itself does not tell us when a transformation respects security properties.

For this, we believe that our work on protocol composition via the authentication tests provides the crucial hint [8]. We show there that if adding a new protocol Π_2 to an existing protocol Π_1 produces no new ways to solve the challenge-response patterns on which Π_1's security goals depend, then Π_2 preserves security goals that Π_1 achieves. We also provided a syntactic way to define and validate the “no new solutions” property. We believe that a rewriting of that property to our current framework provides a sufficient criterion for preserving security properties through protocol elaboration.

Such a result would complement the protocol transformation techniques developed by Datta, Derek, Mitchell, and Pavlovic in an outstanding series of papers including [3,4]. The authors explore a variety of protocols with common ingredients, showing how they form a sort of family tree, related by a number of operations on protocols.

Our definition of multiprotocol from [8] covers both [4]'s *parallel composition* and its *sequential composition*. *Refinement* enriches the message structure of a protocol. Their *transformation* moves information between protocol messages, either to reduce the number of messages or to provide a tighter binding among parameters. Our notion of transformation appears to cover both refinement and their transformation, although the “no new solutions” property appears more likely to work with refinements than with transformations in their sense.

Despite their rich palette of operations, their main results are restricted to parallel and sequential composition [4, Thms. 4.4, 4.8]. Each result applies to particular proofs of particular security goals G_1. Each proof relies on a set Γ of invariant formulas that Π_1 preserves. If a secondary protocol Π_2 respects Γ, then G_1 holds of the parallel composition $\Pi_1 \cup \Pi_2$ (Thm. 4.4). Thm 4.8, on sequential composition, is more elaborate but comparable. By contrast, the key theorem of [8] is one uniform assertion about all security goals, independent of their proofs. It ensures that Π_2 will respect all usable invariants of Π_1. This syntactic property, checked once, suffices permanently, without looking for invariants to re-establish.

References

1. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 2–17. Springer, Heidelberg (2007)
2. Cortier, V., Delaitre, J., Delaune, S.: Safely composing security protocols. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 352–363. Springer, Heidelberg (2007)
3. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Abstraction and refinement in protocol derivation. In: Proceedings of Computer Security Foundations Workshop. IEEE CS Press, Los Alamitos (2004)
4. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13(3), 423–482 (2005)
5. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Completeness of the authentication tests. In: Biskup, J., Lopez, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 106–121. Springer, Heidelberg (2007)
6. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007), http://eprint.iacr.org/2006/435
7. Fröschle, S.: Adding branching to the strand space model. In: Proceedings of EXPRESS 2008. Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam (2008) (to appear)
8. Guttman, J.D.: Cryptographic protocol composition via the authentication tests. In: de Alfaro, L. (ed.) Foundations of Software Science and Computation Structures (FOSSACS). LNCS, vol. 5504, pp. 303–317. Springer, Heidelberg (2009)
9. Guttman, J.D., Herzog, J.C., Ramsdell, J.D., Sniffen, B.T.: Programming cryptographic protocols. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 116–145. Springer, Heidelberg (2005)
10. Guttman, J.D., Thayer, F.J.: Protocol independence through disjoint encryption. In: Proceedings, 13th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos (2000)
11. Guttman, J.D., Thayer, F.J.: Authentication tests and the structure of bundles. Theoretical Computer Science 283(2), 333–380 (2002); Conference version appeared in IEEE Symposium on Security and Privacy (May 2000)
12. Hui, M.L., Lowe, G.: Fault-preserving simplifying transformations for security protocols. Journal of Computer Security 9(1/2), 3–46 (2001)
13. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: ESOP Proceedings. LNCS. Springer, Heidelberg (2009)

Formal Validation of OFEPSP+ with AVISPA*

Jorge L. Hernandez-Ardieta, Ana I. Gonzalez-Tablas, and Benjamin Ramos

Department of Computer Science, University Carlos III of Madrid
Avda. de la Universidad 30, 28911 Leganes, Spain
jlopez.ha@gmail.com, aigonzal@inf.uc3m.es, benja1@inf.uc3m.es

Abstract. Formal validation of security protocols is of utmost importance before they gain market or academic acceptance. In particular, the results obtained from the formal validation of the improved Optimistic Fair Exchange Protocol based on Signature Policies (OFEPSP+) are presented. OFEPSP+ ensures that no party gains an unfair advantage over the other during the protocol execution, while substantially reducing the probability of a successful attack on the protocol due to a compromise of the signature creation environment. We have used the Automated Validation of Internet Security Protocols and Applications (AVISPA) and the Security Protocol ANimator for AVISPA (SPAN), two powerful automated reasoning technique tools to formally specify and validate security protocols for the Internet.

Keywords: Fair exchange, security protocol, formal validation, AVISPA, SPAN.

1 Introduction

Formal validation of security protocols is of utmost importance before they gain market or academic acceptance. Some standard and widely used security protocols for the Internet have been proved to suffer from critical design flaws that an attacker can exploit to subvert their security. The reason is that their security goals were merely informally evaluated, obviating potential attack paths. Automated reasoning techniques are commonly used to evaluate the protocols in a formal way, increasing the assurance respecting the purported security. In this sense, the Automated Validation of Internet Security Protocols and Applications (AVISPA) [2] and the Security Protocol ANimator for AVISPA (SPAN) [12] tools have been used to validate the correctness and safety of the improved Optimistic Fair Exchange Protocol based on Signature Policies (OFEPSP+).

OFEPSP [13], the protocol from which OFEPSP+ has been designed, is a protocol oriented to Internet transactions where two parties need to exchange information in a fair and secure manner. The origin sends a signed message to

* The authors would like to thank the AVISPA project team, and specially Laurent Vigneron and Luca Viganò, for their useful comments on the preliminary versions of OFEPSP+ HLPSL specification. The authors wish also to thank the reviewers for their valuable remarks.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 124–137, 2009.

the receiver while the receiver sends back a proof of receipt of the message. Therefore, both parties are making a commitment in the transaction: the origin cannot repudiate having sent the message and the receiver cannot repudiate its reception. As a fair exchange protocol, OFEPSP ensures that no party gains an unfair advantage over the other during the protocol execution. Therefore, either both parties obtain the expected information or none of them obtains any useful information from the other. As an optimistic protocol, a Trusted Third Party (TTP) is included in the design but participating only when a party's misbehavior or protocol error occurs.

Many fair exchange protocols found in literature are designed using symmetric encryption, assuring the undisclosure of the message sent by the origin until the receiver has made a commitment in the transaction [16]. In our case, OFEPSP is based on signature policies [10]. A signature policy is a document that collects a set of rules to create and validate electronic signatures, and under which an electronic signature can be determined to be valid in a particular transaction context. Signature policies are defined both in human readable form and structured form using an agreed syntax and encoding, like ASN.1 or XML. The signature policy reference is included as a signed property in each signature performed by the parties, allowing any verifier to ascertain if the signature matches the requirements imposed in the policy. Therefore, signature creation and verification processes can be completely carried out in an automatic and transparent way in accordance with the signature policy rules.

In order to tie down the origin and receiver respecting the exchanged information, most protocols use digital signatures. Due to the cryptographic basis of digital signatures, a correctly verified signature is known to have been computed with a particular private key. The common (and mistaken) established assumption is that the entity which owns the private key cannot repudiate having performed such a signature as nobody else could have computed it. If applied to fair exchange protocols, the signed information sent by the origin and the proof of receipt cannot be repudiated by the corresponding parties. However, several feasible and practical attacks on the signature creation environment are found in the literature [1,11,14,15,17]. Therefore, an attacker gaining access to the origin's private key or just the signature creation environment could impersonate her in a transaction, no matter how the underlying protocol is designed.

To reduce this risk, we improved OFEPSP, dividing the origin's environment in a Signature Creation Environment (SCE) and a Transaction Confirmation Environment (TCE). This division substantially reduces the probability of a successful attack on the protocol as both environments are needed to create the valid evidence. The security of both the SCE and the TCE can be compromised. However, the probability of a successful distributed attack on SCE and TCE by two different malwares, and which collaborate in order to undermine the protocol security, is, in any case, substantially lower that the probability of compromising the security of a single environment. OFEPSP+ assures that an attacker will not gain any benefit from a potential security compromise on either the SCE or the TCE, enforcing the non-repudiation property of the evidence generated during a protocol run. To the best of our knowledge, this is the first time a fair exchange protocol takes the security of the parties' environments into consideration for the overall protocol design. OFEPSP+ can be applied to a variety of scenarios like contract signing or e-commerce transactions, where the origin corresponds to an end user using a potentially compromised SCE, like her Personal Computer.

In this article, OFEPSP+ is briefly presented and the formal validation of the protocol detailed. The preliminary results are promising, but some work must still be done to obtain a high assurance of the protocol security. In this first approach, the formal validation covered masquerading and integrity attacks. Fairness could not be modeled yet taking into account standard AVISPA features. Nonetheless, future work will cover the use of additional predicates and special goal formulas in order to incorporate fairness in the validation.

The article is organized as follows. The next Sect. 2 describes OFEPSP+ protocol. Section 3 covers the formal analysis of the protocol by means of AVISPA and SPAN tools. And finally, we conclude the article in Sect. 4.

2 The Improved Optimistic Fair Exchange Protocol Based on Signature Policies

This Sect. describes the improved Optimistic Fair Exchange Protocol based on Signature Policies (OFEPSP+). The basic notation and definitions are given first. Afterwards, OFEPSP+ set of protocols is explained in "Alice & Bob" notation.

2.1 Basic Notation and Definitions

Following notation is necessary for the correct understanding of further Sects.:

SP	Signature policy used during the protocol
$X \rightarrow Y : m$	Party X sends message m to party Y
$X \leftarrow Z : SP$	Party X retrieves SP from a repository located at party Z
$S_x\left(m\right)$	Signature of party x generated on message m
$S_x\left(m \mid SP\right)$	Signature of party x generated on message m under SP conditions

$POO = S_O\left(m, \ell, \mathit{tpl\_id} \mid SP\right)$

Proof of origin of message m.

$POR = S_R\left(m, \ell, \mathit{tpl\_id} \mid SP\right)$

Proof of receipt of message m.

$NRO = S_O\left(POR \mid SP\right)$

Non-repudiation evidence of origin of message m generated on POR.

$NRR = S_R\left(NRO \mid SP\right)$

Non-repudiation evidence of receipt of message m generated on NRO.

$NRA_O = S_O\left(NRR \mid SP\right)$

Non-repudiation evidence of acknowledgment generated by origin on NRR. This is the valid evidence that completes the protocol.

$NRA_{TTP} = S_{TTP}\,(NRR|SP)$

Non-repudiation evidence of acknowledgment generated by the TTP on NRR. This is the valid evidence when the recovery protocol has been executed.

Each protocol run is identified by a protocol identifier ℓ. A template is used by the parties in order to fix the information to be sent by the origin. This template is referenced by the template identifier *tpl_id*. The template must be defined by the receiver according to the transaction needs. The message m sent by the origin must be further processed by the receiver taking into account the template information.

2.2 Main Protocol

OFEPSP+ consists of one main protocol, explained in this Subsect., and two subprotocols, called recovery subprotocol and abort subprotocol, which are further explained. During the exchange process, the origin must use two different platforms, called the Signature Creation Environment (SCE) and the Transaction Confirmation Environment (TCE). SCE is the platform where the signature application is installed and used by the origin to create electronic signatures on her behalf. The origin must use a hardware cryptographic device with signing capabilities (e.g. smart card) for that purpose. This cryptographic device is regarded as part of the SCE platform. On the other hand, the origin acknowledges the performed signatures by means of a second cryptographic device (and thus different signing key) used in a different platform with another signature application (e.g. mobile device with cryptographic capabilities), that is, the TCE.

In the main protocol, the origin initiates the transaction by sending the signed message to the receiver by means of SCE. Afterwards, the origin confirms the initiated transaction by using TCE. The receiver will exchange several intermediate evidences with both the SCE and TCE, until the final valid evidence NRA is generated. Next, the main protocol is formalized using the notation of SubSect. 2.1 above:

(1) $O_{SCE} \leftarrow R : tpl\,[tpl\_id]\,, S_R\,(tpl\,[tpl\_id])$

First, the origin (O) requests the template identified by *tpl_id* to the receiver (R) (1) by means of the SCE platform.

(2) $O_{SCE} \leftarrow$ *TTP-SP*: $SP, S_{TTP-SP}\,(SP)$

In next step (2) the origin retrieves the signature policy SP necessary to communicate with the receiver. Once the origin has obtained the template and the signature policy, she can produce the message and POO taking into account this information.

(3) $O_{SCE} \rightarrow R : m, \ell, tpl\_id, POO$

Afterwards (3), the origin starts the protocol itself by sending the message m, a unique protocol identifier ℓ, the template reference and the POO.

(4) $R \leftarrow \textit{TTP-SP}\text{: } SP, S_{TTP-SP}\,(SP)$

(5) $R \rightarrow O_{TCE} : m, \ell, tpl\_id, POO, POR$

The receiver retrieves the signature policy (4), if not obtained yet, and validates the received POO. Afterwards, the receiver generates and sends the POR to the origin's TCE (5). The receiver must also send all the information received from the origin in step (3) in order to allow her to validate the initiated transaction using the second platform TCE. Step (4) can be avoided for efficiency purposes if the receiver accesses the TTP-SP once and afterwards manages a local copy of the signature policy, provided that it is within its validity period.

(6) $O_{TCE} \rightarrow R : NRO$

The origin must generate the NRO only if the information received in step (5) corresponds to a desired transaction and POO and POR are correctly verified.

(7) $R \rightarrow O_{SCE} : NRR$

Once the origin has confirmed the transaction by means of the NRO, the receiver sends the NRR to the origin's SCE platform (7).

(8) $O_{SCE} \rightarrow R : NRA_O$

In the last step (8) the origin completes the transaction by sending the NRA to the receiver.

Although not shown above, evidences POO, POR, NRO, NRR and NRA must be time-stamped. The time-stamping procedure must be carried out according to known standards, and implies the participation of a Time-Stamping Authority (TSA). The time-stamp is an assertion of proof given by the TSA that the datum existed before the specified time.

2.3 Recovery Subprotocol

The recovery subprotocol allows the receiver to obtain evidence NRA in case of a protocol interruption or origin's misbehavior, and must be executed if the receiver does not receive the NRA within a specific time interval. OFEPSP+ recovery subprotocol consists of next steps:

```
(1)   $R \rightarrow TTP : H\,(m, \ell, tpl\_id)\,, \ell, POO, POR, NRO, NRR$
      if (protocol aborted) then
(2a)    $TTP \rightarrow R : S_{TTP}\,(S_O\,(abort, \ell|SP)\,|SP)$
      else
(2b)    $TTP \leftarrow \textit{TTP-SP}\text{: } SP$
(3b)    $TTP \rightarrow R, O_{SCE}, O_{TCE} : NRA_{TTP}$
```

In (1) the receiver sends the produced evidences POO, POR, NRO and NRR to the TTP. Also, and in order to protect the privacy of the parties, the information signed in POO and POR - the message, the protocol identifier and the template reference - is not sent in plain text but the hash of their concatenated values. Yet the TTP is still able to verify POO and POR by directly using the hash, provided that a digital signature scheme based on public key cryptography is used (i.e. RSA, DSA, ECDSA). The TTP must decrypt POO and POR - using the corresponding public keys -, obtaining the hash of the signed information, which must correspond to the value of $H\left(m, \ell, tpl\_id\right)$. ℓ is sent in (1) to allow the TTP to retrieve and update the information associated to the transaction.

If the protocol has already been aborted, the TTP merely forwards the abort evidence to the receiver (2a). On the other hand, the TTP generates the NRA_{TTP} taking into account the referenced signature policy - (2b) and (3b) -, but only in the first request. The evidence must be stored in a local database along with the received information. Subsequently, the TTP will reuse it, improving the efficiency.

It is important to remark that the signatures that correspond to the evidences generated in the protocol (POO, POR, NRO, NRR and NRA) must be generated according to electronic signature standards (please refer to [13] for further information). Thereby, a reference to the signature policy used in the protocol is included as a signed property in the specific electronic signature format chosen for the transaction (i.e. XAdES, CAdES). This permits the TTP to know and retrieve the signature policy and avoids an attacker to modify the referenced signature policy.

2.4 Abort Subprotocol

The abort subprotocol allows the origin to abort the protocol execution if a receiver's malicious behavior is suspected or an error during the protocol run has occurred. OFEPSP+ abort subprotocol consists of next steps:

(1) $O_{SCE|TCE} \rightarrow TTP : abort, \ell, S_O\left(abort, \ell | SP\right)$
if (recovery protocol executed) then
(2a) $TTP \rightarrow O_{SCE|TCE} : NRA_{TTP}$
else
(2b) $TTP \leftarrow$ TTP-SP: SP
(3b) $TTP \rightarrow O_{SCE|TCE} : S_{TTP}\left(S_O\left(abort, \ell | SP\right) | SP\right)$

For efficiency purposes, $S_{TTP}\left(S_O\left(abort, \ell | SP\right) | SP\right)$ is only generated in the first time (2b) and (3b), being reused in subsequent executions of the abort subprotocol. On the other hand, if the protocol has been recovered (2a), the TTP just retrieves the NRA_{TTP} from its data base, forwarding it to the origin.

3 OFEPSP+ Validation with AVISPA and SPAN

This Sect. covers the validation process using the Automated Validation of Internet Security Protocols and Applications tool (AVISPA) and the Security Protocol ANimator for AVISPA (SPAN).

AVISPA [2,3] provides a suite of applications for building and analyzing formal models of security protocols. AVISPA incorporates four backends: the On-the-Fly Model-Checker (OFMC) [6], the Constraint-Logic-based model-checker

(CL-AtSe) [18], the SAT-based Model-Checker (SATMC) [4], and the Tree Automata based Automatic Approximations for the Analysis of Security Protocols (TA4SP) [7]. These modules implement different automated reasoning techniques to formally analyze the protocol specification. On the other hand, SPAN [12] offers a graphical user interface that allows the protocol designer to easily interact with AVISPA capabilities.

Protocol models must be written in the High Level Protocol Specification Language (HLPSL) [5,8], which allows the protocol designer to describe the security protocol and specify its intended security properties. HLPSL details have been omitted for lack of space.

AVISPA adopts the standard intruder model of Dolev and Yao (DY model) [9], in which the intruder has complete control over the network but cannot break cryptography. The intruder may intercept, analyze, and/or modify messages (as far as he knows the required keys), and send any message he composes to whoever he pleases, posing as any other agent. The goal of OFEPSP+ validation was to check the correctness and safety of the protocol respecting DY model.

The validation methodology followed can be summarized in next 3 steps:

1. OFEPSP+ specification in HLPSL.
2. HLPSL correctness verification.
3. OFEPSP+ security validation.

3.1 OFEPSP+ Specification in HLPSL

OFEPSP+ complexity lies in the existence of four entities. The first two entities, SCE and TCE, are managed by the origin of the protocol, but, in HLPSL, had to be considered as two different roles played by a different agent each. The other two entities, Receiver and TTP, were modeled as the receiver and the TTP roles, respectively, played by the corresponding agents. Each role implemented its own state transition system according to the steps indicated in the protocol described in Sect. 2. Thanks to the *set* structure available in HLPSL, the TTP's evidence database could be modeled and the TTP behavior accurately defined.

Restrictions Applied

The protocol steps described in Sect. 2 could be modeled, except next four issues.

Signature Policy-based Design

OFEPSP+ fairness property is enforced by the signature policy-based design, assuring that an incomplete evidence (POO, POR, NRO or NRR) does not tie down any of the parties involved. The existence of NRA is imperative to prove the commitment made by both parties, and its creation (either by the origin or the TTP) must fulfill the policy constraints and requirements. We found that the usage of signature policies could not be modeled in HLPSL. HLPSL allows translating an Alice & Bob chart into a more detailed and expressive language. However, not every protocol behavior can be mapped in HLPSL. Therefore, the protocol steps related to the policy retrieval were discarded during the HLPSL definition.

Time-Stamping
Time-stamps are needed in the protocol to allow the parties to know when the recovery or abort subprotocols must be executed. The trigger is a timeout defined in the signature policy for each case. However, HLPSL only allows the establishment of a generic timeout as an incoming message to a role. This feature avoided us to model the specific timeouts, and thus, the time-stamping processes were obviated. We have checked that this constraint has not modified neither the protocol behavior nor its security goals fulfillment.

Template Usage
In OFEPSP+ the origin must create the message m according to the template constraints. This measure restricts the semantics of the signed information, avoiding most of semantic attacks currently developed [1,14,15]. However, and as previously mentioned, in this first approach we wanted to evaluate the security measures against DY model. For that reason, the template retrieval by the origin in the first step of the main protocol was discarded as well.

Server Role Capabilities
Finally, the TTP (server) role was initially too complex for the backend analyzers of AVISPA, due to the number of transition combinations considered during the validation process. Note that our TTP is designed as an e-notary, storing and managing the evidences generated during a protocol run in which the TTP takes part. Besides, there were three situations where the TTP could intervene: an abort requested by both the SCE and the TCE, and a recovery requested by the receiver. This led to a huge role definition with 9 transitions. Taking into account that CL-AtSe considers, by default, that each transition can be applied at most 3 times, the backend had to manage 27 possible transitions during the analysis. It seemed to be too complex.

For that reason, we simplified the role server, excluding the TCE from making abort requests. The origin is still able to abort the protocol by using the SCE. Thus, the server role was modeled considering next three states: 0 as the initial state; 1 as the state when an abort has been done first; and 2 as the state when a recovery has been done first. In each state, the server can receive both abort and recovery requests, leading to 6 defined transitions.

As a result of previous modifications applied, the server role cannot respond to several parallel sessions. The transitions are sequential, that is, once a transaction has been aborted or recovered, the server will stay in that state (1 or 2, respectively) for the rest of requests, no matter if they come from another session. Nonetheless, and as will be seen further, we were able to test the protocol with parallel sessions between SCE, TCE and Receiver, looking for possible attacks though the server behaved in this way.

We know that we have limited the possible space of attacks, but the decision was made in order to allow the backends to correctly analyze the protocol.

Analysis Scenarios
HLPSL must cover the definition of two special roles: session and environment. Respecting the session role, we just instantiated the roles sce, tce and receiver

with the adequate information. Mention that the agents SCE and TCE, since both are managed by the same origin, own a pre-shared knowledge: the message to be sent. The template reference is also a pre-shared knowledge between the SCE, TCE and the Receiver agents.

One of the most important parts of the HLPSL is the initial knowledge allocated to the intruder. In this sense, and for test purposes, we defined five different template references (tpl_id, tpl_id2, tpl_id3, tpl_id4) and four different messages (msg, msg2, msg3, msg4), assigning the subset {msg2, msg3, tpl_id2, tpl_id3, tpl_id4} as knowledge to the intruder.

Afterwards, we defined several analysis scenarios with different sessions configurations (see section 3.3 for details). Due to the constraints applied to the server role capabilities commented above, we instantiated just one server role in each scenario.

Security Goals

AVISPA supports three types of goals so far: *secrecy_of*, (strong) *authentication_on* and *weak_authentication_on*. In the latter, the piece of data used to authenticate the agent can be reused by an attacker, so reply attacks are not considered by the analyzers. As AVISPA does not explicitly support fairness and non-repudiation security goals, some fair exchange/non-repudiation protocols modeled with AVISPA used *secrecy_of* goal to achieve it. However, OFEPSP+ does not provide confidentiality on any item. Fairness is achieved by means of the signature policy fulfillment, as explained above. Therefore, currently we have only modeled authentication goal respecting the exchanged evidences.

3.2 HLPSL Correctness Verification

In this step, the aim was twofold: verify the syntactic/semantic correctness and executability of the HLPSL specification; and that the HLPSL specification implemented the intended protocol behavior.

For the syntactic, semantic and executability verification, next AVISPA and SPAN tools were used:

HLPSL2IF: This tool translates the HLPSL specification into the Intermediate Format (IF). A successful translation implies that the syntax of the protocol is correct. OFEPSP+ HLPSL file was correctly translated into the corresponding IF.

Protocol simulation: By simulating the protocol with SPAN, the semantic of the protocol's HLPSL is verified and a Message Sequence Chart (MSC) visualized. In our case, the semantic was correctly verified but the MSC could not be shown. It frequently happens when the transition labels and state values are not perfectly set. In any case, it does not imply an error in the specification.

OFMC search tree option: OFMC offers the possibility to browse the search tree through a path indicated by the indexes of the successors to follow. As a result, one can decide which choice point take in a specific point of the search tree deduced from the IF. This option allows the tester to check if every transition can be taken during a protocol run. In our case, every transition could be chosen at some time in the search tree.

CL-AtSe no executability option: CL-AtSe offers the possibility of tracing the protocol specification without being analyzed. The output shows the so called *Initial System State*, representing both the intruder and honest participants states in CL-AtSe just after reading and interpreting the IF file. While the intruder state is just represented by a list of knowledges, the honest participants are described by a set of instantiated roles, called *Interpreted protocol specification.* This option is useful to check that CL-AtSe interprets the protocol transitions as expected. Each role consists in a tree where unary nodes are protocol steps and n-ary nodes are choice points. In our case, each possible transition was represented in the tree.

SATMC check only executability: With this option, SATMC checks on executability of actions/rules without any intruder, allowing the tester to debug the specification. The output trace showed that every rule could be executed.

Session compilation with OFMC: With session compilation (sessco), OFMC finds a replay attack even without a second parallel session. It first simulates a run of the whole system and in a second run, it lets the intruder take advantage of the knowledge learned in the first run. Sessco is also handy for a quick check of executability. However, as stated in AVISPA documentation, if one role can loop (i.e. remain in the same control state forever and make infinitely many steps), sessco is not possible, and OFMC aborts with an error message. That is our case in some transitions of role server, and thus, we could not use this option.

CL-AtSe no executability option and *OFMC search tree option* helped us also to ascertain that the HLPSL specification matched the intended protocol behavior.

3.3 OFEPSP+ Security Validation

The results obtained from validating OFEPSP+ with OFMC and CL-AtSe backends are shown in next Tables 1, 2 and 3. In case of SATMC, the result was always "Inconclusive". Tests reports showed us that SATMC did not find an attack, but it warned that, with SATMC backend, intruder is not allowed to generate fresh terms (as in sce role). As a consequence, attacks based on such an ability would not be reported. TA4SP was not used because it does not support *sets* up to now. The analysis scenarios referred in Tables below are described in Table 4[1].

Configurations applied in Table 1 were aimed at finding attacks in a normal session (cfg1) or sessions when the intruder impersonates one of the legitimate agents - SCE (cfg2), TCE (cfg3) or Receiver (cfg4).

Configurations shown in Table 2[2] were focused on violating the security goals when two coherent parallel sessions are executed (cfg5) and when a legitimate

[1] Intruder is denoted as 'i'. We did not instantiated any session with the intruder playing the role of the server because we consider the TTP to be honest.

[2] OFMC executed with maximum search depth of 17. CL-AtSe executed with -nb 1 option (maximum 1 loop iteration in any trace) for test marked with (*).

Table 1. Validation results with OFMC and CL-AtSe respecting a single session with legitimate agents and single sessions with intruder playing the role of a legitimate agent

Analysis scenario	OFMC	CL-AtSe
cfg1	SAFE	SAFE
cfg2	SAFE	SAFE
cfg3	SAFE	SAFE
cfg4	SAFE	SAFE

Table 2. Validation results with OFMC and CL-AtSe respecting parallel sessions with legitimate agents playing different roles

Analysis scenario	OFMC	CL-AtSe
cfg5	SAFE	SAFE(*)
cfg6	SAFE	SAFE
cfg7	SAFE	SAFE
cfg8	SAFE	SAFE

Table 3. Validation results with OFMC and CL-AtSe respecting parallel sessions with intruder playing as legitimate agent(s)

Analysis scenario	OFMC	CL-AtSe
cfg9	SAFE(*)	SAFE
cfg10	SAFE(*)	SAFE
cfg11	SAFE(*)	SAFE
cfg12	SAFE	SAFE
cfg13	SAFE	SAFE
cfg14	SAFE	SAFE

party is playing a role for which is not intended to in case of parallel sessions (cfg6, cfg7 and cfg8). We realized that each participant's identifier had to be included in each evidence generated in order to avoid this sort of attack. In particular, we used the public keys of SCE, TCE, Receiver, and TTP.

Table 3[3] contains the set of configurations where an intruder is playing the role of legitimate agent(s) when two parallel sessions are executed. Note that the knowledge own by the intruder in each configuration differs. The aim was to find possible security goals violations in a session when carried out from an intruder running in another different session. Mention that when the intruder plays a legitimate role in a session, the goals involving him are not considered by AVISPA (otherwise, he could always achieve an attack).

Based on the results obtained from the tests, our protocol fulfills the security goals G2 - Message authentication (includes message integrity), G17 - Accountability, G18 - Proof of Origin and G19 - Proof of Delivery for evidences POO,

[3] Results marked with (*) mean that OFMC was launched with maximum search depth of 17.

Table 4. Analysis scenario configurations for the tests

Analysis scenario	sessions configuration
cfg1	$session\,(sce, tce, r, s, ..., msg, tpl\_id)$
cfg2	$session\,(i, tce, r, s, ..., msg, tpl\_id)$
cfg3	$session\,(sce, i, r, s, ..., msg, tpl\_id)$
cfg4	$session\,(sce, tce, i, s, ..., msg, tpl\_id)$
cfg5	$session\,(sce, tce, r, s, ..., msg, tpl\_id)$ $session\,(sce, tce, r, s, ..., msg, tpl\_id)$
cfg6	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(r, tce, sce, s, \dots, msg, tpl\_id)$
cfg7	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(sce, r, tce, s, \dots, msg, tpl\_id)$
cfg8	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(tce, sce, r, s, \dots, msg, tpl\_id)$
cfg9	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(i, tce, r, s, \dots, msg, tpl\_id)$
cfg10	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(sce, i, r, s, \dots, msg3, tpl\_id3)$
cfg11	$session\,(sce, tce, r, s, \dots, msg, tpl\_id)$ $session\,(sce, tce, i, s, \dots, msg4, tpl\_id4)$
cfg12	$session\,(i, tce, r, s, \dots, msg2, tpl\_id2)$ $session\,(sce, i, r, s, \dots, msg3, tpl\_id3)$
cfg13	$session\,(i, tce, r, s, \dots, msg2, tpl\_id2)$ $session\,(sce, tce, i, s, \dots, msg3, tpl\_id3)$
cfg14	$session\,(sce, i, r, s, \dots, msg, tpl\_id)$ $session\,(sce, tce, i, s, \dots, msg, tpl\_id)$

POR, NRO, NRR and NRA. Goals description are given in Deliverable 6.1 "List of selected problems" in AVISPA project [2].

Due to our protocol design, evidences are not protected against reply attacks (G3), and thus goal Entity Authentication (G1) is not achieved either. For that reason, only *weak_authentication_on* goal was assigned. We think that including a nonce in the requests generated by the receiver would enforce goals G1 and G3 for NRO, NRR and NRA. However, tests conducted including that nonce did not reach a conclusion, maybe due to the huge number of transitions the backends had to analyze. As a result, our assumption could not be proved.

4 Conclusions and Work in Progress

In this article an improved Optimistic Fair Exchange Protocol based on Signature Policies (OFEPSP+), which evolves a previously presented protocol [13], has been briefly presented and analyzed.

The protocol has been evaluated with the Automated Validation of Internet Security Protocols and Applications (AVISPA) and the Security Protocol ANimator for AVISPA (SPAN) against the intruder model of Dolev-Yao. The methodology followed to specify the protocol in the High Level Protocol Specification Language (HLPSL) and to validate it by means of AVISPA backends has been presented along with the results obtained. We found several problems during the specification stage, like modeling the intended protocol behavior in HLPSL, or checking that the protocol features that could not be defined in HLPSL did not affect the security properties evaluated (e.g. the signature policy-based design).

The preliminary results are promising, but some work must still be done. We are currently working on modifying the transition labels and state values in order to get a protocol simulation, what will allow us to interact in a protocol run as an active intruder. Afterwards, we will enrich the test scenarios to increase the level of assurance respecting the security goals fulfillment. Finally, and as the biggest challenge, fairness will be specified by means of special predicates and goal formulas. We hope that these results and current work will allow us to implement a proved secure fair exchange protocol.

Acknowledgments. This research has been partially supported by the Ministry of Industry, Tourism and Trade of Spain, in the framework of the project CENIT-Segur@, reference CENIT-2007 2004. (https://www.cenitsegura.es)

References

1. Alsaid, A., Mitchel, C.J.: Dynamic content attacks on digital signatures. Information Management & Computer Security 13(4), 328–336 (2005)
2. AVISPA: Automated validation of internet security protocols and applications. FET Open Project IST-2001-39252 (2003), `http://www.avispa-project.org/`
3. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Hankes Drielsma, P., Heam, P.-C., Kouchnarenko, O., Mantovani, J., Modersheim, S., von Oheimb, D., Rusinowitch, M., Santos Santiago, J., Turuani, M., Vigano, L., Vigneron, L.: The AVISPA Tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
4. Armando, A., Compagna, L.: SATMC: A SAT-based model checker for security protocols. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 730–733. Springer, Heidelberg (2004)
5. AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language (2003), `http://www.avispa-project.org/publications.html`
6. Basin, D.A., Sebastian, M., Vigano, L.: Ofmc: A symbolic model checker for security protocols. Int. J. Inf. Sec. 4(3), 181–208 (2005)
7. Boichut, Y., Heam, P.-C., Kouchnarenko, O., Oehl, F.: Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In: Proc. of Automated Verification of Infinite States Systems (AVIS 2004). ENTCS, pp. 1–11 (2004)
8. Chevalier, Y., Compagna, L., Cuellar, J., Hankes Drielsma, P., Mantovani, J., Modersheim, S., Vigneron, L.: A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In: Proc. SAPS 2004. Austrian Computer Society (2004)

9. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 2(29) (1983)
10. ETSI TR 102 041 v1.1.1. Signatures policies report. European Telecommunications Standards Institute (2002)
11. Girard, P., Giraud, J.-L.: Software attacks on smart cards. Information Security Technical Report 8(1), 55–66 (2003)
12. Glouche, Y., Genet, T., Heen, O., Courtay, O.: A Security Protocol Animator Tool for AVISPA. In: ARTIST2 Workshop on Security Specification and Verification of Embedded Systems, Pisa (2006)
13. Hernandez-Ardieta, J.L., Gonzalez-Tablas, A.I., Alvarez, B.R.: An Optimistic Fair Exchange Protocol based on Signature Policies. Computers & Security 27(7-8), 309–322 (2008)
14. Jsang, A., Povey, D., Ho., A.: What You See is Not Always What You Sign. In: The Proceedings of the Australian UNIX User Group, Melbourne (2002)
15. Kain, K.: Electronic Documents and Digital Signatures. Master Thesis (2003)
16. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Computer Communications 25, 1601–1621 (2002)
17. Spalka, A., Cremers, A.B., Langweg, H.: Trojan Horse Attacks on Software for Electronic Signatures. Informatica 26(2), 191–203 (2002)
18. Turuani, M.: The CL-Atse Protocol Analyser. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)

On the Automated Correction of Protocols with Improper Message Encoding

Dieter Hutter[1] and Raúl Monroy[2]

[1] DFKI, Cartesium 2.41, Enrique Schmidt Str. 5, D-28359 Bremen, Germany
hutter@dfki.de
[2] Tecnológico de Monterrey, Campus Estado de México
Carr. Lago de Guadalupe Km. 3.5, Atizapán, Estado de México, 52926, México
raulm@itesm.mx

Abstract. Security protocols are crucial to achieve trusted computing. However, designing security protocols is not easy and so security protocols are typically faulty and have to be repaired. Continuing previous work we present first steps to automate this repairing process, especially for protocols that are susceptible to type-flaw attacks. To this end, we extend the notion of strand spaces by introducing an implementation layer for messages and extending the capabilities of a penetrator to swap messages that share the same implementation. Based on this framework we are able to track type flaw attacks to incompatibilities between the way messages are implemented and the design of concrete security protocols. Heuristics are given to either change the implementation or the protocol to avoid these situations.

1 Introduction

A security protocol is a protocol that aims to establish one or more security goals, often a combination of integrity, authentication or confidentiality. Security protocols are critical applications, since they are crucial to achieve trusted computing. So they are thoroughly studied to guarantee that there does not exist an interleaving of protocol runs violating a security goal, called an *attack*.

Designing correct security protocols, however, has proven to be problematic, since it is difficult to anticipate what an adversary can learn from observing the simultaneous execution of an arbitrary number of protocols, given some initial knowledge. Under the formal approach to security protocol verification, the adversary usually is Dolev-Yao: he is able to delay messages or prevent them reach their destination; he can analyse messages without breaking cryptography; he can build messages using his initial knowledge or reusing components from messages previously intercepted; he can send messages as if they were of his own or somebody else's.

Protocol flaws can usually be understood as violations to well-known design guidelines, such as those given in [1,2,17]. These kinds of principles capture prudent practises in security protocol design, while pinpointing features that make protocols difficult to verify or possibly susceptible to an attack, and should

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 138–154, 2009.

thus be avoided. Principles in [1] spot flaws caused by overlooking information security flow, while those in [2,17] spot flaws, commonly referred to as *protocol failures*, caused by misusing cryptosystems. In this paper, attention is restricted only to the design principles given in [1].

Guidelines for protocol design should be treated carefully, though. This is partly because they capture prudent practises of the 1990's, when aspects like resilience or denial-of-service attacks were not considered yet. But this is also partly because some guidelines are rather cunbersome. For example, some principles are unnecessarily normative, suggesting one should add full information to every protocol message (protocol ID, session ID, etc.) or suggesting one should use different encryption systems or add hashings to each message, e.g. [3,6]. Protocol designers are often unwilling to follow these guidelines. They aim at achieving security guarantees by applying a combination of cryptographic primitives.

A *type flaw attack* is a kind of replay attack[1] where a principal accepts a message of one type as a message of another. Heather *et al.* [11] have shown that it is possible, under certain circustamces and protocol assumptions, to prevent type flaw attacks by *tagging* every protocol message, and elements thereof, with a string indicating its intended type. However, as noticed by Malladi and Alves-Foss [13], message tagging makes it easier to elaborate a password guessing attack. This is because each tag provides the adversary a means for identifying a hit, since the tag, which is a meaningful string, might become readable after a code cracking attempt.

In this paper, we shall prove that blind message tagging is unnecessary. We shall argue that to prevent a type flaw attack, it is enough to adopt good practises of message encoding, as proposed by [1]. In particular, upon reception of a protocol message, an agent should be able to verify that the message is associated to a particular protocol step and to a particular run (see design principles 6—8 and 10). We shall see that our approach can provide a protocol message with as much tagging as necessary but also as less tagging as possible without getting faulty.

Continuing previous work [12], we propose a method that aims at automatically fixing security protocols that are susceptible to a replay attack. We rely on existing state-of-the-art tools, such as OFMC [4,5] or Cl-Atse [7,8], capable of finding a type flaw attack to an input faulty protocol. Our method then analyses the protocol and the attack to identify the faulty steps of the protocol and synthesises appropriate changes to fix them. This yields an improved version of the protocol that should be analysed and potentially patched again until no further flaws can be detected.

To capture type flaw attacks, we extend the notion of strand spaces [19] (see Section 2). Crucial to our approach is the distinction of a protocol specification at an abstract level and the implementation of messages at a concrete level. A type flaw attack is feasible only if the penetrator is capable of supplying a

[1] A *replay attack* is a form of attack where a data transmission is repeated or delayed.

honest principal a camouflaged message whose implementation equals that of the message being spoofed.

We discuss on the soundness of the extended version of strand spaces (see Section 3) and provide a method for automatic protocol repair (see Section 4).

2 Strands for Type Flaw Attacks

2.1 Message Terms

Abstract messages, ranged over by $M_1, M_2, \ldots$, are also called terms. The set of terms, A, is freely generated from two disjoint sets, the set of texts (T) and the set of keys (K), by means of concatenation, $M_1; M_2$, and encryption, $\{\!|M|\!\}_K$ ($K \in \mathsf{K}$). T contains Nonce, the set of nonces, $N_a, N_b, \ldots$; $\mathsf{Timestamps}$, the set of timestamps, $T_a, T_b, \ldots$; Agent, the set of principals, $A, B, \ldots$; and Tags, the set of tags, $\ell_a, \ell_b, \ldots$. There are two functions, one maps principals, $A, B, \ldots$, to their public keys, $K_a^+, K_b^+, \ldots$, and the other a pair of principals, $\langle A, B\rangle$, to their symmetric shared key, K_{ab}. K comes with an inverse operator mapping each member of a key pair for an asymmetric cryptosystem to the other, $(K_a^+)^{-1} = K_a^-$, and each symmetric key to itself, $(K_{ab})^{-1} = K_{ab}$.

The subterm relation, $\sqsubseteq$, which we borrow from [19], is the least relation such that: $M \sqsubseteq M$; $M \sqsubseteq \{\!|M_1|\!\}_K$ if $M \sqsubseteq M_1$; $M \sqsubseteq M_1; M_2$ if $M \sqsubseteq M_1$ or $M \sqsubseteq M_2$. Notice that, for any $K \in \mathsf{K}$, $K \sqsubseteq \{\!|M|\!\}_K$ only if $K \sqsubseteq M$. We extend $\sqsubseteq$ to a homomorphism over sets of terms in the expected manner. A message is *atomic* if it is not an encrypted term or a concatenated one. A message M_0 is a *component* of M, if $M_0 \sqsubseteq M$, M_0 is not a concatenated term, and for every $M_1 \neq M_0$ such that $M_0 \sqsubseteq M_1 \sqsubseteq M$ implies that M_1 is a concatenated term.

Definition 1. *Let* R *be a set of concrete, object-level messages and let* $\mathbb{I}$ *be a mapping from* A *to* R*. Further, let* $M_1 \approx M_2$ *abbreviate* $\mathbb{I}(M_1) = \mathbb{I}(M_2)$*. Then,* $\mathbb{I}$ *is an* implementation *of* A *iff the following holds:*

$$\forall M, M_1, M_2 \in \mathsf{A}.\ M_1 \approx M_2 \to M_1; M \approx M_2; M \quad (1)$$

$$\forall M, M_1, M_2 \in \mathsf{A}.\ M_1 \approx M_2 \to M; M_1 \approx M; M_2 \quad (2)$$

$$\forall M_1, M_2 \in \mathsf{A}. \forall K \in \mathsf{K}.\ M_1 \approx M_2 \to \{\!|M_1|\!\}_K \approx \{\!|M_2|\!\}_K \quad (3)$$

$$\forall K, K' \in \mathsf{K}. \forall M \in \mathsf{A}.\ K \approx K' \to \{\!|M|\!\}_K \approx \{\!|M|\!\}_{K'} \quad (4)$$

$$\forall M_1, M_2 \in \mathsf{Agent}.\ M_1 = M_2 \leftrightarrow M_1 \approx M_2 \quad (5)$$

$$\forall M_1, M_2 \in \mathsf{Nonce}.\ M_1 = M_2 \leftrightarrow M_1 \approx M_2 \quad (6)$$

$$\forall M_1, M_2 \in \mathsf{Timestamps}.\ M_1 = M_2 \leftrightarrow M_1 \approx M_2 \quad (7)$$

$$\forall M_1, M_2 \in \mathsf{K}.\ M_1 = M_2 \leftrightarrow M_1 \approx M_2 \quad (8)$$

Axioms (5)—(8) specify that the identification of different (abstract) messages can only occur between messages of different types. If two nonces are considered to be different on the abstract level then we also assume that their implementations are different. However, we can still identify, for instance, the implementation of a nonce with the implementation of an encrypted message or a timestamp.

2.2 Strands, Strand Spaces and Bundles

A *strand* is a sequence of nodes. Every node in a strand denotes a communicating event, where transmission (respectively reception) of a term M is denoted as $+M$ (respectively $-M$). Accordingly, we will call a node n either *positive* or *negative* and will use $\mathsf{msg}(n)$ and $\mathsf{sign}(n)$ to respectively obtain the node term and the communication event thereby denoted.

Let s be a strand and let $\langle s, i\rangle$ and $\langle s, i+1\rangle$ denote the i-th and the $i+1$-th nodes of s. Then $\langle s, i\rangle \Rightarrow \langle s, i+1\rangle$. $\Rightarrow^+$ and $\Rightarrow^*$ are used to respectively denote the transitive and the transitive-reflexive closure of $\Rightarrow$. $\langle s_s, i\rangle \rightarrow \langle s_r, j\rangle$ denotes inter-strand communication, $s_s \neq s_r$. It requires the nodes to be both complementary, in terms of polarity, $\mathsf{sign}(\langle s_s, i\rangle) = +$ while $\mathsf{sign}(\langle s_r, j\rangle) = -$, and matching, in terms of the message being exchanged, $\mathsf{msg}(\langle s_s, i\rangle) = \mathsf{msg}(\langle s_r, j\rangle)$.

A strand represents a protocol run from the local perspective of a participant. If the participant is honest, the strand, as well as each individual strand node, is said to be *regular* and *penetrator* otherwise. A *strand space* Σ is a set of strands, where $\Rightarrow$ and $\rightarrow$ impose a graph structure on the nodes of Σ. The pair $(\Sigma, \mathcal{P})$ is said to be an *infiltrated strand space*, whenever Σ is a strand space and $\mathcal{P} \subseteq \Sigma$, such that every $p \in \mathcal{P}$ is a penetrator strand.

A *penetrator strand* hooks together several penetrator traces, which are actions that characterise the abilities of the penetrator, according to the standard Dolev-Yao model. Our extension to the penetrator model consists of adding the **I** action, which captures the penetrator's ability of causing a type flaw attack.

Definition 2 (Penetrator Trace). *Let* $\mathsf{K_P}$ *denote the keys initially known to the penetrator; then, a* penetrator trace *is (as given in [19]) one of the following:*

- **(M)essage***:* $\langle +M\rangle$*, where* $M \in \mathsf{Agent}$ *or* $M \in \mathsf{Nonce}$*;*
- **(K)ey***:* $\langle +K\rangle$*, where* $K \in \mathsf{K_P}$*;*
- **(F)lush***:* $\langle -M\rangle$*;*
- **(T)ee***:* $\langle -M, +M, +M\rangle$*;*
- **(C)oncatenation***:* $\langle -M_1, -M_2, +M_1; M_2\rangle$*;*
- **(S)eparation***:* $\langle -M_1; M_2, +M_1, +M_2\rangle$*;*
- **(E)ncryption***:* $\langle -K, -M, +\{\!|M|\!\}_K\rangle$*; and*
- **(D)ecryption***:* $\langle -\{\!|M|\!\}_K, +K, +M\rangle$*;*

plus the action ***I*** *defined by:*

> **(I)mplementation***:* $\langle -M, +M'\rangle$*, provided that* $M \approx M'$ *(The messages* M *and* M' *are respectively called* camouflaged *and* spoofed*).*

In an infiltrated strand space, $(\Sigma, \mathcal{P})$, penetrator traces of type **M** are assumed to be unable to model unguessable nonces and so the penetrator can build masquerading messages using **M**, **K**, **F**, **T**, **C**, **S**, **E**, **D** and **I**, only. Thus, to persuade a honest principal that a camouflaging message can be accepted to spoof some other, the penetrator has the obligation of proving that, at the object level, the implementation of both messages are equal.

Definition 3. *A finite, acyclic graph, $\mathcal{B} = \langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$, is a* bundle *if for every $n_2 \in \mathcal{N}$, i) if* $\mathsf{sign}(n_2) = -$*, then there is a unique $n_1 \in \mathcal{N}$ with $n_1 \rightarrow n_2$; and ii) if $n_1 \Rightarrow n_2$ then $n_1 \in \mathcal{N}$ and $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i+1\rangle$. Let $\mathcal{B}$ be a bundle, then $\prec_{\mathcal{B}}$ and $\preceq_{\mathcal{B}}$ denote respectively the transitive and the transitive-reflexive closure of* $(\rightarrow \cup \Rightarrow)$.

A bundle is said to be *regular* if it contains no penetrator strands and penetrator otherwise. A message M is said to be a component of a node n if it is a component of $\mathsf{msg}(n)$. A node n is said to be an **M**, **K**, etc. node if n lies on a penetrator strand with a trace of kind **M**, **K**, etc.

In the original approach, the notion of a node originating a term is a syntactical property of strands [19]:

> "Let $\mathcal{B} = \langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$ be a bundle. An unsigned term M *originates* at a node $n \in \mathcal{N}$ if $\mathsf{sign}(n) = +$; $M \sqsubseteq \mathsf{msg}(n)$; and $M \not\sqsubseteq \mathsf{msg}(n')$, for every $n' \Rightarrow^{+} n$. M is said to be *uniquely originating* if it originates on a unique $n \in \mathcal{N}$."

So, if for instance the term is a nonce, depending on whether the nonce occurs first in a received or in a sent message of a strand, the associated principal can either reuse the nonce received from elsewhere or create a fresh one. Here, we have to lift this notion of nodes that originate terms to match our extended version of strand spaces. This is because when considering the concrete representation of messages it is possible that different (abstract) messages share the same implementation. Hence, considering the reception of a message by a principal on the abstract level allows us to conceal the fact that that principal gets also knowledge about other messages sharing the same implementation. The only principal caring about this ambiguity at the implementation level is, of course, the penetrator, since he may want to tunnel a message camouflaged as a different one along a protocol run. Upon the reception of the camouflaged message he is also aware of the spoofed message and the **I**-rule allows him to switch between these different (abstract) messages.

The following definition allows us to collect the information about the camouflage actions that the penetrator has performed up to a specific node in the bundle.

Definition 4. *Let $\mathcal{B} = \langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$ be a bundle and let n_1 and n_2 be nodes in $\mathcal{B}$ ($n_1, n_2 \in \mathcal{N}$). Whenever there is an **I** trace in $\mathcal{B}$ with nodes n_1 and n_2 having signed terms $-M$ and $+M'$, respectively, with $M \not\equiv M'$, we write $itr_{\mathcal{B}}(n_1, -M, n_2, +M')$.*

Definition 5 (Scope). *Let $\mathcal{B} = \langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$ be a bundle and let $n \in \mathcal{N}$. Then, the* set of spoofed actions up to *node n, called the* scope *of n, wrt $\mathcal{B}$, written* $\mathsf{Sc}_{\mathcal{B}}(n)$*, is given by:*

$$\mathsf{Sc}_{\mathcal{B}}(n) = \{M \approx M' : \exists n_1, n_2 \text{ such that } itr_{\mathcal{B}}(n_1, -M, n_2, +M') \text{ and } n_2 \prec_{\mathcal{B}} n\}$$

The knowledge of the camouflaged messages is crucial for the refined notion of originating nodes:

Definition 6. *Let $\mathcal{B} = \langle \mathcal{N}, (\rightarrow \cup \Rightarrow)\rangle$ be a bundle and let $n \in \mathcal{N}$ be a node in $\mathcal{B}$. An atomic term M* originates *at n iff:*

- $\mathsf{sign}(n) = +$;
- $M \sqsubseteq \mathsf{msg}(n)$;
- $M \not\sqsubseteq \mathsf{msg}(n')$ *for every* $n' \Rightarrow^{+} n$ *and*
- $M \not\sqsubseteq \mathsf{Sc}_{\mathcal{B}}(n')$ *if* $\exists n', M_1, M_2$ *such that* $itr_{\mathcal{B}}(n', -M_1, n, +M_2)$.

Before concluding this section, we illustrate the notions above with example descriptions of two protocol attack.

2.3 Woo and Lam's Protocol Attack

For brevity, protocols will be specified using Alice and Bob notation. So a protocol is given by a sequence of steps, each of the form $q.\ A \rightarrow B : M$, meaning that, at step q, agent A sends message M to agent B, which B receives. We use S and Spy to refer to the server and the penetrator, respectively. Consider the Woo and Lam π_1 Protocol [20]:

$$\begin{array}{l}
1.\ A \rightarrow B : A \\
2.\ B \rightarrow A : N_b \\
3.\ A \rightarrow B : \{\!|A; B; N_b|\!\}_{K_{as}} \\
4.\ B \rightarrow \mathsf{S}\ : \{\!|A; B; \{\!|A; B; N_b|\!\}_{K_{as}}|\!\}_{K_{bs}} \\
5.\ \mathsf{S}\ \rightarrow B : \{\!|A; B; N_b|\!\}_{K_{bs}}
\end{array}$$

The Woo and Lam π_1 protocol exhibits a type flaw attack, which is based on the property that the implementation of nonces can be confused with the implementation of encrypted messages. We capture this property by refining our theory of implementation for this protocol with the corresponding additional axiom:

$$\forall N \in \mathsf{Nonce}. \exists K \in \mathsf{K}, M \in A. \quad N \approx \{\!|M|\!\}_K \tag{9}$$

Then, the penetrator bundle illustrating a type flaw attack on this protocol is as follows:

$$\begin{tikzcd}
\mathsf{Spy} & B \\
p_1\,(+A) \arrow[r] \arrow[d, Rightarrow] & s_1\,(-A) \arrow[d, Rightarrow] \\
p_2\,(-N_b) \arrow[d, Rightarrow, "\mathbf{I}-trace"'] & s_2\,(+N_b) \arrow[l] \arrow[d, Rightarrow] \\
p_3\,(+\{\!|M|\!\}_K) \arrow[r] & s_3\,(-\{\!|M|\!\}_K) \arrow[d, Rightarrow] \\
p_4\,(-\{\!|A; B; \{\!|M|\!\}_K|\!\}_{K_{bs}}) \arrow[d, Rightarrow, "\mathbf{I}-trace"'] & s_4\,(+\{\!|A; B; \{\!|M|\!\}_K|\!\}_{K_{bs}}) \arrow[l] \arrow[d, Rightarrow] \\
p_5\,(+\{\!|A; B; N_b|\!\}_{K_{bs}}) \arrow[r] & s_5\,(-\{\!|A; B; N_b|\!\}_{K_{bs}})
\end{tikzcd}$$

This example penetrator bundle contains two $\mathbf{I}$ traces: the first is $\langle -N_b, +\{\!|M|\!\}_K\rangle$ and the second $\langle -\{\!|A; B; \{\!|M|\!\}_K|\!\}_{K_{bs}}, +\{\!|A; B; N_b|\!\}_{K_{bs}}\rangle$.

The node p_3 with $\mathsf{msg}(p_3) = \{\!|M|\!\}_K$ is an originating node for M and K because both messages do not occur in the previous message, N_b, of the first **I** trace (when read from top to bottom). Nor do they occur in the scope of p_2, with $\mathsf{msg}(p_2) = N_b$ (which is actually empty). In order to justify the use of the **I** trace, we have to guarantee that $N_b \approx \{\!|M|\!\}_K$. Since M and K originate at this **I** trace we do not care about the resulting values of M and K in the spoofed message. Selecting the witnesses (skolem-terms) of the existential quantified variables in (9) the proof obligation is a simple instance of (9).

By contrast consider the second **I** trace, even though N_b does not occur in $\mathsf{msg}(p_4)$, the node p_5 with $\mathsf{msg}(p_5) = \{\!|A; B; N_b|\!\}_{K_{bs}}$ is *not* an originating node for N_b. However, N_b occurs in the scope of node p_5:

$$\mathsf{Sc}_{\mathcal{B}}(p_5) = \{N_b \approx \{\!|M|\!\}_K\}.$$

This context knowledge in combination with axiom (2) guarantees the property

$$\{\!|A; B; N_b|\!\}_{K_{bs}} \approx \{\!|A; B; \{\!|M|\!\}_K|\!\}_{K_{bs}}$$

and therefore the soundness of the **I**-rule application.

2.4 The KP Protocol

Snekkenes [18] discusses the KP protocol, which is subject to a type flaw attack. The protocol is as follows:

$$\begin{array}{lrl}
1. & A \rightarrow B & : A; N_a; B \\
2. & B \rightarrow KDC & : A; N_a; B; N_b \\
3. & KDC \rightarrow B & : \{\!|K_S; A; N_b; \{\!|A; B; N_a; K_S|\!\}_{K_{as}}|\!\}_{K_{bs}} \\
4. & B \rightarrow A & : \{\!|A; B; N_a; K_S|\!\}_{K_{as}}; \{\!|N_a; N_c; B|\!\}_{K_S} \\
5. & A \rightarrow B & : \{\!|N_c; A|\!\}_{K_S}
\end{array}$$

The attack is based on the facts that:

1. on step 3, the principal B extracts $\{\!|A; B; N_a; K_S|\!\}_{K_{as}}$ from the component encrypted under K_{bs} and then, on step 4, sends this extracted message to A;
2. keys share common implementations with agents and encrypted messages; and
3. therefore the implementation of the message in step 3 can be misinterpreted (in a different run) as the implementation of the first component of the message in step 4.

Similar to the previous example, an appropriate theory of message implementations contain axioms to deduce the fact that keys (generated dynamically by the server) and agents (or encrypted messages, respectively) potentially share the same implementation:

$$\forall A \in \mathsf{Agent}.\ \exists K \in \mathsf{K}.\quad A \approx K \tag{10}$$

$$\forall K \in \mathsf{K}.\ \exists K' \in \mathsf{K}, M \in A.\quad K \approx \{\!|M|\!\}_{K'} \tag{11}$$

$$\begin{tikzcd}
KDC & B & \mathsf{Spy} & A \\
 & & p_1(+B;N'_b;A) \arrow[r] \arrow[d, Rightarrow] & s'_1(-B;N'_b;A) \arrow[d, Rightarrow] \\
 & & p_2(-B;N'_b;A;N'_a) & s'_2(+B;N'_b;A;N'_a) \arrow[l] \\
 & & \cdots & \\
 & s_1(-A;N'_a;B) \arrow[d, Rightarrow] & p_3(+A;N'_a;B) \arrow[l] & \\
k_1(-\ldots) \arrow[d, Rightarrow] & s_2(+A;N'_a;B;N'_b) \arrow[l] \arrow[d, Rightarrow] & & \\
k_2(-\ldots) \arrow[r] & s_3(-\left\{\!\left| K_S;A;N_b;\{\!|A;B;N'_a;K_S|\!\}_{K_{as}} \right|\!\right\}_{K_{bs}}) \arrow[d, Rightarrow] & & \\
 & s_4(+\{\!|A;B;N'_a;K_S|\!\}_{K_{as}};\{\!|N'_a;N_d;B|\!\}_{K_S}) \arrow[r] & p_4(-\{\!|A;B;n'_A;K_S|\!\}_{K_A};\{\!|N'_a;N_d;B|\!\}_{K_S}) & \\
 & & \cdots & \\
 & & p_5(+\{\!|A;B;N'_a;K_S|\!\}_{K_{as}}) \arrow[d] & \\
 & & p_6(-\{\!|A;B;N'_a;K_S|\!\}_{K_{as}}) \arrow[d, Rightarrow, "\mathbf{I}-\mathit{trace}"'] & \\
 & & p_7(+\{\!|K'_S;B;N'_a;K_S|\!\}_{K_{as}}) \arrow[d] & \\
 & & p_8(-\{\!|K'_S;B;N'_a;K_S|\!\}_{K_{as}}) \arrow[d, Rightarrow, "\mathbf{I}-\mathit{trace}"'] & \\
 & & p_9(+\left\{\!\left|K'_S;B;N'_a;\{\!|M|\!\}_K\right|\!\right\}_{K_{as}}) \arrow[r] & s'_3(-\left\{\!\left|K'_S;B;N'_a;\{\!|M|\!\}_K\right|\!\right\}_{K_{as}}) \arrow[d, Rightarrow] \\
 & & p_{10}(-\{\!|M|\!\}_K;\{\!|N'_b;N'_d;A|\!\}_{K'_S}) \arrow[d, Rightarrow, "\mathbf{I}-\mathit{trace}"'] & s'_4(+\{\!|M|\!\}_K;\{\!|N'_b;N'_d;A|\!\}_{K'_S}) \arrow[l] \\
 & & p_{11}(+K_S;\{\!|N'_b;N'_d;A|\!\}_{K'_S}) & \\
 & & \cdots & \\
 & s_5(-\{\!|N_d;A|\!\}_{K_S}) & p_{12}(+\{\!|N_d;A|\!\}_{K_S}) \arrow[l] &
\end{tikzcd}$$

Fig. 1. Type flaw attack in the KP-protocol in [18]

Fig. 1 illustrates the resulting attack in the strand space notation using our notion of **I**-traces. This example contains three **I**-traces. Node p_7 originates the key K'_S on the penetrator strand. Since this key has not been seen by the regular principles, the penetrator is free to choose its value and axiom (10) applies. Similar arguments hold for p_9 originating M and K using axiom (11). The scope of p_{10} is $\{A \approx K'_S, K_S \approx \{\!|M|\!\}_K\}$ which trivially implies that the messages in p_{10} and p_{11} share the same implementation.

3 Feasibility of Type Flaw Attacks

3.1 Extending Strand Space Theory

In the last section we extended the strand space notation by an axiomatic specification of messages at an implementation level and by an additional **I** penetrator strand connecting this implementation level with the abstract level of strand spaces. While the original purpose of this extension was to provide a uniform representation language for protocol runs (potentially containing type flaw attacks), it is also interesting to investigate how the verification methodology of strand spaces can be lifted to our extended approach.

A central part of this methodology are outgoing (or ingoing, respectively) authentication tests [10]. Suppose, a nonce N is only known by an honest principal. If this principal sends this nonce N as part of an encrypted message $\{\!|M|\!\}_K$ around and later on receives the same nonce in clear text (i.e. outside its original encryption by K) then somebody must have decrypted the message possessing the appropriate key K^{-1}. If this key is only knowledgeable to a single person (and has not been sent around) then we can conclude that this particular person has been involved in processing the message.

The question arises whether this property holds also including **I**-traces. Suppose, N occurs outside of its encryption by K in the second node of an **I**-trace. We consider two cases. First, suppose N originates at the second node of the **I**-trace. In this case the message used to create N has been created without any knowledge of N since otherwise N would be part of the scope of the particular **I**-trace node, which would contradict our assumption that N originates at the **I**-trace. Now, suppose that N does not originate at the **I**-trace node. Either N is (unencrypted) also part of the received first message of the **I**-trace and we can use the argument of the original strand space theorem that this cannot happen or there is a type confusion going on and both messages of the **I**-trace share the same interpretation. Since the equality of implementation has to be implied by the scope of the node (and by assumption N occurs only encrypted with K in the scope), this can only be true iff the implementation of the encryption satisfies some non-standard properties (i.e. it can be simulated by some other operation on messages without K^{-1}).

3.2 Implementation

In our approach we separate the protocol level operating on abstract messages from its implementation level. This is in contrast to many other approaches that encode implementation details in an equality theory on (abstract) messages. The benefit is that we can use (arbitrary) algebraic specifications to formalise properties (in particular equality and inequality) of the message implementation. This knowledge about the implementation is used to verify side conditions of **I**-traces. In order to apply **I**-traces in the penetrator bundle we have to make sure that both messages of the trace share the same implementation. This is a task that can be given to an automated theorem prover or to specialised deduction system incorporating domain specific knowledge (e.g. SMT-provers).

Axioms (1)—(8) reflect only the minimal requirements to an implementation. Typically, one would extend this set of axioms by a more detailed specification of how messages and their operations are implemented. It is easy to formalise a message implementation theory that takes care of the length of messages, i.e. that assumes that all types of atomic messages have a specific length. Then, reasoning about such a theory requires arithmetic (and in the worst case properties of least common multiples). Another possibility is use Meadow's probabilistic approach [15] to reason about the equality of the implementations of two messages. However, this is still future work.

The axiomatisation given in our examples is idealistic in a sense that we define possible type flaws regardless of the concrete instances of a nonce or key. It might be the case that there are some nonces that do not share the implementation with any encrypted message. However, incorporating this sort of probabilistic information would cause a far more complex specification of the implementation theory and therefore we stick to a generalisation of such properties which result in a worst case analysis.

The execution of the **I**-rule is trivial as it only forwards a received message duping the receiver of the message that it would be a different abstract message. The crucial question is then: how likely is it that a penetrator can betray the receiver in such a way? The philosophy behind the design of strand spaces was that the independent choice of two nonces will result in two different nonces. This philosophy is implemented in demanding that nonces have to be uniquely originating in bundles. Similarly, we typically assume that choosing independently two different abstract messages will also result in two different messages on the implementation level.

I traces impose the restriction on the occurring messages that their implementations have to be equal. In contrast to standard bundles, as introduced in [19], the property of being a bundle depends on the theory of implementations and for each proposed instance $\langle -t, +t' \rangle$ of an **I** trace we have to verify the constraints that $t \approx t'$. An atomic message originating in t' can be "freely" chosen by the penetrator who selects an appropriate instance the implementation of which is equal to the corresponding bits of the implementation of t. If the message does not originate in t' its value is fixed by the bundle at some other node and we have to prove that its implementation is equal to the corresponding bits of the implementation of t.

4 A Method for Protocol Repair

Repairing a security protocol susceptible to a type flaw attack can be, accordingly, achieved at the abstract level, at the concrete level or at a combination thereof. Protocol repair at the abstract level involves changing the structure of the camouflaged message so that it can no longer spoof the other (or vice versa). By contrast, protocol repair at the concrete level is achieved by modifying the theory of implementation underlying the input faulty protocol; these changes ought to guarantee that each application of the **I**-rule is no longer sound. We now elaborate on protocol repair at the abstract level.

4.1 Protocol Repair at the Abstract Level

The intruder can elaborate a type flaw attack on a protocol whenever he is able to make a protocol principal accept a message of one type, M, as a message of another, M'. In that case, M and M' share the same implementation, $M \approx M'$. Thus, to remove the protocol type flaw, it suffices to break $M \approx M'$.

Following [12], we approach protocol repair by translating some of Abadi and Needham's informal principles for the design of security protocols [1] into formal requirements on sets of protocol steps. In particular, here we focus on principle 10: A message is *properly encoded* if it is possible to deduce that the message belongs to a protocol, including the protocol step, and in fact to a particular run of the protocol [1].

We attempt to achieve proper encoding of a message M by rearranging its structure while keeping its meaning intact. If this operation does not suffice to break the confusion between M and M', we may then insert into M vacuous terms, tags actually, as in [11]. Finally, when incurring on these changes, we make sure that the new protocol message does not clash with some other. The following definitions are used in the definition of our patch method.

Definition 7 (Visible Content). *The* visible content $ct_S(M)$ *of a message* M *wrt a set of keys* S *is given by:*

$$\begin{aligned} ct_S(M) &= \{M\} && \textit{if } M \textit{ is atomic} \\ ct_S(M_1; M_2) &= ct_S(M_1) \cup ct_S(M_2) && \\ ct_S(\{\!|M|\!\}_K) &= \begin{cases} ct_S(M) & \\ \emptyset & \end{cases} && \begin{matrix} \textit{if } K^{-1} \in S \\ \textit{if } K^{-1} \notin S \end{matrix} \end{aligned}$$

Definition 8. *Two messages* M *and* M' *are* equivalent under rearrangement, $M \equiv M'$ *for short, iff* $ct_S(M) = ct_S(M')$ *for all sets of keys* S.

Example 1. $\{\!|A; B; M|\!\}_{K_{as}} \equiv \{\!|M; A; B|\!\}_{K_{as}}$, because either $K_{as} \notin S$ then

$$ct_S(\{\!|A; B; M|\!\}_{K_{as}}) = \emptyset = ct_S(\{\!|M; A; B|\!\}_{K_{as}})$$

or $K_{as} \in S$ then

$$\begin{aligned} ct_S(\{\!|A; B; M|\!\}_{K_{as}}) &= ct_S(A; B; M) = \{A, B\} \cup ct_S(M) \\ = ct_S(M) \cup \{A, B\} &= ct_S(M; A; B) = ct_S(\{\!|M; A; B|\!\}_{K_{as}}) \end{aligned}$$

Name: *type_flaw*
Input: $P \in \Sigma$, $\mathcal{B}$, $\mathcal{B}_R$
Preconditions:

`%` Spy reuses cypher-text $\{\!|M|\!\}_K$:
$\exists i, i', j, M, K.$
$\langle s_r, i\rangle \prec_{\mathcal{B}} \langle s_p, i'\rangle \preceq_{\mathcal{B}} \langle s_q, j\rangle \wedge$
$\{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_r, i\rangle) \wedge \{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_p, i'\rangle) \wedge \{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_q, j\rangle)$

`%` Strand s_p corresponds to the penetrator, while
`%` Strand s_q to the principal being deceived.
$\wedge \langle s_r, i\rangle$ and $\langle s_q, j\rangle$ are regular but $\langle s_p, i'\rangle$ is not
$\wedge \mathsf{sign}(\langle s_r, i\rangle) = + = \mathsf{sign}(\langle s_p, i'\rangle)$ but $\mathsf{sign}(\langle s_q, j\rangle) = -$

`%` The message being replayed lies on an I-trace
$\wedge\, itr_{\mathcal{B}}(s_p, -\{\!|M|\!\}_K, s'_p, +\{\!|M'|\!\}_{K'})$ with $s'_p \rightarrow s_q$

Repair:

`%` Break similarity of $\{\!|M|\!\}_K$:
select M'' such that $\{\!|M|\!\}_K \leq_{\mathsf{A}_0} \{\!|M''|\!\}_K$, where $\mathsf{A}_0 \subseteq \mathsf{Tags}$ is a minimal set of tags, and such that $\{\!|M''|\!\}_K$ is collision free with respect to
$L = \{\{\!|M'|\!\}_{K'} : \{\!|M'|\!\}_{K'} \sqsubseteq \mathsf{msg}(n), n \in \mathcal{B}_R\}$.
Replace $\{\!|M''|\!\}_K$ for $\{\!|M|\!\}_K$ in P.

Fig. 2. The *type-flaw* abstract-level patch method

Definition 9 (Monotonicity). *Let M, M' be messages and let $\mathsf{A}_0 \subseteq \mathsf{Tags}$ be a set of tags. $M \leq_{\mathsf{A}_0} M'$ iff $ct_S(M) \subseteq ct_S(M')$ and $ct_S(M') \subseteq ct_S(M) \cup ct_S(\mathsf{A}_0)$ for all $S \subseteq \mathsf{K}$.*

Definition 10 (Collision Freeness). *Let S be a set of keys, let M be a message and let $\mathsf{A}' \subseteq \mathsf{A}$ be a set of messages. M is* collision free *with respect to A' iff for all $M' \in ct_S(M)$ and for all $M'' \in \mathsf{A}'$ it is the case that $M' \not\approx M''$.*

Our method for the repair of a security protocol susceptible to a replay attacks is shown in Fig. 2.[2] It is given as a patch method [12]. A *patch method* is a 4-tuple (name, input, preconditions, patch). The first component is the *name* of the method. The second component is the *input*, the description of faulty protocol P, a bundle $\mathcal{B}$ describing the attack, and a representative bundle $\mathcal{B}_R$ describing the intended run of the protocol. The third component is the *preconditions*, a formula written in a meta-logic that the input objects must satisfy. We use these preconditions to predict whether the associated patch will make the protocol no longer susceptible to the attack. The fourth component is the *patch*, a procedure specifying how to mend the input protocol.

When input the Woo and Lam π_1 protocol, and upon the proviso that:

$$\forall A \in \mathsf{Agent}, N \in \mathsf{Nonce}.\, A \not\approx N \tag{12}$$

[2] Notice that we consider encrypted message components only, since a replay attack builds on the re-use of non-trivial messages.

$$\forall M_1, M_2, M_3, M_4 \in \mathsf{A}.\, M_1 \approx M_3 \wedge M_2 \approx M_4 \rightarrow M_1; M_2 \approx M_3; M_4 \tag{13}$$

the abstract-level *type_flaw* method repairs it, yielding:[3]

$$\begin{array}{l}
1.\ A \rightarrow B : A \\
2.\ B \rightarrow A : N_b \\
3.\ A \rightarrow B : \{\!|A; B; N_b|\!\}_{K_{as}} \\
4.\ B \rightarrow \mathsf{S}\ : \{|\boxed{\{|A; B; N_b|\}_{K_{as}}; A; B}|\}_{K_{bs}} \\
5.\ \mathsf{S}\ \rightarrow B : \{\!|A; B; N_b|\!\}_{K_{bs}}
\end{array}$$

To be able to apply the **I** rule with this mended protocol, the penetrator would have to prove:

$$\forall A, B, N_b.\, \exists M', K'.\ \{\!|A; B; N_b|\!\}_{K_{bs}} \approx \{\!|\{\!|M'|\!\}_{K'}; A; B|\!\}_{K_{bs}}$$

which, by (13), requires in turn that:

$$\forall A, B, N_b.\, \exists M', K'.\ (A \approx \{\!|M'|\!\}_{K'} \ \wedge\ B = A \ \wedge\ N_b \approx B)$$

Which contradicts (12). We now elaborate on protocol repair at the concrete level.

4.2 Protocol Repair at the Concrete Level

We assume an underlying theory of message implementation, $\mathcal{I}$. $\mathcal{I}$ accommodates axioms (1)—(8) as well as others that are particular to a concrete development of messages. Example of these kinds of additional axioms may involve:

$\forall A \in \mathsf{Agent}, K \in \mathsf{K}.\ A \not\approx K$	Disjunction of Agent 1
$\forall A \in \mathsf{Agent}, N \in \mathsf{Nonce}.\ A \not\approx N$	Disjunction of Agent 2
$\forall A \in \mathsf{Agent}, T \in \mathsf{Timestamps}.\ A \not\approx T$	Disjunction of Agent 3
$\forall A \in \mathsf{Agent}, M \in \mathsf{A}, K \in \mathsf{K}.\ A \not\approx \{\!|M|\!\}_K$	Disjunction of Agent 4
$\forall M, M' \in \mathsf{A}.\ M \not\approx M; M'$	Monotonicity of Concatenation 1
$\forall M, M' \in \mathsf{A}.\ M \not\approx M'; M$	Monotonicity of Concatenation 2

In general, axioms other than basic are either about type disjunction or about properties of compound messages, such as type length, c.f. (13).

Clearly, at the implementation level, a type flaw attack is possible because $\mathcal{I}$ contains axioms that overlook a means of explicitly providing type disjunction. For instance, the type flaw attack on the Woo and Lam π_1 protocol is justified when $\mathcal{I}$ contains axiom (9), where an encrypted message can be confused with a nonce. Repairing a protocol at the concrete level can thus be achieved by removing from $\mathcal{I}$ these kinds of overlooking axioms while explicitly adding their negation. This way the **I** trace is no longer applicable. The method presented in Fig. 3 follows this idea. It makes use of the following definitions:

[3] Protocol changes are enclosed within a solid box to ease reference.

Name: *type_flaw*
Input: $P \in \Sigma$, $\mathcal{B}$, $\mathbf{B}_R$
Preconditions:

`% Spy reuses cypher-text` $\{\!|M|\!\}_K$:
$\exists i, i', j, M, K.$
$\langle s_r, i\rangle \prec_{\mathcal{B}} \langle s_p, i'\rangle \preceq_{\mathcal{B}} \langle s_q, j\rangle \wedge$
$\{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_r, i\rangle) \wedge \{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_p, i'\rangle) \wedge \{\!|M|\!\}_K \sqsubseteq \mathsf{msg}(\langle s_q, j\rangle)$

`% Strand` s_p `corresponds to the penetrator, while`
`% Strand` s_q `to the principal being deceived.`
$\wedge \langle s_r, i\rangle$ and $\langle s_q, j\rangle$ are regular but $\langle s_p, i'\rangle$ is not
$\wedge \mathsf{sign}(\langle s_r, i\rangle) = + = \mathsf{sign}(\langle s_p, i'\rangle)$ but $\mathsf{sign}(\langle s_q, j\rangle) = -$

`% The message being replayed lies on an I-trace`
$\wedge\, itr_{\mathcal{B}}(s_p, -\{\!|M|\!\}_K, s'_p, +\{\!|M'|\!\}_{K'})$ with $s'_p \rightarrow s_q$

Repair:

Let $\mathbf{A} = \{Ax : Ax \vdash (M \approx M')\downarrow_{\mathcal{R}_B \cup \mathcal{R}_C}$, with $Ax \in \mathcal{I}$ and $M \approx M' \in \mathsf{Sc}_{\mathcal{B}}(s'_p)\}$
Let $\mathcal{I} \leftarrow (\mathcal{I} \setminus \mathbf{A}) \cup \{\mathsf{Cl}(\neg\mathsf{Mtrx}(Ax)) : Ax \in \mathbf{A}\}$

Fig. 3. The *type-flaw* concrete-level patch method

Definition 11. *A* term-rewriting system *(TRS), $\mathcal{R}$, is a set of* rewrite-rules*, each of which is of the form $L :\Rightarrow R$. A rule $L :\Rightarrow R$ applies to a term t (in A) if a subterm s of t matches L with some substitution σ of terms (in A) for variables appearing in L. The rule is applied by replacing s in t with the corresponding right-hand side $R\sigma$ of the rule. We write $t :\Rightarrow t'$ to indicate that the term t is transformed into t' by an application of this rewriting. A TRS is* terminating *iff the sequence $t_1 :\Rightarrow \ldots :\Rightarrow t_n$ is finite. t_n is the* normal form *of t_1. Let $\mathcal{R}$ be a terminating TRS; then, we write $t\downarrow_{\mathcal{R}}$ to denote the normal form of t wrt $\mathcal{R}$.*

Definition 12. *Let A be a first-order logic formula. Then $\mathsf{Mtrx}(A)$ is the matrix of A, a formula just as A except that it contains no quantifiers, while $\mathsf{Cl}(A)$ returns the closure of A, a formula obtained from A by attaching a universal quantifier to it for each of its free variables.*

Since we reason backwards, from a formula to the axioms, implications can be used as rewrite-rules from right to left. We use $\mathcal{R}_B$ and $\mathcal{R}_C$ to respectively denote the set of rewrite-rules gained from (1)—(8) and from the axioms stating properties of compound messages, such as (13).

When input the KP protocol, and under the assumption that we use (1)—(8) and (13) as rewrites, the concrete-level *type_flaw* patch method will remove (10) and (11) from $\mathcal{I}$, while inserting into it the following axioms:

$$\forall A \in \mathsf{Agent}, K \in \mathsf{K}.\ A \not\approx K$$
$$\forall K, K' \in \mathsf{K}, M \in \mathsf{A}.\ K \not\approx \{\!|M|\!\}_{K'}$$

5 Related Work

In the past several approaches have been developed to deal with potential type-flaw attacks. This work can be classified into two different areas. The first area is concerned with changing the representation of messages to prevent type-flaw attacks in the first place. Heather et al. [11] use a tagging of messages to identify the type of a message in a unique way. Considering original messages combined with their tagging as new atomic entities, such messages constitute a generated algebraic datatype with non-overlapping ranges of the constructors satisfying the most important properties of the abstract message theory. Since tags themselves will reveal information about a message to a penetrator there are several refinements of this tagging approach to minimize the set of subterms of a message that have to be tagged (e.g. [13,14]).

The second area is concerned with the verification of protocols that may contain type-flaw attacks. A prominent approach is to replace the standard representation of messages as a freely generated datatype by a more involved datatype dealing with equations between constructor terms. As a consequence terms representing messages have to be unified modulo a theory modelling the equality relation on messages [4,7,9]. While this approach assumes that only entire messages can be interpreted in different ways, Meadows [15,16] investigated the problem that the implementation of a message could be cut into pieces and one of such pieces might be used to mock another message, e.g. some part of a bit string representing an encrypted message is reused as a nonce. In her model messages are represented as bit streams. Based on an information flow analysis of the protocol and the knowledge of how abstract messages are represented as bit streams probabilities are computed as to how likely a message could be guessed (or constructed) by a penetrator. This is in contrast to our possibilistic approach in which we abstract from unlikely events, e.g. that independently guessing a key and a nonce will result in messages that share the same implementation. This is reflected in our notion of originating messages occurring on **I** traces. The strand space approach excludes protocol runs in which messages are not uniquely originating resulting in a possibilitic approach in a somewhat idealised world. In our approach, for instance, a nonce can be only camouflaged by a message which itself is camouflaged at some point with the help of the same nonce (c.f. definition 6).

6 Conclusions

We presented an extension of the strand space approach to deal with type flaw attacks. We separated the level of abstract protocols from the implementation level the properties of which are specified by algebraic specifications and thus can be explored by automated theorem provers. While in the first place we developed this extension as an underlying notation for repairing faulty protocols, it seems to be promising also to investigate how the theory of protocol verification based on strand spaces can be carried over to our extended notion. The extension of strand spaces allows us to formulate heuristics to repair faulty protocols (due to type flaw attacks). It is interesting to see that there are typically two ways to overcome type flaw attacks. On one hand we can change the implementation in order to avoid the particular equalities on implemented messages and on the other hand we can change the protocol on the abstract level in order avoid situations in which one can exploit these properties in the implementation.

References

1. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. IEEE Transactions on Software Engineering 22(1), 6–15 (1996)
2. Anderson, R., Needham, R.: Robustness principles for public key protocols. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 236–247. Springer, Heidelberg (1995)
3. Aura, T.: Strategies against replay attacks. In: Proceedings of the 10th IEEE Computer Security Foundations Workshop, CSFW 1997, pp. 59–69. IEEE Computer Society, Los Alamitos (1997)
4. Basin, D., Mödersheim, S., Viganò, L.: Algebraic intruder deductions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS, vol. 3835, pp. 549–564. Springer, Heidelberg (2005)
5. Basin, D.A., Mödersheim, S., Viganò, L.: Ofmc: A symbolic model checker for security protocols. International Journal of Information Security 4(3), 181–208 (2005)
6. Carlsen, U.: Cryptographic protocols flaws. In: Proceedings of the 7th IEEE Computer Security Foundations Workshop, CSFW 1994, pp. 192–200. IEEE Computer Society, Los Alamitos (1994)
7. Chevalier, Y., Rusinowitch, M.: Combining intruder theories. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 639–651. Springer, Heidelberg (2005)
8. Chevalier, Y., Vigneron, L.: Automated unbounded verification of security protocols. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 324–337. Springer, Heidelberg (2002)
9. Gao, H., Bodei, C., Degano, P.: A formal analysis of complex type flaw attacks on security protocols. In: Meseguer, J., Roşu, G. (eds.) AMAST 2008. LNCS, vol. 5140, pp. 167–183. Springer, Heidelberg (2008)
10. Guttman, J.-D., Thayer, F.-J.: Authentication tests and the structure of bundles. Theoretical Computer Science 283(2), 333–380 (2002)
11. Heather, J., Lowe, G., Schneider, S.: How to prevent type flaw attacks on security protocols. In: Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW 2000, pp. 255–268. IEEE Computer Society, Los Alamitos (2000)
12. López-Pimentel, J.C., Monroy, R., Hutter, D.: On the automated correction of security protocols susceptible to a replay attack. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 594–609. Springer, Heidelberg (2007)
13. Malladi, S., Alves-Foss, J.: How to prevent type-flaw guessing attacks on password protocols. In: Proceedings of the 2003 Workshop on Foundations of Computer Security (FCS 2003), pp. 1–12. Technical Report of University of Ottawa (2003)
14. Malladi, S., Alves-Foss, J., Heckendorn, R.: On preventing replay attacks on security protocols. In: Proceedings of the International Conference on Security and Management, ICSM 2002, pp. 77–83 (2002), `http://citeseer.ist.psu.edu/malladi02preventing.html` (retrieved February 1, 2007)

15. Meadows, C.: Identifying potential type confusion in authenticated messages. In: Foundations of Computer Security 2002 (2002)
16. Meadows, C.: A procedure for verifying security against type confusion attacks. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop, CSFW 2003, pp. 62–74. IEEE Computer Society, Los Alamitos (2003)
17. Moore, J.H.: Protocol failures in cryptosystems. Proceedings of the IEEE 76(5), 594–602 (1988); reprinted in: Simmons, G.J. (ed.)Contemporary Cryptology: The Science of Information Integrity, pp. 541–558. IEEE Computer Science Press, Los Alamitos (1991)
18. Snekkenes, E.: Roles in cryptographic protocols. In: Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 105–119. IEEE Computer Society Press, Los Alamitos (1992)
19. Thayer, F.-J., Herzog, J.-C., Guttman, J.-D.: Strand spaces: Proving security protocols correct. Journal of Computer Security 7(2-3), 191–230 (1999)
20. Woo, T.Y.C., Lam, S.S.: A lesson on authentication protocol design. Operating Systems Review 28(3), 24–37 (1994)

Finite Models in FOL-Based Crypto-Protocol Verification

Jan Jürjens[1] and Tjark Weber[2]

[1] Open University (UK), Microsoft Research, Cambridge,
and Robinson College (Univ. Cambridge)
http://www.jurjens.de/jan
[2] Computer Laboratory, University of Cambridge
tw333@cam.ac.uk

Abstract. Cryptographic protocols can only be secure under certain inequality assumptions. Axiomatizing these inequalities explicitly is problematic: stating too many inequalities may impair soundness of the verification approach. To address this issue, we investigate an alternative approach (based on first-order logic) that does not require inequalities to be axiomatized. A derivation of the negated security property exhibits a protocol attack, and absence of a derivation amounts to absence of the investigated kind of attack.

We establish a fragment of FOL strictly greater than Horn formulas in which the approach is sound. We then show how to use finite model generation in this context to prove the absence of attacks. To demonstrate its practicality, the approach is applied to several well-known protocols, including ones relying on non-trivial algebraic properties. We show that it can be used to deal with infinitely many principals (and thus sessions).

1 Introduction

Cryptographic protocol analysis often models operations on messages in terms of a free algebra. Encryption of a message m with a key k, for instance, may be represented as the term $e(k, m)$. Decryption can be represented either implicitly, by including a rule that says that if a principal knows a (symmetric) key k and an encrypted message $e(k, m)$, then he can also learn m, or explicitly, using a decryption operator d that is assumed to satisfy a cancellation rule $d(k, e(k, m)) = m$. The knowledge of a Dolev-Yao attacker is defined inductively: he knows any messages that are transmitted over an insecure channel, and he can learn new messages by performing, e.g., encryption, decryption (provided he knows the key), and by sending messages to other principals to learn their response.

Typically automated or interactive theorem proving is used with the free algebra model to establish security properties of protocols [1]. Security proofs, however, must make crucial use of freeness: to show that transmitting a message m over an insecure channel does not reveal a secret s, we of course have to assume that s is distinct from m. Since the attacker can encrypt, we also have to assume that s is distinct from $e(m, m)$, $e(m, e(m, m))$, etc. Moreover, to show that

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 155–172, 2009.

the attacker cannot learn s, we have to assume that the attacker's *only* means of gaining knowledge are according to the rules of the inductive definition, i.e., that his knowledge is understood as a least fixed point—because without this understanding, the attacker *might* know everything. Theorem provers usually implement an "open world" semantics, where only those statements are provably false that hold in no model of the axioms. To show secrecy, distinctness and least fixed-point axioms therefore have to be asserted explicitly in these systems.

This is sometimes done automatically [2]. Otherwise, however, it can be tedious and error-prone. Stating too few inequalities may impair completeness of the verification approach, and stating too many may impair soundness. One has to be careful not to assert erroneous axioms, which might render the protocol formalization inconsistent and allow one to prove anything (e.g., security of a protocol that is in fact insecure).

In this paper we investigate the alternative approach that one can negate the conjecture about a protocol's security: instead of showing that the attacker does not know a secret s, we attempt to prove that he knows s. A proof of this conjecture corresponds to an attack, and the proven absence of a proof (equivalently, by soundness and completeness of first-order logic, a counterexample to the conjecture) corresponds to security of the protocol.[1]

Note that this approach allows us to establish security of the protocol without stating any freeness or fixed-point axioms. It is related to the famous "closed world assumption", which was popularized by Prolog and is now also used in, e.g., the ProVerif protocol verifier [4]. The closed world assumption gives rise to the treatment of negation as failure: every statement that cannot be proved is considered false.

Thus model generation can be used to prove protocols secure: a simple 2-element model (in which m and s are interpreted differently) suffices to show that transmission of m does not necessarily reveal the secret s to the attacker. In Sect. 2, we will show that this is in fact sufficient to conclude secrecy of s in the free algebra model.

In Sect. 3, we demonstrate practicality of the approach by applying it to abstract first-order formalizations of three well-known protocols: the RSA Probabilistic Signature Scheme (RSA-PSS), the Wired Equivalent Privacy (WEP) protocol, and the Needham-Schroeder-Lowe (NSL) authentication protocol. Section 4 concludes.

2 FOL-Based Crypto-Protocol Verification: Derivations vs. Models

We consider protocol formalizations in first-order logic, as e.g. in [4,5]. We restrict ourselves to secrecy properties in this paper. Blanchet [6] discusses how authentication can be treated in this framework.

[1] A similar approach was taken in [3], although without investigating the questions we consider in this paper.

Central to the formalization of a protocol is a unary predicate knows (defined inductively) that describes which messages are known to a Dolev-Yao attacker. Showing security of a protocol then amounts to showing that the attacker does *not* know a certain secret in the free algebra model.

However, when giving a proof that the axioms of a protocol formalization imply that the attacker does not know the secret, we implicitly have to consider *all* models of the axioms. In some of these models, equalities may hold that one would expect an implementation of the protocol to avoid: e.g., that the secret is equal to a publicly known value. To consider the free algebra model only, traditionally further axioms must be asserted: namely freeness of operations (to prevent unwanted *confusion* in the model), and a least fixed-point axiom for the knows predicate (to prevent the attacker from knowing messages without reason).

Stating these axioms explicitly seems inelegant. It introduces the risk of including too few distinctness axioms, impairing completeness of the verification approach by giving rise to spurious attacks (based on accidental equalities between different terms). Including too many axioms, on the other hand, might lead to an inconsistent protocol specification, thereby impairing soundness of the verification approach. We would like to avoid these dangers.

There is an alternative to asserting these axioms, and consequently proving security in the free algebra model. We will show that for many protocols, security can be established by exhibiting just *one* (arbitrary) model in which the attacker does not know the secret.

2.1 Model-Theoretic Foundations

Note that automated theorem provers implicitly consider every possible model satisfying the given axioms to see whether it satisfies the given conjecture, not only the quotients of the free algebra under the axioms. This means that in the models considered, additional properties not following from the given axioms may hold. In the case of cryptographic protocols, this may mean that a secret key coincides with a public value and therefore becomes known to the adversary. This is of course something which one would assume an implementation of the protocol to avoid, and therefore one would like to analyze the protocol under the assumption that this does not happen. There are two ways to deal with this situation:

1. If one wants to formulate the conjecture in a way that a proof of the conjecture means that the protocol is secure, one needs to explicitly include axioms which prevent an unwanted collapse of the model (for example, axioms requiring each constant in the model describing a secret value to be different from any other value).
2. Alternatively, one can formulate the conjecture in a negated way so that a proof of the conjecture corresponds to an attack, and the proven absence of a proof (equivalently, by soundness and completeness of FOL, a counterexample to the formula) corresponds to the security of the protocol. This

makes sure that, when considering a given protocol execution (i.e., a given instantiation of the message variables), all models of the formulas have to fulfill the attack conjecture in order for an attack to be detected, in particular also the quotient of the free algebra under the formula, which does not satisfy any equations that would be assumed not to hold in an implementation of the protocol (for example, between a secret key and a public value) and which we call a *confusion-free model* (to be defined precisely below). That way, false positives arising in this way can be avoided.

We would like to avoid having to introduce the distinctness axioms in the first option, since it would introduce the risk of including too few distinctness axioms (again giving rise to false counter-examples to the security of the specification), or too many (which may lead to an inconsistency of the formula on the whole).

The second option, however, raises an issue with respect to the completeness of the analysis: Note that in general it is *not* the case that any formula which holds in the quotient of a free algebra under given axioms also holds in every other model of the axioms. Therefore, in the second approach explained above, finding a counter-example which satisfies the axioms but not the conjecture does not in general have to mean that the confusion-free models of the axioms do not satisfy the conjecture either. This, however, is what one would like to establish here to show that the program is secure in a certain sense. Intuitively speaking, having a counter-example just means that there exists an implementation of the protocol which happens to be secure against the conjecture, maybe only because certain non-standard equalities happen to hold, whereas one would like to show that any reasonable implementation of the protocol, in particular the ones which correspond to the confusion-free models, is secure.

Therefore, we would like to establish under which conditions on the set of axioms and which conditions on the conjecture it is indeed the case that the following logical equivalence holds: There exists a model which satisfies the axioms but not the conjecture if and only if the confusion-free models satisfying the axioms do not satisfy the conjecture.

We consider the negated security conjecture, i.e., that the attacker knows the secret. A proof of this "insecurity conjecture" corresponds to an attack on the protocol. Since the proof implicitly considers *all* models of the protocol axioms, the attack cannot depend on accidental equalities; it will be possible in the free algebra model as well.

Recall that if a conjecture cannot be derived from a given set of axioms, this does not mean that one can derive the negation of the conjecture from the axioms. The conjecture may just be independent of the axioms, i.e., it may hold in some models satisfying the axioms, but not in others. In general it is not the case that any formula which holds in the free algebra model also holds in every other model of the axioms.

We now identify conditions on protocol formalizations that are sufficient to conclude security (i.e., that the attacker does not know the secret in the free algebra model) from the proven absence of a proof of the "insecurity conjecture" (or equivalently, by soundness and completeness of first-order logic, from a counterexample). Note that this is closely related to the treatment of negation as failure, which is justified in, e.g., Prolog by the "closed world assumption".[2]

Let us recall some concepts from mathematical logic. We observe that a formalization of security protocols in first-order logic (or more precisely, of the corresponding knows predicate describing the attacker's knowledge) can often be given by *strict Horn clauses.* The following definition is standard [9].

Definition 1 (Strict Horn Clause). *A* strict Horn clause *is a formula that consists of universal (first-order) quantifiers followed by a quantifier-free formula of the form φ (a* fact*), or $\varphi_1 \wedge \ldots \wedge \varphi_n \Longrightarrow \varphi$ (a* rule*), where the formulas φ_1, ..., φ_n, φ are all atomic.*

Theories consisting of strict Horn clauses always have an *initial model.* This is known as the *initial model theorem* [9].

Theorem 1 (Initial Model Theorem). *Let $\mathcal{T}$ be a theory consisting of strict Horn clauses. Then $\mathcal{T}$ has a model A with the property that for every model B of $\mathcal{T}$ there is a unique homomorphism from A to B. (Such a model A is called an* initial model *of $\mathcal{T}$. It is unique up to isomorphism.)*

The initial model satisfies the *no junk* and *no confusion* properties; its universe is indeed the freely generated term algebra of messages. Moreover, the initial model (also known as the *least Herbrand model*) satisfies only those atomic sentences that are derivable from $\mathcal{T}$. Thus, the initial model is precisely the free algebra model that we are interested in. Moreover, if we can find *any* model B in which the attacker does not know the secret, then the existence of a homomorphism from the initial model A to B implies that the attacker does not know the secret in the initial model either. (Recall that a homomorphism h from A to B, by definition, preserves satisfaction: if $A \models \mathsf{knows}(s)$, then $B \models \mathsf{knows}(h(s))$. Therefore, by contraposition, if $B \not\models \mathsf{knows}(h(s))$, then $A \not\models \mathsf{knows}(s)$.)

Theorem 1 also applies to theories that contain equality. (For a proof sketch, note that equality is a predicate that can be axiomatized by strict Horn clauses. We then obtain the initial model by taking equivalence classes of elements. A detailed proof can be found in [9].) Therefore the theorem covers both implicit and explicit decryption (cf. Sect. 1). More generally it covers *varieties*, i.e., algebraic structures defined by identities.

We would like to investigate now to what extent Thm. 1 can be strengthened to cover a larger fragment of FOL exceeding strict Horn clauses, in particular wrt. confusion-freeness in crypto-protocol verification.

For the purposes of this paper, we are only concerned with the additional knowledge that the adversary may gain from exploiting certain *equalities* in specific models of the axioms (which correspond to implementations of the protocol specification). One could also investigate other kind of relations besides equalities which the adversary may exploit to gain knowledge, although we believe

[2] It is interesting to note that there are non-standard interpretations of classical logic formulas such as inductive logic [7] that also enforce certain "closed world" assumptions and that have recently been used for crypto-protocol verification [8].

that in many cases these could be modeled by equalities, if necessary introducing extra function operators.

Therefore, we are interested in the models M of a given set $\mathcal{A}$ of axioms in the (logical) signature Σ (containing function and relation symbols) that are *confusion-free* in the following sense: for all terms $t1$, $t2$ in Σ with free variables $\boldsymbol{x}$, if the formula $\forall \boldsymbol{x}.\, t1 = t2$ holds in M, then it holds in all models of $\mathcal{A}$.

Suppose we are given a signature Σ and a set $\mathcal{A}$ of axioms and a conjecture c. We would like to establish sufficient conditions on $\mathcal{A}$ and c that imply existence of a set $\mathcal{G}$ of confusion-free models such that if all models in $\mathcal{G}$ satisfy c, then each model of $\mathcal{A}$ satisfies c.

Our approach is to find a way to construct all models of $\mathcal{A}$ out of the models in $\mathcal{G}$ in a way that preserves satisfaction of c. We recall a few definitions from algebraic model theory [10]. A *limit sentence* is a sentence of the form

$$(\forall x_1, \ldots, x_n)(\phi(x_1, \ldots, x_n) \Longrightarrow (\exists! y_1, \ldots, y_m)\psi(x_1, \ldots, x_n, y_1, \ldots, y_m)).$$

Note that, in particular, universal Horn sentences are limit sentences. However, limit sentences are strictly more expressive than universal Horn sentences: for example, one can show that the class of algebras satisfying the limit formula $\forall x.\, (\alpha(x) \wedge \beta(x) \Longrightarrow \exists! y.\, \rho(x, y))$ is not definable by a universal Horn theory [10, p.210].

A subclass $\mathcal{S}$ of a class $\mathcal{M}$ of models is called *reflective* if for every structure M in $\mathcal{M}$ there exists a structure S in $\mathcal{S}$ and a homomorphism $r : M \to S$ (the *reflection homomorphism*) such that for every structure S' in $\mathcal{S}$ and for every homomorphism $f : M \to S'$ there exists a unique homomorphism $f' : S \to S'$ such that $f = f' \circ r$.

Theorem 2. *Suppose that $\mathcal{A}$ is a set of limit sentences, c a universally quantified conjunction of atomic formulas, and $\mathcal{G}$ the set of reflections of the free Σ-structures on finitely many generators under the sentences in $\mathcal{A}$. Then the structures in $\mathcal{G}$ are confusion-free models such that if all models in $\mathcal{G}$ satisfy c, then each model of $\mathcal{A}$ satisfies c.*

Proof. Let T be a set of limit sentences and $\mathcal{A}$ the class of models of T. One can show that the class of $\mathcal{A}$-structures is closed under limits and directed colimits in the class of Σ-structures. By [10, 2.48], this implies that the class of $\mathcal{A}$-structures is reflective in the class of Σ-structures.[3] Therefore, the set $\mathcal{G}$ of reflections of the free Σ-structures on finitely many generators under the sentences in $\mathcal{A}$ is confusion-free. Note that every Σ-structure is a directed colimit of the free Σ-structures on finitely many generators. This implies that every $\mathcal{A}$-structure is a directed colimit of the structures in $\mathcal{G}$. Also, it follows that every $\mathcal{A}$-structure satisfies any universally quantified conjunction c of atomic formulas satisfied by all structures in $\mathcal{G}$ (because these are preserved by quotients and, as limit sentences, by directed colimits). This proves the above theorem.

[3] Note that it follows from [11] that the converse does not hold, that is a reflective subclass of a class of structures which is closed under colimits need not be definable by a limit theory.

Theorem 2 covers approaches such as [12] (formalizing BAN logic),[4] as well as [3,13] (that use similar formalizations as ours here). [13] also remarks that a significant number of protocols and security requirements can in fact be expressed using Horn formulas. [14] uses the (arguably inelegant) approach that the protocol specifier has to explicitly introduce relevant inequalities (such as different public keys being unequal). The paper does not comment on how to deal with a possible incompleteness or inconsistency of the resulting axioms. All of the above approaches do not explicitly consider the issue of soundness with respect to cryptographic inequalities raised above.

It would be worthwhile to consider whether Thm. 2 can be generalized from limit theories to basic theories defined in [10, 5.31], but we do not need this here.

Note that the restrictions on $\mathcal{A}$ cannot be relaxed arbitrarily. For instance, with $\mathcal{A} = \{e(k,a) \neq e(k,b) \implies \mathsf{knows}(s)\}$ and $c = \mathsf{knows}(s)$ one obtains a model which satisfies the axiom (by falsifying its premise) but not the conjecture, although the corresponding protocol should be regarded as insecure (because one would usually assume an implementation to satisfy $e(k,a) \neq e(k,b)$, unless $a = b$). The reason is that limit sentences do not admit negation.

In cryptographic protocols, the usage of negated equations in the protocol formalization applies for example to the error treatment when a cryptographic certificate verification fails. Certificate verification is typically performed by checking an equation $c = c'$, where c and c' are cryptographic expressions. For error handling one would need to specify a behavior that is executed when $c = c'$ fails to hold, i.e., when $c \neq c'$ holds.

To deal with the fact that this is not supported by limit sentences, we simply leave out the precondition $c \neq c'$ from the formalization. This abstraction is safe (because the attacker needs to do less work to achieve his goal), and it is not overly unrealistic, because one would usually assume that the attacker would anyhow be able to provoke a failed certificate check (simply by producing a wrong certificate) to try to exploit a possible security weakness in the error treatment. Nevertheless, we are currently considering whether one can make use of ideas on elimination of negation in term algebras in this context [15,16].

Also, the restrictions on c cannot be relaxed arbitrarily, for example because quotients of free algebras do not in general preserve inequalities and implications. Thus, for $\mathcal{A} = \emptyset$ and $c = (e(k,a) = e(k,b) \implies \mathsf{knows}(s))$ a counterexample is found, although one would usually assume an implementation to satisfy c (vacuously, since a and b are distinct because nothing forces them to be equal here).

However, in our practical examples it has so far always been possible to formalize the conjecture in a way that the restrictions do not become a problem. If they would, one could still take the approach of considering the security conjecture directly (without negating) and asserting suitable distinctness axioms. If

[4] Note that there a conjecture formalizes a security (rather than insecurity) property, but our theorem easily transforms to that case, since BAN logic does not formalize the adversary knowledge but rather the protocol participants assurance and can thus be treated in a dual way.

the model generator then finds counterexamples to both the conjecture and its negation, one knows that the conjecture is independent of the axioms; thus, more distinctness axioms should be introduced. However, this has not happened yet in our usage of this approach in a variety of examples and application case-studies.

2.2 Using Finite Model Generation for Verifying Crypto-Protocols

In Sect. 2.1, we have given sufficient conditions under which security of a protocol (i.e., that the attacker does not know the secret in the free algebra model) can be concluded from existence of just one (arbitrary) model in which the attacker does not know the secret. Unlike the free algebra model, which is necessarily infinite, this model may even be finite. Thus finite model generation [17] can be used to search for it.

A finite model generator is an automatic software tool that attempts to build a finite model of a (typically first-order) formula. In a sense, model generation is dual to theorem proving: while the latter establishes validity, the former establishes satisfiability (and may provide counterexamples by considering the negation of a formula).

Note that some formulas have infinite models only. Thus, failure to find a finite model does not prove unsatisfiability. Validity and finite satisfiability of first-order formulas are semi-decidable, general satisfiability however is not. In the following Sect. 3 we will see that these theoretical restrictions are of limited importance in practice.

3 Case Studies

To further illustrate the approach for protocol verification discussed in this paper, and to demonstrate its practicality, we have applied the approach to three well-known (and frequently studied) protocols: the RSA Probabilistic Signature Scheme (RSA-PSS) [18], the Wired Equivalent Privacy (WEP) protocol [19], and the Needham-Schroeder-Lowe (NSL) authentication protocol [20].

We present abstract protocol formalizations in TPTP [21] syntax. The TPTP library is a collection of standard benchmark problems for first-order theorem provers. The concrete syntax uses `&` for conjunction, `=>` for implication, and `!` (followed by a list of variables in square brackets) for universal quantification.

Vampire 10.0, an automatic theorem prover for first-order logic, was used to find attacks, and Paradox 2.3 [22], a finite model generator, was employed to search for models that show security. Both Vampire (which was invoked via Sutcliffe's "System on TPTP" [23] web interface) and Paradox support TPTP syntax as their input format.

We have also formalized these protocols in Isabelle/HOL [2], an interactive theorem prover and model generator for higher-order logic. Since the Isabelle/HOL formalization differs from the TPTP formalization only in terms of concrete syntax, we do not show it in this paper.

As is often done in protocol analysis, we assume *perfect cryptography* and consider abstract versions of these protocols only, i.e., we do not aim to verify an actual implementation, nor the mathematics underlying the cryptographic primitives (except where stated otherwise).

3.1 RSA-PSS

RSA-PSS [18] is a digital signature scheme that follows the usual "hash-then-sign" paradigm. RSA refers to the now classic algorithm for public-key cryptography devised by Rivest, Shamir, and Adleman [24]. PSS stands for "Probabilistic Signature Scheme", first described by Bellare and Rogaway [25]. Starting with a message m that is to be signed, the RSA-PSS protocol—at a very abstract level—proceeds in two steps:

1. Apply a one-way hash function to the message m to produce an encoded message $\mathsf{hash}(m)$.
2. Apply a signature function to the encoded message, using a private key k, to produce a signature $\mathsf{sign}(\mathsf{hash}(m), k)$.

The message m is then sent together with its signature, $\mathsf{sign}(\mathsf{hash}(m), k)$. The signature can be verified by the receiver using the sender's public key k^{-1}. Note that RSA-PSS is not an encryption algorithm; m becomes publicly known.

A detailed Isabelle/HOL formalization of the RSA-PSS protocol by Lindenberg and Wirt is available [26]. For our purposes, however, it will be sufficient to model hashing and signing as uninterpreted functions. Our analysis is therefore not specific to PSS hashing. A third function, conc, forms the concatenation of two messages.

We assume a naive implementation of the RSA signature function that suffers from an undesirable homomorphism property, which allows the attacker to compute the signature of concatenated messages from signatures for their components (and vice versa):

$$\mathsf{sign}(\mathsf{conc}(a, b), k) = \mathsf{conc}(\mathsf{sign}(a, k), \mathsf{sign}(b, k)). \tag{1}$$

If we consider a modified protocol without PSS hashing, then it is easy to show from (1) that the attacker can forge the signature for $\mathsf{conc}(b, a)$ if he knows the signature for $\mathsf{conc}(a, b)$. Vampire finds a proof of this result in less than a second.

Our goal is rather simple: we want to show that PSS hashing breaks this homomorphism property, thereby improving security of the signature scheme. We consider a protocol run where the message $\mathsf{conc}(a, b)$ is hashed, signed with some private key k, and then transmitted over an insecure connection. The first-order formulas that model the RSA-PSS protocol and the abilities of a potential Dolev-Yao attacker are shown in Fig. 1. Note that we do *not* assume sign to satisfy a homomorphism property similar to (1) wrt. hash.

Can we conclude from these axioms that the attacker knows the signature $\mathsf{sign}(\mathsf{hash}(\mathsf{conc}(b, a)), k)$? (In our formalization, it is certainly *possible* that the attacker knows this signature—since the knows predicate, as discussed in Sect. 1, may be true everywhere—but is it also *necessary*?) The answer is: No. Paradox finds a counterexample with just four elements (shown in Fig. 2) in about two

```
fof(knows_hash, axiom,(![X]:(knows(X)=>knows(hash(X))))).
fof(knows_sign,axiom,(![X,K]:((knows(X)&knows(K))=>knows(sign(X,K))))).
fof(remove_sign,axiom,(![X,K]:((knows(sign(X,K))&knows(invs(K)))=>knows(X)))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
fof(sign_hom,axiom,(![X,Y,K]:(sign(conc(X,Y),K)=conc(sign(X,K),sign(Y,K))))).
fof(protocol_msg,axiom,(knows(conc(conc(a,b),sign(hash(conc(a,b)),k))))).
fof(public_key,axiom,(knows(invs(k)))).
fof(attack,conjecture,(knows(sign(hash(conc(b,a)),k)))).
```

Fig. 1. TPTP encoding of the RSA-PSS protocol

$\mathsf{hash}(1)$	$= 2$	$\mathsf{hash}(2)$	$= 4$	$\mathsf{hash}(3)$	$= 3$	$\mathsf{hash}(4)$	$= 4$
$\mathsf{invs}(1)$	$= 3$	$\mathsf{invs}(2)$	$= 3$	$\mathsf{invs}(3)$	$= 1$	$\mathsf{invs}(4)$	$= 3$
$\mathsf{sign}(1,1)$	$= 2$	$\mathsf{sign}(1,2)$	$= 2$	$\mathsf{sign}(1,3)$	$= 1$	$\mathsf{sign}(1,4)$	$= 2$
$\mathsf{sign}(2,1)$	$= 2$	$\mathsf{sign}(2,2)$	$= 2$	$\mathsf{sign}(2,3)$	$= 3$	$\mathsf{sign}(2,4)$	$= 1$
$\mathsf{sign}(3,1)$	$= 1$	$\mathsf{sign}(3,2)$	$= 1$	$\mathsf{sign}(3,3)$	$= 3$	$\mathsf{sign}(3,4)$	$= 2$
$\mathsf{sign}(4,1)$	$= 2$	$\mathsf{sign}(4,2)$	$= 2$	$\mathsf{sign}(4,3)$	$= 4$	$\mathsf{sign}(4,4)$	$= 1$
$\mathsf{conc}(1,1)$	$= 1$	$\mathsf{conc}(1,2)$	$= 2$	$\mathsf{conc}(1,3)$	$= 3$	$\mathsf{conc}(1,4)$	$= 1$
$\mathsf{conc}(2,1)$	$= 2$	$\mathsf{conc}(2,2)$	$= 2$	$\mathsf{conc}(2,3)$	$= 3$	$\mathsf{conc}(2,4)$	$= 2$
$\mathsf{conc}(3,1)$	$= 3$	$\mathsf{conc}(3,2)$	$= 3$	$\mathsf{conc}(3,3)$	$= 3$	$\mathsf{conc}(3,4)$	$= 3$
$\mathsf{conc}(4,1)$	$= 4$	$\mathsf{conc}(4,2)$	$= 2$	$\mathsf{conc}(4,3)$	$= 3$	$\mathsf{conc}(4,4)$	$= 1$
a	$= 4$	b	$= 1$	k	$= 3$		
$\mathsf{knows}(1)$		$\mathsf{knows}(2)$		$\neg\mathsf{knows}(3)$		$\mathsf{knows}(4)$	

Fig. 2. Model showing security of RSA-PSS hashing

seconds on a current personal computer. This proves that the attack mentioned above (exploiting (1)) is no longer possible with hashing.[5] Of course the interpretation of, e.g., conc is not the usual one in this finite model. By Thm. 1, however, the result also holds in the free algebra model (modulo (1)).

3.2 Wired Equivalent Privacy

The Wired Equivalent Privacy (WEP) protocol [19] was introduced in 1997 to provide confidentiality for wireless networks. Later serious weaknesses were identified, and the protocol is now considered deprecated. In the context of this paper the protocol is interesting because it employs the "exclusive or" function, which can be characterized algebraically by associativity, commutativity, existence of a neutral element, and nilpotency:

$$\begin{aligned} &\mathsf{xor}(\mathsf{xor}(x,y),z) = \mathsf{xor}(x,\mathsf{xor}(y,z)), \qquad && \mathsf{xor}(x,0) = x, \\ &\mathsf{xor}(x,y) = \mathsf{xor}(y,x), && \mathsf{xor}(x,x) = 0. \end{aligned}$$

Note that all four axioms are universally quantified identities.

[5] We have also shown that this remains true even if the hashing function is reversible, i.e., if we add an axiom `![X]:(knows(hash(X))=>knows(X))`.

```
fof(xor_assoc,axiom,(![X,Y,Z]:(xor(xor(X,Y),Z)=xor(X,xor(Y,Z))))).
fof(xor_comm,axiom,(![X,Y]:(xor(X,Y)=xor(Y,X)))).
fof(xor_neutral,axiom,(![X]:(xor(X,zero)=X))).
fof(xor_nilpotent,axiom,(![X]:(xor(X,X)=zero))).
fof(knows_xor,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(xor(X,Y))))).
fof(knows_zero,axiom,(knows(zero))).
fof(knows_rc4,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(rc4(X,Y))))).
fof(knows_c,axiom,(![X,Y]:(knows(X)=>knows(c(X))))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
fof(protocol_msg,axiom,(knows(conc(v,xor(conc(m,c(m)),rc4(v,k)))))).
% an arbitrary delta message
fof(knows_d,axiom,(knows(d))).
% homomorphism property for c
fof(c_hom,axiom,(![X,Y]:(c(xor(X,Y))=xor(c(X),c(Y))))).
% homomorphism property for conc
fof(conc_hom,axiom,(![X1,Y1,X2,Y2]:(xor(conc(X1,Y1),conc(X2,Y2))
  =conc(xor(X1,X2),xor(Y1,Y2))))).
% the protocol is malleable
fof(attack,conjecture,
  (knows(conc(v,xor(conc(xor(m,d),c(xor(m,d))),rc4(v,k)))))).
```

Fig. 3. TPTP encoding of the Wired Equivalent Privacy protocol

Our formalization is based on [27, Sect. 3.3]. The WEP protocol uses the RC4 algorithm, which generates a sequence of pseudo-random bits from an initial vector v and a shared secret key k. Moreover, it uses a checksum algorithm c, e.g., cyclic redundancy check. We model both RC4 and c by uninterpreted functions.

To encrypt a message m, principal A chooses an initial vector v, computes $\mathsf{RC4}(v, k)$ and $\mathsf{c}(m)$, encrypts $\mathsf{conc}(m, \mathsf{c}(m))$ with $\mathsf{RC4}(v, k)$ (using xor), and sends both v and the ciphertext to principal B. To decrypt, B (who must know the shared key k) then computes $\mathsf{RC4}(v, k)$, obtains $\mathsf{conc}(m, \mathsf{c}(m))$ from the ciphertext (again using xor), and verifies the checksum $\mathsf{c}(m)$.

If both conc and c satisfy a homomorphism property with respect to xor (see Fig. 3), the protocol becomes malleable: a Dolev-Yao attacker can modify a ciphertext arbitrarily without disrupting the checksum [28]. Vampire automatically proves the attack in about three seconds.

The attack is no longer possible if we drop the `conc_hom` axiom. Paradox then finds a counterexample (of size 8) to the `attack` conjecture in less than a second. We omit the model for space reasons.

If we instead drop the `c_hom` axiom, Paradox—perhaps surprisingly—fails to refute the attack. We used it to exhaustively check all models of size 15 or less (which took several hours on a current personal computer); none of them constitute a counterexample to the conjecture. Possibly, all counterexamples are infinite.

3.3 Needham-Schroeder-Lowe with ECB

The Needham-Schroeder-Lowe (NSL) protocol [20] is perhaps the most frequently studied authentication protocol of all. Our formalization is based on [27, Sect. 3.5]. The NSL protocol consists of three messages.

1. First, principal A sends $\mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(B))$ to principal B, where $\mathsf{pk}(B)$ is B's public key, and N_A is a fresh (e.g., random) value.
2. B decrypts this message, using his secret key $\mathsf{sk}(B)$, to learn N_A. He then sends $\mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, B)), \mathsf{pk}(A))$ to A, where N_B is again a fresh value.
3. Third, A decrypts B's message to learn N_B, and replies to B with $\mathsf{encrypt}(N_B, \mathsf{pk}(B))$. B verifies receipt of this message.

At the end of the protocol, A and B are convinced to talk with each other, and to share the secrets N_A and N_B. The protocol, however, is flawed if Electronic Code Book (ECB) is used for encryption. ECB is a block cipher mode that simply encrypts each message block separately. ECB (formalized by the uninterpreted function encrypt below) renders the NSL protocol insecure because it satisfies a homomorphism property:

$$\mathsf{encrypt}(\mathsf{conc}(a, b), k) = \mathsf{conc}(\mathsf{encrypt}(a, k), \mathsf{encrypt}(b, k)). \tag{2}$$

The attack works as follows. Assuming that principal A initiates a protocol session S_1 with the intruder by sending $\mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(I))$, the intruder can then

1. impersonate A to initiate a session S_2 with principal B:

$$(S_2.1) \quad \mathrm{I(A)} \rightarrow \mathrm{B}: \quad \mathsf{encrypt}(\mathsf{conc}(N_A, A), \mathsf{pk}(B))\,,$$

2. learn the secret N_B by abusing A to decrypt B's response:

$$\begin{array}{lll} (S_2.2) & \mathrm{B} \rightarrow \mathrm{I(A)}: & \mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, B)), \mathsf{pk}(A)) \\ (S_1.2) & \mathrm{I} \rightarrow \ \mathrm{A}\ : & \mathsf{encrypt}(\mathsf{conc}(N_A, \mathsf{conc}(N_B, I)), \mathsf{pk}(A)) \\ (S_1.3) & \mathrm{A} \rightarrow \ \mathrm{I}\ : & \mathsf{encrypt}(N_B, \mathsf{pk}(I)), \end{array}$$

3. and finally use N_B to trick B (who is actually communicating with the intruder) into believing that he is communicating with A:

$$(S_2.3) \quad \mathrm{I(A)} \rightarrow \mathrm{B}: \quad \mathsf{encrypt}(N_B, \mathsf{pk}(B))\,.$$

The ECB homomorphism property (2) is used in step 2 of the attack.

The protocol formalization in TPTP syntax is shown in Fig. 4. We assume that exactly one principal, A, initiates a session with the intruder. To model freshness of N_B, we use a Skolem function (rather than a constant) nb that takes three arguments, which correspond to the three parameters in the protocol's first message, i.e., N_A, A, and B. Vampire automatically proves the attack in about two seconds.

```
fof(knows_encrypt,axiom,(![X,K]:((knows(X)&knows(K))=>knows(encrypt(X,K))))).
fof(remove_encrypt,axiom,(![X,Y]:((knows(encrypt(X,pk(Y)))&knows(sk(Y)))
  =>knows(X)))).
fof(knows_conc,axiom,(![X,Y]:((knows(X)&knows(Y))=>knows(conc(X,Y))))).
fof(remove_conc,axiom,(![X,Y]:(knows(conc(X,Y))=>(knows(X)&knows(Y))))).
% principal A initiates a session with the intruder I
fof(protocol_msg_1,axiom,(knows(encrypt(conc(na,a),pk(i))))).
fof(protocol_msg_2,axiom,(![Na,A,B]:(knows(encrypt(conc(Na,A),pk(B)))
  =>knows(encrypt(conc(Na,conc(nb(Na,A,B),B)),pk(A)))))).
% principal A will respond to the intruder's reply
fof(protocol_msg_3,axiom,(![Nb]:(knows(encrypt(conc(na,conc(Nb,i)),pk(a)))
  =>knows(encrypt(Nb,pk(i)))))).
fof(knows_i,axiom,(knows(i))).
fof(knows_pka,axiom,(knows(pk(a)))).
fof(knows_pkb,axiom,(knows(pk(b)))).
fof(knows_pki,axiom,(knows(pk(i)))).
fof(knows_ski,axiom,(knows(sk(i)))).
% ECB homomorphism property
fof(ecb_hom,axiom,(![X,Y,K]:(encrypt(conc(X,Y),K)
  =conc(encrypt(X,K),encrypt(Y,K))))).
fof(attack,conjecture,(knows(nb(na,a,b)))).
```

Fig. 4. TPTP encoding of the Needham-Schroeder-Lowe protocol

On the other hand, if we do not assume (2), the attacker can no longer gain knowledge of $\mathsf{nb}(N_A, A, B)$. Paradox finds a counterexample of size 4 in just about a second. We omit the model for space reasons.

Our formalization does not model the state of principal A. In reality, the intruder can abuse A to decrypt a value only once. In our formal model, the intruder is more powerful: there appears to be an unbounded number of protocol sessions initiated by A available to him. However, this is not used in Vampire's proof of the attack, and it does not affect security of the protocol when (2) is omitted.

Infinitely Many Principals

We can model a potentially infinite number of principals by introducing a unary function next and a predicate $\mathsf{principal}$, and requiring $\mathsf{principal}(\mathsf{i})$ as well as $\forall x. (\mathsf{principal}(x) \implies \mathsf{principal}(\mathsf{next}(x)))$. Note that these assertions are strict Horn clauses. If we modify the NSL formalization accordingly (by replacing `protocol_msg_1`, `protocol_msg_3`, and `knows_pkX` from Fig. 4 with suitably quantified versions), Paradox still finds a counterexample of size 4. Thus, the NSL protocol is (by Thm. 1) secure even in the presence of infinitely many principals.

4 Conclusion

In this paper, we have proposed an approach to crypto-protocol verification that proceeds by negating the security conjecture, and then employs finite model generation to show security by means of a single (counter-) model. This idea was independently pioneered in an earlier paper by Selinger [29]. Here we have established conditions under which the approach is sound, thereby perhaps remedying the "uneasy feeling" that Selinger in that paper mentioned that he had because of the models' potential simplicity. Our result also shows that approaches such as [3] are sound in the sense that our Theorem 2 applies to it. We have also shown that the search for models can be automated not just "in principle", but that current model generators are extremely valuable for this task.

Theorem proving, in this approach, can be used to find attacks. Unlike in the traditional free algebra approach, no freeness axioms or least fixed-point axioms for the attacker's knowledge are required.

In contrast to ProVerif, we do not merely claim security of protocols, but produce "security certificates": models that can be verified independently. This has recently been explored further by Goubault-Larrecq [30], who translates models into Coq proofs of protocol correctness. Furthermore, our use of a standard, highly efficient theorem prover and model generator leads to effortless support for a larger fragment of first-order logic, including, e.g., equality.

The proposed technique has both theoretical and practical limitations. It is sound for protocols that can be formalized by strict Horn clauses (including identities), and more generally, limit theories (cf. Sect. 2). On the practical side, finite model generators may fail to find a counterexample (which would demonstrate security of the protocol) because of resource (i.e., runtime, memory) constraints, or simply because no finite model exists. Therefore the approach is not complete; it may fail to demonstrate the security of a secure protocol. Nevertheless, its successful application to three well-known protocols in this paper provides evidence that the approach is both sufficiently versatile and practical. It can simplify protocol formalizations and security proofs significantly.

In this paper, we only considered secrecy properties. However, the general results carry over to other properties that can be formalized within limit theories (such as authentication by correspondence properties).

In future work, it would be interesting to investigate how the ideas discussed here could be used in the context of verifying implementations, rather than specifications of crypto protocols, e.g., in the context of [31], or the verification of secure information flow, e.g., based on [32].

References

1. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
2. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
3. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS, vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
4. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: CSFW-14, pp. 82–96. IEEE Computer Society, Los Alamitos (2001)

5. Jürjens, J.: A domain-specific language for cryptographic protocols based on streams. Journal of Logic and Algebraic Programming (JLAP) (2009)
6. Blanchet, B.: From secrecy to authenticity in security protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 342–359. Springer, Heidelberg (2002)
7. Comon, H., Nieuwenhuis, R.: Induction = I-axiomatization + first-order consistency. Technical report, ENS Cachan (1998)
8. Steel, G., Bundy, A., Maidl, M.: Attacking a protocol for group key agreement by refuting incorrect inductive conjectures. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 137–151. Springer, Heidelberg (2004)
9. Hodges, W.: Model Theory. Cambridge University Press, Cambridge (1993)
10. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. London Math. Soc. Lect. Note Ser., vol. 189. Cambridge University Press, Cambridge (1994)
11. Jürjens, J.: On a problem of Gabriel and Ulmer. Journal of Pure and Applied Algebra 158, 183–196 (2001)
12. Schumann, J.: Automatic verification of cryptographic protocols with SETHEO. In: CADE (1999)
13. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. Sci. Comput. Program. 50(1-3), 51–71 (2004)
14. Cohen, E.: First-order verification of cryptographic protocols. Journal of Computer Security 11(2), 189–216 (2003)
15. Lassez, J.L., Maher, M.J., Marriott, K.: Elimination of negation in term algebras. In: Tarlecki, A. (ed.) MFCS 1991. LNCS, vol. 520, pp. 1–16. Springer, Heidelberg (1991)
16. Comon, H., Fernández, M.: Negation elimination in equational formulae. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629. Springer, Heidelberg (1992)
17. Weber, T.: SAT-based Finite Model Generation for Higher-Order Logic. PhD thesis, Technische Universität München (2008)
18. RSA Laboratories: PKCS #1: RSA Cryptography Standard Version 2.1. (June 2002)
19. Institute of Electrical and Electronics Engineers: IEEE Std 802.11-1997 (1997)
20. Lowe, G.: An attack on the Needham-Schroeder public key authentication protocol. Information Processing Letters 56(3), 131–136 (1995)
21. Sutcliffe, G., Suttner, C.B.: The TPTP problem library: CNF release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)
22. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: CADE, Workshop W4 (2003)
23. Sutcliffe, G.: System on TPTP, `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP` (accessed January 9, 2009)
24. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
25. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
26. Lindenberg, C., Wirt, K.: SHA1, RSA, PSS and more. In: Klein, G., Nipkow, T., Paulson, L. (eds.) The Archive of Formal Proofs (May 2005), `http://afp.sourceforge.net/entries/RSAPSS.shtml`, Formal proof development

27. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
28. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: The insecurity of 802.11. In: MOBICOM, pp. 180–188 (2001)
29. Selinger, P.: Models for an adversary-centric protocol logic. Electr. Notes Theor. Comput. Sci. 55(1) (2001)
30. Goubault-Larrecq, J.: Towards producing formally checkable security proofs, automatically. In: Computer Security Foundations (CSF), pp. 224–238 (2008)
31. Jürjens, J.: Security analysis of crypto-based java programs using automated theorem provers. In: ASE, pp. 167–176. IEEE Computer Society, Los Alamitos (2006)
32. Jürjens, J.: Secure information flow for concurrent processes. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 395–409. Springer, Heidelberg (2000)

A Appendix

This appendix contains finite models that demonstrate security of the Needham-Schroeder-Lowe protocol (in Fig. 5) and of the Wireless Equivalent Privacy protocol (in Fig. 6). These models were mentioned in Sect. 3.3 and Sect. 3.2, respectively, of the paper. They were omitted from the paper for space reasons.

$\mathsf{conc}(1,1)$	$=1$	$\mathsf{conc}(1,2)$	$=2$	$\mathsf{conc}(1,3)$	$=1$	$\mathsf{conc}(1,4)$	$=2$
$\mathsf{conc}(2,1)$	$=2$	$\mathsf{conc}(2,2)$	$=2$	$\mathsf{conc}(2,3)$	$=2$	$\mathsf{conc}(2,4)$	$=2$
$\mathsf{conc}(3,1)$	$=3$	$\mathsf{conc}(3,2)$	$=2$	$\mathsf{conc}(3,3)$	$=1$	$\mathsf{conc}(3,4)$	$=4$
$\mathsf{conc}(4,1)$	$=4$	$\mathsf{conc}(4,2)$	$=2$	$\mathsf{conc}(4,3)$	$=2$	$\mathsf{conc}(4,4)$	$=4$
$\mathsf{encrypt}(1,1)$	$=3$	$\mathsf{encrypt}(1,2)$	$=4$	$\mathsf{encrypt}(1,3)$	$=1$	$\mathsf{encrypt}(1,4)$	$=4$
$\mathsf{encrypt}(2,1)$	$=4$	$\mathsf{encrypt}(2,2)$	$=4$	$\mathsf{encrypt}(2,3)$	$=2$	$\mathsf{encrypt}(2,4)$	$=4$
$\mathsf{encrypt}(3,1)$	$=3$	$\mathsf{encrypt}(3,2)$	$=1$	$\mathsf{encrypt}(3,3)$	$=3$	$\mathsf{encrypt}(3,4)$	$=4$
$\mathsf{encrypt}(4,1)$	$=4$	$\mathsf{encrypt}(4,2)$	$=3$	$\mathsf{encrypt}(4,3)$	$=3$	$\mathsf{encrypt}(4,4)$	$=4$
$\mathsf{nb}(1,1,1)$	$=1$	$\mathsf{nb}(1,1,2)$	$=3$	$\mathsf{nb}(1,1,3)$	$=3$	$\mathsf{nb}(1,1,4)$	$=1$
$\mathsf{nb}(1,2,1)$	$=1$	$\mathsf{nb}(1,2,2)$	$=1$	$\mathsf{nb}(1,2,3)$	$=2$	$\mathsf{nb}(1,2,4)$	$=1$
$\mathsf{nb}(1,3,1)$	$=3$	$\mathsf{nb}(1,3,2)$	$=1$	$\mathsf{nb}(1,3,3)$	$=3$	$\mathsf{nb}(1,3,4)$	$=1$
$\mathsf{nb}(1,4,1)$	$=1$	$\mathsf{nb}(1,4,2)$	$=3$	$\mathsf{nb}(1,4,3)$	$=4$	$\mathsf{nb}(1,4,4)$	$=2$
$\mathsf{nb}(2,1,1)$	$=1$	$\mathsf{nb}(2,1,2)$	$=4$	$\mathsf{nb}(2,1,3)$	$=1$	$\mathsf{nb}(2,1,4)$	$=1$
$\mathsf{nb}(2,2,1)$	$=4$	$\mathsf{nb}(2,2,2)$	$=2$	$\mathsf{nb}(2,2,3)$	$=4$	$\mathsf{nb}(2,2,4)$	$=2$
$\mathsf{nb}(2,3,1)$	$=1$	$\mathsf{nb}(2,3,2)$	$=4$	$\mathsf{nb}(2,3,3)$	$=2$	$\mathsf{nb}(2,3,4)$	$=3$
$\mathsf{nb}(2,4,1)$	$=1$	$\mathsf{nb}(2,4,2)$	$=2$	$\mathsf{nb}(2,4,3)$	$=1$	$\mathsf{nb}(2,4,4)$	$=2$
$\mathsf{nb}(3,1,1)$	$=4$	$\mathsf{nb}(3,1,2)$	$=1$	$\mathsf{nb}(3,1,3)$	$=1$	$\mathsf{nb}(3,1,4)$	$=3$
$\mathsf{nb}(3,2,1)$	$=3$	$\mathsf{nb}(3,2,2)$	$=1$	$\mathsf{nb}(3,2,3)$	$=4$	$\mathsf{nb}(3,2,4)$	$=3$
$\mathsf{nb}(3,3,1)$	$=1$	$\mathsf{nb}(3,3,2)$	$=4$	$\mathsf{nb}(3,3,3)$	$=1$	$\mathsf{nb}(3,3,4)$	$=1$
$\mathsf{nb}(3,4,1)$	$=1$	$\mathsf{nb}(3,4,2)$	$=1$	$\mathsf{nb}(3,4,3)$	$=4$	$\mathsf{nb}(3,4,4)$	$=4$
$\mathsf{nb}(4,1,1)$	$=4$	$\mathsf{nb}(4,1,2)$	$=4$	$\mathsf{nb}(4,1,3)$	$=2$	$\mathsf{nb}(4,1,4)$	$=3$
$\mathsf{nb}(4,2,1)$	$=4$	$\mathsf{nb}(4,2,2)$	$=1$	$\mathsf{nb}(4,2,3)$	$=2$	$\mathsf{nb}(4,2,4)$	$=1$
$\mathsf{nb}(4,3,1)$	$=2$	$\mathsf{nb}(4,3,2)$	$=2$	$\mathsf{nb}(4,3,3)$	$=2$	$\mathsf{nb}(4,3,4)$	$=1$
$\mathsf{nb}(4,4,1)$	$=4$	$\mathsf{nb}(4,4,2)$	$=1$	$\mathsf{nb}(4,4,3)$	$=3$	$\mathsf{nb}(4,4,4)$	$=4$
$\mathsf{pk}(1)$	$=3$	$\mathsf{pk}(2)$	$=4$	$\mathsf{pk}(3)$	$=1$	$\mathsf{pk}(4)$	$=2$
$\mathsf{sk}(1)$	$=2$	$\mathsf{sk}(2)$	$=4$	$\mathsf{sk}(3)$	$=1$	$\mathsf{sk}(4)$	$=4$
a	$=1$	b	$=1$	i	$=3$	N_A	$=3$
$\mathsf{knows}(1)$		$\neg\mathsf{knows}(2)$		$\mathsf{knows}(3)$		$\neg\mathsf{knows}(4)$	

Fig. 5. Model showing security of NSL

xor(1, 1)	= 5	xor(1, 2)	= 7	xor(1, 3)	= 4	xor(1, 4)	= 3
xor(1, 5)	= 1	xor(1, 6)	= 8	xor(1, 7)	= 2	xor(1, 8)	= 6
xor(2, 1)	= 7	xor(2, 2)	= 5	xor(2, 3)	= 8	xor(2, 4)	= 6
xor(2, 5)	= 2	xor(2, 6)	= 4	xor(2, 7)	= 1	xor(2, 8)	= 3
xor(3, 1)	= 4	xor(3, 2)	= 8	xor(3, 3)	= 5	xor(3, 4)	= 1
xor(3, 5)	= 3	xor(3, 6)	= 7	xor(3, 7)	= 6	xor(3, 8)	= 2
xor(4, 1)	= 3	xor(4, 2)	= 6	xor(4, 3)	= 1	xor(4, 4)	= 5
xor(4, 5)	= 4	xor(4, 6)	= 2	xor(4, 7)	= 8	xor(4, 8)	= 7
xor(5, 1)	= 1	xor(5, 2)	= 2	xor(5, 3)	= 3	xor(5, 4)	= 4
xor(5, 5)	= 5	xor(5, 6)	= 6	xor(5, 7)	= 7	xor(5, 8)	= 8
xor(6, 1)	= 8	xor(6, 2)	= 4	xor(6, 3)	= 7	xor(6, 4)	= 2
xor(6, 5)	= 6	xor(6, 6)	= 5	xor(6, 7)	= 3	xor(6, 8)	= 1
xor(7, 1)	= 2	xor(7, 2)	= 1	xor(7, 3)	= 6	xor(7, 4)	= 8
xor(7, 5)	= 7	xor(7, 6)	= 3	xor(7, 7)	= 5	xor(7, 8)	= 4
xor(8, 1)	= 6	xor(8, 2)	= 3	xor(8, 3)	= 2	xor(8, 4)	= 7
xor(8, 5)	= 8	xor(8, 6)	= 1	xor(8, 7)	= 4	xor(8, 8)	= 5
zero	= 5						
RC4(1, 1)	= 5	RC4(1, 2)	= 8	RC4(1, 3)	= 2	RC4(1, 4)	= 5
RC4(1, 5)	= 1	RC4(1, 6)	= 1	RC4(1, 7)	= 2	RC4(1, 8)	= 7
RC4(2, 1)	= 5	RC4(2, 2)	= 2	RC4(2, 3)	= 3	RC4(2, 4)	= 2
RC4(2, 5)	= 2	RC4(2, 6)	= 5	RC4(2, 7)	= 1	RC4(2, 8)	= 7
RC4(3, 1)	= 5	RC4(3, 2)	= 1	RC4(3, 3)	= 1	RC4(3, 4)	= 1
RC4(3, 5)	= 5	RC4(3, 6)	= 1	RC4(3, 7)	= 5	RC4(3, 8)	= 7
RC4(4, 1)	= 1	RC4(4, 2)	= 2	RC4(4, 3)	= 2	RC4(4, 4)	= 3
RC4(4, 5)	= 5	RC4(4, 6)	= 5	RC4(4, 7)	= 1	RC4(4, 8)	= 7
RC4(5, 1)	= 5	RC4(5, 2)	= 4	RC4(5, 3)	= 1	RC4(5, 4)	= 5
RC4(5, 5)	= 1	RC4(5, 6)	= 4	RC4(5, 7)	= 1	RC4(5, 8)	= 7
RC4(6, 1)	= 5	RC4(6, 2)	= 5	RC4(6, 3)	= 2	RC4(6, 4)	= 4
RC4(6, 5)	= 5	RC4(6, 6)	= 7	RC4(6, 7)	= 1	RC4(6, 8)	= 7
RC4(7, 1)	= 6	RC4(7, 2)	= 1	RC4(7, 3)	= 1	RC4(7, 4)	= 2
RC4(7, 5)	= 1	RC4(7, 6)	= 7	RC4(7, 7)	= 6	RC4(7, 8)	= 7
RC4(8, 1)	= 7	RC4(8, 2)	= 7	RC4(8, 3)	= 7	RC4(8, 4)	= 7
RC4(8, 5)	= 7	RC4(8, 6)	= 7	RC4(8, 7)	= 7	RC4(8, 8)	= 7
c(1)	= 5	c(2)	= 2	c(3)	= 1	c(4)	= 1
c(5)	= 5	c(6)	= 7	c(7)	= 2	c(8)	= 7
conc(1, 1)	= 1	conc(1, 2)	= 2	conc(1, 3)	= 7	conc(1, 4)	= 8
conc(1, 5)	= 1	conc(1, 6)	= 3	conc(1, 7)	= 2	conc(1, 8)	= 8
conc(2, 1)	= 3	conc(2, 2)	= 8	conc(2, 3)	= 7	conc(2, 4)	= 7
conc(2, 5)	= 4	conc(2, 6)	= 3	conc(2, 7)	= 4	conc(2, 8)	= 2
conc(3, 1)	= 2	conc(3, 2)	= 8	conc(3, 3)	= 7	conc(3, 4)	= 8
conc(3, 5)	= 2	conc(3, 6)	= 4	conc(3, 7)	= 2	conc(3, 8)	= 2
conc(4, 1)	= 8	conc(4, 2)	= 7	conc(4, 3)	= 7	conc(4, 4)	= 7
conc(4, 5)	= 2	conc(4, 6)	= 3	conc(4, 7)	= 4	conc(4, 8)	= 2
conc(5, 1)	= 5	conc(5, 2)	= 8	conc(5, 3)	= 6	conc(5, 4)	= 2
conc(5, 5)	= 5	conc(5, 6)	= 2	conc(5, 7)	= 2	conc(5, 8)	= 7
conc(6, 1)	= 8	conc(6, 2)	= 2	conc(6, 3)	= 4	conc(6, 4)	= 2
conc(6, 5)	= 3	conc(6, 6)	= 4	conc(6, 7)	= 6	conc(6, 8)	= 7
conc(7, 1)	= 6	conc(7, 2)	= 7	conc(7, 3)	= 3	conc(7, 4)	= 6
conc(7, 5)	= 2	conc(7, 6)	= 2	conc(7, 7)	= 3	conc(7, 8)	= 7
conc(8, 1)	= 7	conc(8, 2)	= 7	conc(8, 3)	= 7	conc(8, 4)	= 7
conc(8, 5)	= 7	conc(8, 6)	= 7	conc(8, 7)	= 7	conc(8, 8)	= 7
m	= 2	v	= 1	k	= 2	d	= 1
knows(1)		¬knows(2)		¬knows(3)		¬knows(4)	
knows(5)		¬knows(6)		¬knows(7)		¬knows(8)	

Fig. 6. Model showing security of WEP

Towards a Type System for Security APIs

Gavin Keighren[1], David Aspinall[1], and Graham Steel[2]

[1] Laboratory for Foundations of Computer Science
School of Informatics, The University of Edinburgh
Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, UK
gavin.keighren@ed.ac.uk, david.aspinall@ed.ac.uk
[2] LSV, INRIA & CNRS & ENS de Cachan
61, avenue du Président Wilson
94235 CACHAN Cedex – France
graham.steel@lsv.ens-cachan.fr

Abstract. Security API analysis typically only considers a subset of an API's functions, with results bounded by the number of function calls. Furthermore, attacks involving partial leakage of sensitive information are usually not covered.

Type-based static analysis has the potential to alleviate these shortcomings. To that end, we present a type system for secure information flow based upon the one of Volpano, Smith and Irvine [1], extended with types for cryptographic keys and ciphertext similar to those in Sumii and Pierce [2]. In contrast to some other type systems, the encryption and decryption of keys does not require special treatment.

We show that a well-typed sequence of commands is non-interferent, based upon a definition of indistinguishability where, in certain circumstances, the adversary can distinguish between ciphertexts that correspond to encrypted public data.

1 Introduction

It is common for computer systems which store, process and manipulate sensitive data to use a dedicated security hardware device (e.g., IBM 4758 [3] and nCipher nShield [4]). The set of functions provided by a security device is termed its *security API*, as they are intended to enforce a security policy as well as provide an interface to the device. A security policy describes the restrictions on the access to, use of, and propagation of data in the system. These restrictions, therefore, must follow as a direct consequence of the API functions which are available to users of the security device.

The analysis of security APIs has traditionally been carried out by enumerating the set of data items which the adversary (a malicious user) can obtain through repeated interactions with the API. While this approach has had reasonable success (e.g., [5,6,7,8,9]), results are typically bounded by the number of API calls, do not consider data integrity, and only detect flaws which involve the release of sensitive data items in their entirety.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 173–192, 2009.

In contrast, static analysis has the potential to provide unbounded results, identify flaws which allow for sensitive data to be leaked via covert control-flow channels, and also deal with data integrity. The type system presented in this paper is the foundation of one such approach, although it does not yet deal with integrity.

Our work builds upon the information-flow analysis capabilities of Volpano, Smith and Irvine's type system [1] by including cryptographic types similar to those from Sumii and Pierce's system for analysing security protocols [2]. Although there are many similarities between security APIs and security protocols, analysis methods for the latter are typically designed to deal with fixed-length specified interactions, and therefore generally do not scale well when applied to arbitrary sequences of interactions.

2 Background

Hardware Security Modules (HSMs) comprise some memory and a processor inside a tamper-proof enclosure which prevents the memory contents from being physically read — any breach causes the memory to be erased within a few micro-seconds. Additional storage is provided by the host system to which the HSM is attached. This leads to a natural partition of memory locations: those inside the HSM are high security, and those on the host system are low security.

Memory locations on the host system are deemed low security, since the attack model for security API analysis assumes that the adversary has full control of the host system. In addition, the adversary is assumed to be capable of calling certain functions provided by the HSM's security API (because, for example, they have hijacked a user's session, or they are a legitimate user themselves).

Figure 1 shows the interactions between the adversary, HSM API, and memory locations in the standard attack scenario. HSM functions may access any memory locations, while the adversary can only access the low security locations. A similar setup applies in the case of software APIs, where the adversary is a malicious client program and the high memory locations correspond to those which are hidden by the API.

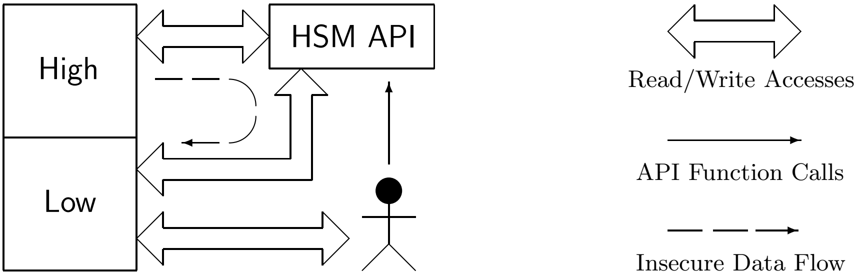


Fig. 1. The interactions between the adversary, the HSM API, and the memory locations

n	$\in$	$\mathcal{N}$	*Names*
a	$\in$	$\mathcal{A}$	*Locations*
l	$\in$	$\mathcal{L}$	*Security levels* (where $\bot,\top \in \mathcal{L}$)
e	$::=$	$n \mid\ !a \mid \mathrm{senc}(e,e) \mid \mathrm{sdec}(e,e) \mid \mathrm{junk}(e)$	*Expressions*
c	$::=$	$a := e \mid c\,;c \mid \epsilon$	*Commands*
u	$::=$	$n \mid \mathrm{senc}(u,u)$	*Non-junk values*
v	$::=$	$u \mid \mathrm{junk}(u)$	*Values*
E	$::=$	$l\,\mathsf{data} \mid l\,\mathsf{key} \mid \mathsf{enc}(\mathsf{E})$	*Security types for expressions*
A	$::=$	$\mathsf{E\,loc}$	*Security type for locations*
C	$::=$	cmd	*Security type for commands*
T	$::=$	$\mathsf{E} \mid \mathsf{A} \mid \mathsf{C}$	*All security types*
ϕ	$::=$	$\phi\,, a \mapsto v \mid \epsilon$	*Store*
Δ	$::=$	$\Delta\,, n:l \mid \Delta\,, a:l \mid \epsilon$	*Security levels environment*
Γ	$::=$	$\Gamma, n:l\,\mathsf{data} \mid \Gamma, n:l\,\mathsf{key} \mid \Gamma, a:\mathsf{A} \mid \epsilon$	*Security types environment*

Fig. 2. Fundamental syntax definitions

The adversary's goal is to execute a series of API function calls such that sensitive data is unintentionally written to the low security memory, or that sensitive data can be inferred from the API's low security output. The aim of security API analysis is to detect such insecure data flows, or to guarantee that no such flows exist.

3 Type System

Figure 2 presents the fundamental syntax definitions upon which our type system is built. The set $\mathcal{N}$ comprises *names* representing regular data items and cryptographic keys; $\mathcal{A}$ is the set of abstract memory *locations*, and $\mathcal{L}$ is the set of security levels which may be associated to names and locations.[1] Although our type system allows for any number of security levels (where $l \in \mathcal{L} \rightarrow \bot \leq l \leq \top$), in this paper we only consider $\bot$ and $\top$ (i.e., low and high) in order to simplify the presentation and discussion.

An *expression* can be a name, the contents of a memory location, the result of a (symmetric) encryption or decryption, or 'junk' which denotes an incorrectly decrypted term. Junk terms contain the expression which would have resulted had the correct decryption key(s) been used, so we can ensure that a junk expression is treated in the same way as the intended non-junk term. A *command* is zero or more instances of the assignment operation. A *value* is a name, a fully evaluated ciphertext, or a junk value.

[1] Security levels are for analysis purposes only — they do not exist in practice (and even if they did, the adversary would not gain anything from knowing a term's security level).

The security types for regular data items and cryptographic keys, $l\,\mathsf{data}$ and $l\,\mathsf{key}$ respectively, associate a security level with their terms. The level associated with regular data denotes the confidentiality level of that data item, whereas the one associated with a cryptographic key denotes the maximum security level of expressions which that key may encrypt. The different semantics are necessary because we allow the security level of expressions to be arbitrarily increased and therefore cannot determine what a key may encrypt based solely on its security level. For example, a low security key can be considered a high security key and thus could be used to encrypt other high security expressions. This approach also allows a high security key to encrypt and decrypt low security expressions without forcing the result of the decryption to be high security.

To recover the precise type of encrypted data when it is subsequently decrypted, we use a type operator; $\mathsf{enc}(\mathsf{E})$ is the type of an encrypted expression of type E. This means that no type information is lost as a result of the encryption/decryption process and also allows us to handle nested encryptions. The security type for locations denotes the most general type of expression which may be stored in that location. We do not associate a security level with the command type, although we will do so in future work.

The store, ϕ, maps locations to values; the security levels environment, Δ, contains the security levels of names and locations (used dynamically), and the security types environment, Γ, contains the security types of names and locations (used statically). We assume there is no overlap between the identifiers used for names and for locations.

3.1 Operational Semantics

Figure 3 presents the operational semantics for command sequences which we consider. We wish to enforce the restriction that only ciphertext is decrypted, therefore any term which evaluates to the decryption of non-encrypted data will get 'stuck.' That is, the term cannot be fully evaluated under the operational semantics.

Other terms which will get stuck include the encryption of data with a key whose security level is too low, the assignment of a value to a location whose security level is too low, and the dereferencing of something other than a location. The first two of these correspond to cases where continued evaluation would result in a security breach:

$$\Delta = \{a:\bot, v_k:\bot,\, v_m:\top\} \qquad \begin{array}{ll} a \;:= \mathrm{senc}(v_k, v_m) & (1) \\ a \;:= v_m & (2) \end{array}$$

Getting stuck in case (1) guarantees that, if the adversary is able to decrypt some given piece of ciphertext, the result will not be sensitive, while getting stuck in case (2) guarantees that sensitive data cannot be written directly to a low security memory location. This latter property is known as 'no-write down,' and is enforced by the rule E-Assign2. The 'no read-up' property follows from the assumption that an observer is only able to read the contents of locations whose

Expressions $\boxed{e \rightarrow_{\Delta\,\phi} e'}$

$$\frac{e_k \rightarrow_{\Delta\,\phi} {e_k}'}{\mathrm{senc}(e_k, e_m) \rightarrow_{\Delta\,\phi} \mathrm{senc}({e_k}', e_m)}\ \text{E-SEnc1}$$

$$\frac{e \rightarrow_{\Delta\,\phi} e'}{\mathrm{senc}(v, e) \rightarrow_{\Delta\,\phi} \mathrm{senc}(v, e')}\ \text{E-SEnc2}$$

$$\frac{}{\mathrm{senc}(u, \mathrm{junk}(u')) \rightarrow_{\Delta\,\phi} \mathrm{junk}(\mathrm{senc}(u, u'))}\ \text{E-SEnc3}$$

$$\frac{}{\mathrm{senc}(\mathrm{junk}(u), v) \rightarrow_{\Delta\,\phi} \mathrm{junk}(\mathrm{senc}(u, v))}\ \text{E-SEnc4}$$

$$\frac{e_k \rightarrow_{\Delta\,\phi} {e_k}'}{\mathrm{sdec}(e_k, e_m) \rightarrow_{\Delta\,\phi} \mathrm{sdec}({e_k}', e_m)}\ \text{E-SDec1}$$

$$\frac{e \rightarrow_{\Delta\,\phi} e'}{\mathrm{sdec}(v, e) \rightarrow_{\Delta\,\phi} \mathrm{sdec}(v, e')}\ \text{E-SDec2}$$

$$\frac{u_k' \neq u_k \quad \Delta \vdash u_k', u_k : l_k \quad \Delta \vdash u_m : l_m \quad l_m \leq l_k}{\mathrm{sdec}(u_k', \mathrm{senc}(u_k, u_m)) \rightarrow_{\Delta\,\phi} \mathrm{junk}(u_m)}\ \text{E-SDec4}$$

$$\frac{}{\mathrm{sdec}(u, \mathrm{junk}(u')) \rightarrow_{\Delta\,\phi} \mathrm{junk}(\mathrm{sdec}(u, u'))}\ \text{E-SDec5}$$

$$\frac{}{\mathrm{sdec}(\mathrm{junk}(u), v) \rightarrow_{\Delta\,\phi} \mathrm{junk}(\mathrm{sdec}(u, v))}\ \text{E-SDec6}$$

Commands $\boxed{\langle \phi, c \rangle \rightarrow_{\Delta} \langle \phi', c' \rangle}$

$$\frac{\langle \phi, c_1 \rangle \rightarrow_{\Delta} \langle \phi', c_1' \rangle}{\langle \phi, c_1 ; c_2 \rangle \rightarrow_{\Delta} \langle \phi', c_1' ; c_2 \rangle}\ \text{E-Cmds1}$$

$$\frac{}{\langle \phi, \epsilon\, ; c \rangle \rightarrow_{\Delta} \langle \phi, c \rangle}\ \text{E-Cmds2}$$

$$\frac{e \rightarrow_{\Delta\,\phi} e'}{\langle \phi, a := e \rangle \rightarrow_{\Delta} \langle \phi, a := e' \rangle}\ \text{E-Assign1}$$

$$\frac{\Delta \vdash a : l_a \quad \Delta \vdash v : l_v \quad l_v \leq l_a}{\langle \phi, a := v \rangle \rightarrow_{\Delta} \langle \phi[a \mapsto v], \epsilon \rangle}\ \text{E-Assign2}$$

$$\frac{\Delta \vdash u_k : l_k \quad \Delta \vdash u_m : l_m \quad l_m \leq l_k}{\mathrm{sdec}(u_k, \mathrm{senc}(u_k, u_m)) \rightarrow_{\Delta\,\phi} u_m}\ \text{E-SDec3}$$

$$\frac{}{!a \rightarrow_{\Delta\,\phi} \phi(a)}\ \text{E-Deref}$$

$$\frac{e \rightarrow_{\Delta\,\phi} e'}{\mathrm{junk}(e) \rightarrow_{\Delta\,\phi} \mathrm{junk}(e')}\ \text{E-Junk1}$$

$$\frac{}{\mathrm{junk}(\mathrm{junk}(u)) \rightarrow_{\Delta\,\phi} \mathrm{junk}(u)}\ \text{E-Junk2}$$

Security Levels of Values

$$\frac{n : l \in \Delta}{\Delta \vdash n : l} \qquad \frac{a : l \in \Delta}{\Delta \vdash a : l} \qquad \frac{\Delta \vdash u_k : l_k \quad \Delta \vdash u_m : l_m \quad l_m \leq l_k}{\Delta \vdash \mathrm{senc}(u_k, u_m) : \bot} \qquad \frac{\Delta \vdash u : l}{\Delta \vdash \mathrm{junk}(u) : l}$$

Fig. 3. The operational semantics for command sequences

associated security level is low enough. This is a legitimate assumption, since the sensitive locations will be those which are inside the tamper-proof HSM whose security API is being analysed, or in the case of software APIs, those locations which are hidden from client programs.

The junk term is returned when a piece of ciphertext is decrypted with a different key from the one which was used to create it (E-SDec4), or when the key or message in an encryption or decryption operation is junk (E-SEnc3, E-SEnc4, E-SDec5 and E-SDec6). In all cases, the expression within the junk term is that which would have been returned had the correct decryption key(s) been used.

Encryption requires that the security level of the key is at least as high as that of the message. However, this restriction is not enforced when the ciphertext is actually created, but rather when it is subsequently decrypted or assigned to a

Commands

$$\frac{\Gamma \vdash c_1 : \mathsf{cmd} \quad \Gamma \vdash c_2 : \mathsf{cmd}}{\Gamma \vdash c_1 \,;\, c_2 : \mathsf{cmd}}\ \textsc{T-Cmds}$$

$$\frac{\Gamma \vdash a : \mathsf{E\,loc} \quad \Gamma \vdash e : \mathsf{E}}{\Gamma \vdash a := e : \mathsf{cmd}}\ \textsc{T-Assign} \qquad \frac{}{\Gamma \vdash \epsilon : \mathsf{cmd}}\ \textsc{T-Empty}$$

Expressions

$$\frac{n : \mathsf{E} \in \Gamma}{\Gamma \vdash n : \mathsf{E}}\ \textsc{T-Name} \qquad \frac{\Gamma \vdash e : \mathsf{E}}{\Gamma \vdash \mathrm{junk}(e) : \mathsf{E}}\ \textsc{T-Junk}$$

$$\frac{a : \mathsf{E\,loc} \in \Gamma}{\Gamma \vdash a : \mathsf{E\,loc}}\ \textsc{T-Loc} \qquad \frac{\Gamma \vdash a : \mathsf{E\,loc}}{\Gamma \vdash\, !a : \mathsf{E}}\ \textsc{T-Deref}$$

$$\frac{\Gamma \vdash e_k : l\,\mathsf{key} \quad \Gamma \vdash e_m : \mathsf{E} \quad \mathit{lvl}(\mathsf{E}) = l}{\Gamma \vdash \mathrm{senc}(e_k, e_m) : \mathsf{enc}(\mathsf{E})}\ \textsc{T-SEnc}$$

$$\frac{\Gamma \vdash e_k : l\,\mathsf{key} \quad \Gamma \vdash e_m : \mathsf{enc}(\mathsf{E}) \quad \mathit{lvl}(\mathsf{E}) = l}{\Gamma \vdash \mathrm{sdec}(e_k, e_m) : \mathsf{E}}\ \textsc{T-SDec}$$

Subtyping

$$\frac{\Gamma \vdash t : \mathsf{T}' \quad \mathsf{T}' <: \mathsf{T}}{\Gamma \vdash t : \mathsf{T}}\ \textsc{T-Sub} \qquad \frac{}{\mathsf{T} <: \mathsf{T}}$$

$$\frac{\mathsf{T} <: \mathsf{T}'' \quad \mathsf{T}'' <: \mathsf{T}'}{\mathsf{T} <: \mathsf{T}'}$$

$$\frac{l \leq l'}{l\,\mathsf{data} <: l'\,\mathsf{data}} \qquad \frac{\mathsf{E} <: \mathsf{E}'}{\mathsf{enc}(\mathsf{E}) <: \mathsf{enc}(\mathsf{E}')}$$

$$\frac{}{l\,\mathsf{key} <: \top\,\mathsf{data}} \qquad \frac{}{\mathsf{enc}(\mathsf{E}) <: \bot\,\mathsf{data}}$$

Security Levels of Types

$$\mathit{lvl}(\mathsf{cmd}) = \bot \qquad \mathit{lvl}(l\,\mathsf{data}) = l$$

$$\mathit{lvl}(l\,\mathsf{key}) = \top \qquad \mathit{lvl}(\mathsf{enc}(\mathsf{E})) = \bot$$

$$\mathit{lvl}(\mathsf{E\,loc}) = \mathit{lvl}(\mathsf{E})$$

Fig. 4. The typing rules of our system

location (i.e., when the security level of the ciphertext has to be determined). The security level of ciphertext is $\bot$ since encryption is used primarily as a means of securely declassifying sensitive data. If the result of an encryption should itself be sensitive, then this can be achieved simply by assigning the ciphertext to a location which stores sensitive data and returning a reference to that location.

3.2 Typing Rules

Figure 4 presents the rules of our type system. As noted previously, a location's type denotes the most general type of values which can be stored in that location. By design, the more general a type is, the greater its security level (i.e., $\mathsf{E} <: \mathsf{E}' \rightarrow \mathit{lvl}(\mathsf{E}) \leq \mathit{lvl}(\mathsf{E}')$). Therefore, the typing rule for assignment (T-Assign) guarantees the 'no write-down' property since the security level associated with the location will be no lower than the one associated with the expression.

Junk terms can have any expression type (T-Junk), since they are generated only as the result of a decryption with the wrong key, and we wish to consider a junk term as being equivalent to the intended result, had the correct decryption key been used. This is to prevent insecure information flows which may otherwise result from the use of an incorrect decryption key.

The contents of a location are given the type of the most general expression that can be stored in that location (T-Deref). Thus, any security result is independent of the values which are actually stored in each memory location.

For encryption, the key used must be able to encrypt messages which are at least as secure as the actual message (T-SEnc). For decryption, the message must be ciphertext and the security level associated with the key must be no lower than the security level associated with the result (T-SDec).

Currently, we restrict keys to having the highest security level, and commands to having the lowest security level, since our focus is on security APIs with secret keys and public functions. Relaxing these restrictions will form part of our future work.

To prove the theorems presented in this paper, we require a couple of standard type-theoretic lemmas. The proofs are quite straightforward and have been omitted.

Lemma 1. *Generation Lemma (Inversion of the Typing Relation)*

1. If $\Gamma \vdash n : \mathsf{T}$ then $\mathsf{T} :> \mathsf{E}$ and $n : \mathsf{E} \in \Gamma$.
2. If $\Gamma \vdash a : \mathsf{T}$ then $\mathsf{T} \equiv \mathsf{E\,loc}$ and $a : \mathsf{E\,loc} \in \Gamma$.
3. If $\Gamma \vdash\, !a : \mathsf{T}$ then $\mathsf{T} :> \mathsf{E}$ and $\Gamma \vdash a : \mathsf{E\,loc}$.
4. If $\Gamma \vdash \mathrm{senc}(e_1, e_2) : \mathsf{T}$ then $\mathsf{T} :> \mathsf{enc}(\mathsf{E})$, $\Gamma \vdash e_1 : l\,\mathsf{key}$, $\Gamma \vdash e_2 : \mathsf{E}$ and $lvl(\mathsf{E}) = l$.
5. If $\Gamma \vdash \mathrm{sdec}(e_1, e_2) : \mathsf{T}$ then $\mathsf{T} :> \mathsf{E}$, $\Gamma \vdash e_1 : l\,\mathsf{key}$, $\Gamma \vdash e_2 : \mathsf{enc}(\mathsf{E})$ and $lvl(\mathsf{E}) = l$.
6. If $\Gamma \vdash \mathrm{junk}(e) : \mathsf{T}$ then $\Gamma \vdash e : \mathsf{T}$.
7. If $\Gamma \vdash a := e : \mathsf{T}$ then $\mathsf{T} \equiv \mathsf{cmd}$, $\Gamma \vdash a : \mathsf{E\,loc}$, and $\Gamma \vdash e : \mathsf{E}$.
8. If $\Gamma \vdash \epsilon : \mathsf{T}$ then $\mathsf{T} \equiv \mathsf{cmd}$.
9. If $\Gamma \vdash c_1 \,; c_2 : \mathsf{T}$ then $\mathsf{T} \equiv \mathsf{cmd}$, $\Gamma \vdash c_1 : \mathsf{cmd}$, and $\Gamma \vdash c_2 : \mathsf{cmd}$.

Proof. Follows from induction on the typing derivations.

Lemma 2. *Canonical Forms Lemma*

1. If $\Gamma \vdash v : \mathsf{enc}(\mathsf{E})$ then $v \equiv \mathrm{senc}(u_k, u_m)$ or $\mathrm{junk}(\mathrm{senc}(u_k, u_m))$.
2. If $\Gamma \vdash v : l\,\mathsf{key}$ then $v \equiv n$ or $\mathrm{junk}(n)$.
3. If $\Gamma \vdash v : l\,\mathsf{data}$ then $v \equiv n$, $\mathrm{senc}(u_k, u_m)$, $\mathrm{junk}(n)$ or $\mathrm{junk}(\mathrm{senc}(u_k, u_m))$.

Proof. Follows from inspection of the typing rules and fundamental definitions.

4 Progress and Preservation

The standard way to establish *type safety* for a type system with respect to an operational semantics is to show that the *progress* and *preservation* properties hold. Preservation establishes that the type of a term is not changed by the evaluation rules, while progress demonstrates that well-typed terms will not get 'stuck.' Stuck terms represent certain error conditions that may arise during evaluation. In our system, for example, a term becomes stuck whenever further evaluation would result in a security leak. Such leaks are prevented in the operational semantics by checks carried out on the security levels in a number of the evaluation rules.

For the progress and preservation properties to hold, the initial store ϕ must be *well-typed*, and the security levels environment Δ must be *level-consistent* with respect to the typing context Γ. Informally, ϕ is well-typed if every value in ϕ has the type predicted by Γ, while Δ is level-consistent with respect to Γ if every name and location in Δ has the same security level as given to it by Γ.

Definition 1. *A store ϕ is* well-typed *with respect to a typing context Γ, written $\Gamma \vdash \phi$, if $dom(\phi) = dom(\Gamma \,|\, \mathsf{loc})$ and, $\forall a \in dom(\phi)$, $\exists \mathsf{E}.\ \Gamma \vdash \phi(a) : \mathsf{E} \wedge \Gamma \vdash a : \mathsf{E\,loc}$.*

Definition 2. *A security levels environment Δ is* level-consistent *with respect to a typing context Γ, written $\Gamma \vdash \Delta$, if $dom(\Delta) = dom(\Gamma)$, and*

- *$\forall n \in dom(\Gamma \,|\, \mathsf{nam})$, $n : \mathsf{E} \in \Gamma \rightarrow n : lvl(\mathsf{E}) \in \Delta$*
- *$\forall a \in dom(\Gamma \,|\, \mathsf{loc})$, $a : \mathsf{E\,loc} \in \Gamma \rightarrow a : lvl(\mathsf{E}) \in \Delta$*

Here, $\mathcal{S} \,|\, \mathsf{nam}$ and $\mathcal{S} \,|\, \mathsf{loc}$ denote the subsets of $\mathcal{S}$ containing only those elements which are names and locations respectively.

Corollary 1. *If $\Gamma \vdash \Delta$, $\Gamma \vdash v : \mathsf{E}$ and $\Delta \vdash v : l$, then $l \leq lvl(\mathsf{E})$.*

Proof. By definition, $v \equiv n$, junk(n), senc(u_k, u_m) or junk(senc(u_k, u_m)). If $v \equiv n$ or junk(n) then, by Lemma 1, $n : \mathsf{E}' \in \Gamma$, where $\mathsf{E}' <: \mathsf{E}$. By $\Gamma \vdash \Delta$, $n : lvl(\mathsf{E}')$ $\in \Delta$ therefore $\Delta \vdash v : lvl(\mathsf{E}')$ and $l = lvl(\mathsf{E}')$. It then follows from $\mathsf{E}' <: \mathsf{E}$ that $lvl(\mathsf{E}') \leq lvl(\mathsf{E})$, so the result holds. If $v \equiv$ senc(u_k, u_m) or junk(senc(u_k, u_m)) then $l = \bot$, so the result holds. □

Theorem 1. *Progress*

i) *If $\Gamma \vdash t : \mathsf{E}$, then either t is a value, or else for any security levels environment Δ and store ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some t' such that $t \rightarrow_{\Delta\,\phi} t'$.*

ii) *If $\Gamma \vdash t : \mathsf{C}$, then either t is the empty command ϵ or else, for any security types environment Δ and store ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some t' and ϕ' such that $\langle \phi\,, t \rangle \rightarrow_{\Delta} \langle \phi', t' \rangle$.*

Proof. By induction on $\Gamma \vdash t : \mathsf{E}$ and $\Gamma \vdash t : \mathsf{C}$: *(selected cases only)*

- Case T-Deref: $t : \mathsf{E} \equiv\ !a : \mathsf{E} \qquad a : \mathsf{E\,loc}$
  The rule E-Deref applies (it follows from $\Gamma \vdash \phi$ that $a \in \phi$).
- Case T-SEnc: $t : \mathsf{E} \equiv$ senc(e_k, e_m) : $\mathsf{enc}(\mathsf{E}')$ $\qquad e_k : l\,\mathsf{key} \qquad e_m : \mathsf{E}' \qquad lvl(\mathsf{E}') = l$
  By the induction hypothesis, either e_k is a value, or else for any Δ and ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some e_k' such that $e_k \rightarrow_{\Delta\,\phi} e_k'$. Similarly for e_m. If e_k is not a value then E-SEnc1 applies; if e_m is not a value (but e_k is) then E-SEnc2 applies; if e_k is a junk value then E-SEnc4 applies; if e_m is a junk value (and e_k is a non-junk value) then E-SEnc3 applies; if both e_k and e_m are non-junk values then t is a value.

- Case T-SDEC: $t : \mathsf{E} \equiv \mathrm{sdec}(e_k, e_m) : \mathsf{E} \quad e_k : l\,\mathsf{key} \quad e_m : \mathsf{enc}(\mathsf{E}) \quad \mathit{lvl}(\mathsf{E}) = l$
  By the induction hypothesis, either e_k is a value, or else for any Δ and ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some e'_k such that $e_k \rightharpoonup_{\Delta\,\phi} e'_k$. Similarly for e_m. If e_k is not a value then E-SDEC1 applies, and if e_m is not a value (but e_k is) then E-SDEC2 applies. If e_k is a value then, by Lemma 2, it must be of the form n or junk(n). The former case is covered by the rules E-SDEC3, E-SDEC4 and E-SDEC5 as described below; in the latter case, E-SDEC6 applies. If e_m is a value then, by Lemma 2, it must be of the form $\mathrm{senc}(u_k, u_m)$ or $\mathrm{junk}(\mathrm{senc}(u_k, u_m))$. In the first case, it follows from Lemma 1 that $\Gamma \vdash u_k : l'\,\mathsf{key}$, $\Gamma \vdash u_m : \mathsf{E}'$ and $\mathit{lvl}(\mathsf{E}') = l'$, where $\mathsf{enc}(\mathsf{E}) :> \mathsf{enc}(\mathsf{E}')$ (therefore $\mathsf{E} :> \mathsf{E}'$). If $e_k = u_k$ then E-SDEC3 applies and if $e_k \neq u_k$ then E-SDEC4 applies. In the second case, where $e_m \equiv \mathrm{junk}(\mathrm{senc}(u_k, u_m))$, the rule E-SDEC5 applies. For rules E-SDEC3 and E-SDEC4, the inequality $l_m \leq l_k$ will be satisfied because it follows from Lemma 1 that $e_k : l\,\mathsf{key} \in \Gamma$, and from $\Gamma \vdash \Delta$ that $e_k : \top \in \Delta$, thus $l_k = \top$.
- Case T-ASSIGN: $t : \mathsf{C} \equiv a := e : \mathsf{cmd} \quad a : \mathsf{E}\,\mathsf{loc} \quad e : \mathsf{E}$
  By the induction hypothesis for Part (i), either e is a value, or else for any Δ and ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some e' such that $e \rightharpoonup_{\Delta\,\phi} e'$. If e is a value, then E-ASSIGN2 applies, otherwise E-ASSIGN1 applies. In the former case, the inequality will hold because, by Lemma 1, $a : \mathsf{E}\,\mathsf{loc} \in \Gamma$, by $\Gamma \vdash \Delta$, $a : \mathit{lvl}(\mathsf{E}) \in \Delta$ therefore $l_a = \mathit{lvl}(\mathsf{E})$, and by Cor. 1, $l_v \leq \mathit{lvl}(\mathsf{E})$.
- Case T-CMDS: $t : \mathsf{C} \equiv c_1\,; c_2 : \mathsf{cmd} \quad c_1 : \mathsf{cmd} \quad c_2 : \mathsf{cmd}$
  By the induction hypothesis, either c_1 is the empty command ϵ or else, for any Δ and ϕ such that $\Gamma \vdash \Delta$ and $\Gamma \vdash \phi$, there exists some c'_1 and ϕ' such that $\langle \phi, c_1 \rangle \rightharpoonup_{\Delta} \langle \phi', c'_1 \rangle$. If $c_1 \equiv \epsilon$ then the rule E-CMDS2 applies, otherwise the rule E-CMDS1 applies. □

Theorem 2. *Preservation*

i) *If $\Gamma \vdash t : E$, $\Gamma \vdash \Delta, \phi$ and there exists some t' such that $t \rightharpoonup_{\Delta\,\phi} t'$, then $\Gamma \vdash t' : E$.*

ii) *If $\Gamma \vdash t : C$, $\Gamma \vdash \Delta, \phi$ and there exists some t' and ϕ' such that $\langle \phi\,, t \rangle \rightharpoonup_{\Delta} \langle \phi', t' \rangle$, then $\Gamma \vdash \phi'$ and $\Gamma \vdash t' : C$.*

Proof. By induction on $\Gamma \vdash t : \mathsf{E}$ and $\Gamma \vdash t : \mathsf{C}$: *(selected cases only)*

- Case T-DEREF: $t : \mathsf{E} \equiv\ !a : \mathsf{E} \quad a : \mathsf{E}\,\mathsf{loc}$
  E-DEREF is the only evaluation rule which may apply, therefore $t' \equiv \phi(a)$. By $\Gamma \vdash \phi$, $\exists \mathsf{E}'$ such that $\Gamma \vdash a : \mathsf{E}'\,\mathsf{loc}$ and $\Gamma \vdash \phi(a) : \mathsf{E}'$. It therefore follows that $\mathsf{E}' \equiv \mathsf{E}$ and so the result holds.
- Case T-SENC: $t : \mathsf{E} \equiv \mathrm{senc}(e_k, e_m) : \mathsf{enc}(\mathsf{E}) \quad e_k : l\,\mathsf{key} \quad e_m : \mathsf{E} \quad \mathit{lvl}(\mathsf{E}) = l$
  There are four evaluation rules which correspond to the transition $t \rightharpoonup_{\Delta\,\phi} t'$: E-SENC1 through E-SENC4. Subcase E-SENC2 has a similar proof to subcase E-SENC1, and subcase E-SENC4 has a similar proof to subcase E-SENC3.

  - Subcase E-SENC1: $e_k \rightharpoonup_{\vartriangle\phi} e'_k \quad t' \equiv \mathrm{senc}(e'_k, e_m)$
    The T-SENC rule has a subderivation whose conclusion is $e_k : l\,\mathsf{key}$ and the induction hypothesis gives us $e'_k : l\,\mathsf{key}$. Therefore, in conjunction with $e_m : \mathsf{E}$ and $\mathit{lvl}(\mathsf{E}) = l$, we can apply the rule T-SENC to conclude that $\mathrm{senc}(e'_k, e_m) : \mathsf{enc}(\mathsf{E})$.
  - Subcase E-SENC3: $e_k \equiv u_k \quad e_m \equiv \mathrm{junk}(u_m) \quad t' \equiv \mathrm{junk}(\mathrm{senc}(u_k, u_m))$
    The T-SENC rule has a subderivation whose conclusion is $\mathrm{junk}(u_m) : \mathsf{E}$, and by Lemma 1 we get $u_m : \mathsf{E}$. Therefore, in conjunction with $u_k : l\,\mathsf{key}$ and $\mathit{lvl}(\mathsf{E}) = l$, we can apply T-SENC and T-JUNK to conclude that $\mathrm{junk}(\mathrm{senc}(u_k, u_m)) : \mathsf{enc}(\mathsf{E})$.

- Case T-SDEC: $t : \mathsf{E} \equiv \mathrm{sdec}(e_k, e_m) : \mathsf{E} \quad e_k : l\,\mathsf{key} \quad e_m : \mathsf{enc}(\mathsf{E}) \quad \mathit{lvl}(\mathsf{E}) = l$
  There are six evaluation rules which correspond to the transition $t \rightharpoonup_{\vartriangle\phi} t'$: E-SDEC1 through E-SDEC6. Subcases E-SDEC1 and E-SDEC2 have similar proofs to subcase E-SENC1 above; subcases E-SDEC5 and E-SDEC6 have a similar proof to subcase E-SENC3 above.
  - Subcase E-SDEC3: $e_k \equiv n \quad e_m \equiv \mathrm{senc}(n, t')$
    The T-SDEC rule has a subderivation whose conclusion is $\mathrm{senc}(n, t') : \mathsf{enc}(\mathsf{E})$. It follows from Lemma 1 that $\Gamma \vdash t' : \mathsf{E}'$ and $\mathsf{enc}(\mathsf{E}') <: \mathsf{enc}(\mathsf{E})$. Thus $\mathsf{E}' <: \mathsf{E}$, and we can apply the T-SUB rule to conclude that $\Gamma \vdash t' : \mathsf{E}$.
  - Subcase E-SDEC4: $e_m \equiv \mathrm{senc}(u_k, u_m) \quad e_k \neq u_k \quad t' = \mathrm{junk}(u_m)$
    The T-SDEC rule has a subderivation whose conclusion is $\mathrm{senc}(u_k, u_m) : \mathsf{enc}(\mathsf{E})$. It follows from Lemma 1 that $\Gamma \vdash u_m : \mathsf{E}'$and $\mathsf{enc}(\mathsf{E}') <: \mathsf{enc}(\mathsf{E})$. Thus $\mathsf{E}' <: \mathsf{E}$, and we can apply T-SUB and T-JUNK to conclude that $\Gamma \vdash \mathrm{junk}(u_m) : \mathsf{E}$.

- Case T-ASSIGN: $t : \mathsf{C} \equiv a := e : \mathsf{cmd} \quad a : \mathsf{E}\,\mathsf{loc} \quad e : \mathsf{E}$
  Two evaluation rules may correspond to the transition $\langle \phi, t\rangle \rightharpoonup_{\vartriangle} \langle \phi', t'\rangle$: E-ASSIGN1 and E-ASSIGN2. The proof for the latter is trivial.
  - Subcase E-ASSIGN1: $\langle \phi, e\rangle \rightharpoonup_{\vartriangle} \langle \phi, e'\rangle \quad t' = a := e'$
    The T-ASSIGN rule has a subderivation whose conclusion is $e : \mathsf{E}$. Applying the induction hypothesis to this subderivation gives us $\Gamma \vdash e' : \mathsf{E}$. In conjunction with the other subderivation $\Gamma \vdash a : \mathsf{E}\,\mathsf{loc}$, we can apply T-ASSIGN to conclude that $\Gamma \vdash a := e' : \mathsf{cmd}$. $\Gamma \vdash \phi'$ follows immediately from the fact that $\phi = \phi'$.

- Case T-CMDS: $t : \mathsf{C} \equiv c_1 \,;\, c_2 : \mathsf{cmd} \quad c_1 : \mathsf{cmd} \quad c_2 : \mathsf{cmd}$
  Two evaluation rules may correspond to the transition $\langle \phi, t\rangle \rightharpoonup_{\vartriangle} \langle \phi', t'\rangle$: E-CMDS1 and E-CMDS2. The proof for the latter is trivial.
  - Subcase E-CMDS1: $\langle \phi, c_1\rangle \rightharpoonup_{\vartriangle} \langle \phi', c'_1\rangle \quad t' = c'_1 \,;\, c_2$
    The T-CMDS rule has a subderivation whose conclusion is $c_1 : \mathsf{cmd}$ and the induction hypothesis gives us $\Gamma \vdash \phi'$ and $\Gamma \vdash c'_1 : \mathsf{cmd}$. Using the latter of these, in conjunction with the other subderivation $\Gamma \vdash c_2 : \mathsf{cmd}$, we can apply the rule T-CMDS to conclude that $\Gamma \vdash c'_1 \,;\, c_2 : \mathsf{cmd}$. □

The following lemma states that the type of an expression is preserved under evaluation with respect to a well-typed store, independent of the actual values contained in the locations of that store, and is required to prove our non-interference result.

Lemma 3. *If* $(\Gamma, a : \mathsf{E\,loc}) \vdash e : \mathsf{E}'$*,* $\Gamma \vdash v : \mathsf{E}$*,* $(\Gamma, a : \mathsf{E\,loc}) \vdash \Delta, \phi$ *and* $e \rightharpoonup^{*}_{\Delta\,\phi'} v'$*, where* $\phi' \equiv \phi[a \mapsto v]$*, then* $\Gamma \vdash v' : \mathsf{E}'$*.*

Proof. By induction on $(\Gamma, a : \mathsf{E\,loc}) \vdash e : \mathsf{E}'$:*(selected cases only)*

- Case T-Deref: $e : \mathsf{E}' \equiv\ !a' : \mathsf{E}' \qquad a' : \mathsf{E}'\,\mathsf{loc}$
  $!a' \rightharpoonup_{\Delta\,\phi'} \phi'(a')$. By Lemma 1, $a' : \mathsf{E}'\,\mathsf{loc} \in \Gamma$. If $a = a'$ then $v' = v$ and $\mathsf{E}' \equiv \mathsf{E}$, thus the result holds. If $a \neq a'$ then $v' = \phi(a')$ and the result follows from $(\Gamma, a : \mathsf{E\,loc}) \vdash \phi$.
- Case T-SEnc: $e : \mathsf{E}' \equiv \mathrm{senc}(e_k, e_m) : \mathsf{enc}(\mathsf{E}'') \qquad e_k : l\,\mathsf{key} \qquad e_m : \mathsf{E}''$
  $lvl(\mathsf{E}'') = l$
  $\mathrm{senc}(e_k, e_m) \rightharpoonup^{*}_{\Delta\,\phi'} \mathrm{senc}(v_k, v_m) \rightharpoonup^{*}_{\Delta\,\phi'} v'$, where $e_k \rightharpoonup^{*}_{\Delta\,\phi'} v_k$ and $e_m \rightharpoonup^{*}_{\Delta\,\phi'} v_m$. By the induction hypothesis, $\Gamma \vdash v_k : l\,\mathsf{key}$ and $\Gamma \vdash v_m : \mathsf{E}''$. If v_k and v_m are both non-junk values, then $v' \equiv \mathrm{senc}(v_k, v_m)$ and the the result follows from T-SEnc. Otherwise, $v_k \equiv \mathrm{junk}(u_k)$ and/or $v_m \equiv \mathrm{junk}(u_m)$, therefore $v' \equiv \mathrm{junk}(\mathrm{senc}(u_k, u_m))$ and the result follows from T-Junk and T-SEnc.
- Case T-SDec: $e : \mathsf{E}' \equiv \mathrm{sdec}(e_k, e_m) : \mathsf{E}' \qquad e_k : l\,\mathsf{key} \qquad e_m : \mathsf{enc}(\mathsf{E}')$
  $\mathrm{sdec}(e_k, e_m) \rightharpoonup^{*}_{\Delta\,\phi'} \mathrm{sdec}(v_k, v_m) \rightharpoonup^{*}_{\Delta\,\phi'} v'$ where $e_k \rightharpoonup^{*}_{\Delta\,\phi'} v_k$ and $e_m \rightharpoonup^{*}_{\Delta\,\phi'} v_m$. By the induction hypothesis, $\Gamma \vdash v_k : l\,\mathsf{key}$ and $\Gamma \vdash v_m : \mathsf{enc}(\mathsf{E}')$, and by Lemma 2, v_k is of the form n or $\mathrm{junk}(n)$, and v_m is of the form $\mathrm{senc}(u_k, u_m)$ or $\mathrm{junk}(\mathrm{senc}(u_k, u_m))$. In both cases for v_m, it follows from Lemma 1 that $\Gamma \vdash u_m : \mathsf{E}''$, where $\mathsf{E}'' <: \mathsf{E}'$. By inspection of the evaluation rules, v' will be of the form u_m or $\mathrm{junk}(u_m)$. In the first case, we can apply T-Sub to $\Gamma \vdash u_m : \mathsf{E}''$ and $\mathsf{E}'' <: \mathsf{E}'$ to conclude that $\Gamma \vdash u_m : \mathsf{E}'$; in the second case, the result follows from T-Sub and T-Junk. □

5 Indistinguishability

Our type system is intended for analysing systems with ciphers that are *repetition concealing* and *which-key concealing* — also known as type-1 ciphers ([10], Sec. 4.2). Repetition concealing means that it is not possible to say whether two messages encrypted under the same key are equal. Which-key concealing means that it is not possible to say whether two keys used to encrypt the same message are equal. Both of these properties are possessed by standard block ciphers, such as DES and AES, when used in CBC or CTR mode ([10], Sec. 4.4). However, these definitions assume that the adversary is unable to correctly decrypt the ciphertexts. This is not strictly the case with security APIs: the API functions can be used to decrypt ciphertexts whose contents are public, whilst keeping the actual values of the keys secret. As a result, we have to capture the ability of the adversary to distinguish between ciphertexts which contain public data, under certain circumstances.

We use the notation $\Gamma \vdash v_1 \sim_l v_2 : \mathsf{E}$ to denote that the values v_1 and v_2 both have type E and are indistinguishable at the security level l, and the notation $\Gamma \vdash \phi \sim_l \phi'$ to denote that the stores ϕ and ϕ' are indistinguishable at the security level l. In both cases, l denotes the maximum security level associated with the locations that an observer can read directly.

Definition 3. *We define the* indistinguishability of two values*, v_1 and v_2, with respect to a typing environment Γ and observation level l, denoted $\Gamma \vdash v_1 \sim_l v_2 : \mathsf{E}$, as the least symmetric relation closed under the following rules, where $\Gamma \vdash v_1, v_2 : \mathsf{E}$:*

- $\Gamma \vdash n_1 \sim_l n_2 : l'\ \mathsf{data}$ *iff* $(l \geq l') \rightarrow (n_1 = n_2)$
- $\Gamma \vdash n_1 \sim_l n_2 : l'\ \mathsf{key}$ *iff* $(l \geq l') \rightarrow (n_1 = n_2)$
- $\Gamma \vdash senc(u_k, u_m) \sim_l senc(u'_k, u'_m) : \mathsf{enc(E)}$ *iff* $(\Gamma \vdash u_m \sim_l u'_m : \mathsf{E}) \wedge (\Gamma \vdash junk(u_m) \sim_l u'_m : \mathsf{E} \vee \Gamma \vdash u_k \sim_l u'_k : \mathsf{lvl(E)\ key})$
- $\Gamma \vdash junk(u) \sim_l junk(u') : \mathsf{E}$
- $\Gamma \vdash junk(n) \sim_l n' : l'\ \mathsf{data}$ *iff* $(l < l')$
- $\Gamma \vdash junk(n) \sim_l n' : l'\ \mathsf{key}$ *iff* $(l < l')$
- $\Gamma \vdash junk(senc(u_k, u_m)) \sim_l senc(u'_k, u'_m) : \mathsf{enc(E)}$ *iff* $\Gamma \vdash junk(u_m) \sim_l u'_m : \mathsf{E}$

If a value has a type which permits it to be observed by the adversary, we must assume that this will eventually occur. It then follows that unencrypted data items which can be observed must be equal for them to be considered indistinguishable. Keys will be distinguishable if the output from their use is distinguishable. That is, by encrypting a known value with each key, decrypting each ciphertext with both keys, then comparing the final results to the original input: if any of the outputs are distinguishable from the input, then the two keys cannot be the same, and are thus distinguishable.

Ciphertexts are indistinguishable if their messages are indistinguishable, and the keys must also be indistinguishable if the observer could otherwise determine when one of the ciphertexts has been incorrectly decrypted. That is, if the keys have a type which allows them to encrypt observable data, then we must assume that the adversary is able to correctly decrypt each ciphertext, and can thus determine whether or not the required keys are the same whenever he can predict the correct output. It follows from the definition that keys which operate on non-observable data are indistinguishable.

Two junk values are indistinguishable, since they are both essentially just random bit-strings. For this reason also, junk names are distinguishable from observable non-junk names. Junk ciphertext is indistinguishable from non-junk ciphertext if the results of decrypting each one cannot be distinguished.

Definition 4. *We define the* indistinguishability of two stores*, ϕ_1 and ϕ_2, with respect to a typing environment Γ and observation level l, denoted $\Gamma \vdash \phi_1 \sim_l \phi_2$, as the least relation closed under the following rules:*

- $\Gamma \vdash \epsilon \sim_l \epsilon$
- $\Gamma \vdash$ *$(\phi, a \mapsto v) \sim_l (\phi', a \mapsto v')$ iff* $\Gamma \vdash \phi \sim_l \phi'$*,* $\Gamma \vdash v, v' : \mathsf{E}$ *and* $\Gamma \vdash v \sim_l v' : \mathsf{E}$

This definition states that two stores are indistinguishable if their domains are equal, and the values stored in equivalent locations are indistinguishable.

6 Non-interference

Informally, non-interference states that changes to non-observable inputs should have no effect on observable outputs. For expressions, this means that given two indistinguishable stores (which differ in the contents of at least one non-observable location), the final values obtained by fully evaluating the same expression with respect to those stores should be indistinguishable. For command sequences, this means that given two indistinguishable stores (which again differ in the contents of at least one non-observable location), the stores which result from fully evaluating the same command sequence with respect to those stores should also be indistinguishable.

Theorem 3. *Non-Interference*

i) *If* $(\Gamma, a : \mathsf{E\,loc}) \vdash e : \mathsf{E}'$*,* $\Gamma \vdash v_1, v_2 : \mathsf{E}$ *and* $\Gamma \vdash \Delta, \phi_1, \phi_2$*, such that* $\Gamma \vdash v_1 \sim_l v_2 : \mathsf{E}$ *and* $\Gamma \vdash \phi_1 \sim_l \phi_2$*, then it follows from* $e \rightarrow_{\Delta\phi'_1}^{*} v'_1$ *and* $e \rightarrow_{\Delta\phi'_2}^{*} v'_2$ *that* $\Gamma \vdash v'_1 \sim_l v'_2 : \mathsf{E}'$*, where* $\phi'_i \equiv \phi_i[a \mapsto v_i]$*.*

ii) *If* $(\Gamma, a : \mathsf{E\,loc}) \vdash c : \mathsf{C}$*,* $\Gamma \vdash v_1, v_2 : \mathsf{E}$ *and* $\Gamma \vdash \Delta, \phi_1, \phi_2$*, such that* $\Gamma \vdash v_1 \sim_l v_2 : \mathsf{E}$ *and* $\Gamma \vdash \phi_1 \sim_l \phi_2$*, then it follows from* $\langle c, \phi'_1 \rangle \rightarrow_{\Delta}^{*} \langle \epsilon, \phi''_1 \rangle$ *and* $\langle c, \phi'_2 \rangle \rightarrow_{\Delta}^{*} \langle \epsilon, \phi''_2 \rangle$ *that* $\Gamma \vdash \phi''_1 \sim_l \phi''_2$*, where* $\phi'_i \equiv \phi_i[a \mapsto v_i]$*.*

Proof. By induction on $(\Gamma, a : \mathsf{E\,loc}) \vdash e : \mathsf{E}'$ and $(\Gamma, a : \mathsf{E\,loc}) \vdash c : \mathsf{C}$:*(selected cases only)*

- Case T-Deref: $e : \mathsf{E}' \equiv\, !a' : \mathsf{E}'$ $a' : \mathsf{E}'\,\mathsf{loc}$ $!a' \rightarrow_{\Delta\phi'_i} \phi'_i(a')$. If $a' = a$, then $v'_i = v_i$ and $\mathsf{E}' \equiv \mathsf{E}$, thus the result follows from $\Gamma \vdash v_1 \sim_l v_2 : \mathsf{E}$. If $a' \neq a$, the result follows from $\Gamma \vdash \phi_1 \sim_l \phi_2$.
- Case T-SEnc: $e : \mathsf{E}' \equiv \mathrm{senc}(e_k, e_m) : \mathsf{enc}(\mathsf{E}'')$ $e_k : l'\,\mathsf{key}$ $e_m : \mathsf{E}''$ $\mathit{lvl}(\mathsf{E}'') = l'$
  $\mathrm{senc}(e_k, e_m) \rightarrow_{\Delta\phi'_1}^{*} \mathrm{senc}(v_k, v_m) \rightarrow_{\Delta\phi'_1}^{*} v'_1$ where $e_k \rightarrow_{\Delta\phi'_1}^{*} v_k$ and $e_m \rightarrow_{\Delta\phi'_1}^{*} v_m$.
  $\mathrm{senc}(e_k, e_m) \rightarrow_{\Delta\phi'_2}^{*} \mathrm{senc}(v'_k, v'_m) \rightarrow_{\Delta\phi'_2}^{*} v'_2$ where $e_k \rightarrow_{\Delta\phi'_2}^{*} v'_k$ and $e_m \rightarrow_{\Delta\phi'_2}^{*} v'_m$.
  It follows from Lemma 3 that $\Gamma \vdash v_k, v'_k : l'\,\mathsf{key}$ and $\Gamma \vdash v_m, v'_m : \mathsf{E}''$, by Lemma 2, $v_k \equiv n$ or $\mathrm{junk}(n)$, and $v'_k \equiv n'$ or $\mathrm{junk}(n')$, and by definition, $v_m \equiv u_m$ or $\mathrm{junk}(u_m)$, and $v'_m \equiv u'_m$ or $\mathrm{junk}(u'_m)$. If $v_k \equiv n$ and $v_m \equiv u_m$ then $v'_1 \equiv \mathrm{senc}(n, u_m)$ [A]; if $v_k \equiv \mathrm{junk}(n)$ and $v_m \equiv u_m$ then, by E-SEnc4, $v'_1 \equiv \mathrm{junk}(\mathrm{senc}(n, u_m))$ [B]; if $v_k \equiv n$ and $v_m \equiv \mathrm{junk}(u_m)$ then, by E-SEnc3, $v'_1 \equiv \mathrm{junk}(\mathrm{senc}(n, u_m))$ [C], and if $v_k \equiv \mathrm{junk}(n)$ and $v_m \equiv \mathrm{junk}(u_m)$ then, by E-SEnc4, E-Junk1, E-SEnc3 and E-Junk2, $v'_1 \equiv \mathrm{junk}(\mathrm{senc}(n, u_m))$ [D]. The equivalent outcomes for v'_2 are denoted by [E] through [H]. There are 16 cases for $\Gamma \vdash v'_1 \sim_l v'_2 : \mathsf{E}'$ which we need to consider (resulting from the cross product of [A,B,C,D] and [E,F,G,H]):

- Subcase [A]×[E]: $\Gamma \vdash \mathrm{senc}(n, u_m) \sim_l \mathrm{senc}(n', u'_m) : \mathsf{enc}(\mathsf{E}'')$
  By the induction hypothesis, $\Gamma \vdash n \sim_l n' : l'\,\mathsf{key}$ and $\Gamma \vdash u_m \sim_l u'_m : \mathsf{E}''$, and since $lvl(\mathsf{E}'') = l'$, the result follows immediately from Def. 3.
- Subcase [A]×[F]: $\Gamma \vdash \mathrm{senc}(n, u_m) \sim_l \mathrm{junk}(\mathrm{senc}(n', u'_m)) : \mathsf{enc}(\mathsf{E}'')$
  By the induction hypothesis, $\Gamma \vdash n \sim_l \mathrm{junk}(n') : l'\,\mathsf{key}$ thus, by Def. 3, $l < l'$. $\Gamma \vdash \mathrm{senc}(n, u_m) \sim_l \mathrm{junk}(\mathrm{senc}(n', u'_m)) : \mathsf{enc}(\mathsf{E}'')$ iff $\Gamma \vdash u_m \sim_l \mathrm{junk}(u'_m) : \mathsf{E}''$ and this holds when $l < lvl(\mathsf{E}'')$. The result then follows from $lvl(\mathsf{E}'') = l'$ and $l < l'$.
- Subcases [A]×[G,H]: $\Gamma \vdash \mathrm{senc}(n, u_m) \sim_l \mathrm{junk}(\mathrm{senc}(u'_k, u'_m)) : \mathsf{enc}(\mathsf{E}'')$
  By the induction hypothesis, $\Gamma \vdash u_m \sim_l \mathrm{junk}(u'_m) : \mathsf{E}''$, thus the result follows immediately from Def. 3.
- Subcases [B,C,D]×[F,G,H]: $\Gamma \vdash \mathrm{junk}(\mathrm{senc}(n, u_m)) \sim_l \mathrm{junk}(\mathrm{senc}(n', u'_m)) : \mathsf{enc}(\mathsf{E}'')$
  The result follows immediately from Def. 3.

Subcase [B]×[E] is similar to subcase [A]×[F] and subcases [C,D]×[E] are similar to subcases [A]×[G,H].

- Case T-SDec: $e : \mathsf{E}' \equiv \mathrm{sdec}(e_k, e_m) : \mathsf{E}' \quad e_k : l'\,\mathsf{key} \quad e_m : \mathsf{enc}(\mathsf{E}')$
  $lvl(\mathsf{E}') = l'$
  $\mathrm{sdec}(e_k, e_m) \rightarrow^{*}_{\vartriangle\phi'_1} \mathrm{sdec}(v_k, v_m) \rightarrow^{*}_{\vartriangle\phi'_1} v'_1$, where $e_k \rightarrow^{*}_{\vartriangle\phi'_1} v_k$ and $e_m \rightarrow^{*}_{\vartriangle\phi'_1} v_m$.
  $\mathrm{sdec}(e_k, e_m) \rightarrow^{*}_{\vartriangle\phi'_2} \mathrm{sdec}(v'_k, v'_m) \rightarrow^{*}_{\vartriangle\phi'_2} v'_2$, where $e_k \rightarrow^{*}_{\vartriangle\phi'_2} v'_k$ and $e_m \rightarrow^{*}_{\vartriangle\phi'_2} v'_m$.
  It follows from Lemma 3 that $\Gamma \vdash v_k, v'_k : l'\,\mathsf{key}$ and $\Gamma \vdash v_m, v'_m : \mathsf{enc}(\mathsf{E}')$, and by Lemma 2, $v_k \equiv n$ or $\mathrm{junk}(n)$, $v'_k \equiv n'$ or $\mathrm{junk}(n')$, $v_m \equiv \mathrm{senc}(u_a, u_b)$ or $\mathrm{junk}(\mathrm{senc}(u_a, u_b))$, and $v'_m \equiv \mathrm{senc}(u'_a, u'_b)$ or $\mathrm{junk}(\mathrm{senc}(u'_a, u'_b))$. If $v_k \equiv n$ and $v_m \equiv \mathrm{senc}(n, u_b)$ then, by E-SDec3, $v'_1 = u_b$ [A]; if $v_k \equiv n$ and $v_m \equiv \mathrm{senc}(u_a, u_b)$ where $u_a \neq n$ then, by E-SDec4, $v'_1 = \mathrm{junk}(u_b)$ [B]; if $v_k \equiv \mathrm{junk}(n)$ and $v_m \equiv \mathrm{senc}(u_a, u_b)$ then, by E-SDec5, $v'_1 = \mathrm{junk}(u_b)$ [C]; if $v_k \equiv n$ and $v_m \equiv \mathrm{junk}(\mathrm{senc}(u_a, u_b))$ then, by E-SDec6, $v'_1 = \mathrm{junk}(u_b)$ [D], and if $v_k \equiv \mathrm{junk}(n)$ and $v_m \equiv \mathrm{junk}(\mathrm{senc}(u_a, u_b))$ then, by E-SDec6, E-Junk1, E-SDec5 and E-Junk2, $v'_1 = \mathrm{junk}(u_b)$ [E]. The equivalent outcomes for v'_2 are denoted by [F] through [J]. There are 25 subcases for $\Gamma \vdash v'_1 \sim_l v'_2 : \mathsf{E}'$ which we need to consider (resulting from the cross product of [A,B,C,D,E] and [F,G,H,I,J]):
  - Subcase [A]×[F]: $\Gamma \vdash u_b \sim_l u'_b : \mathsf{E}'$
    By the induction hypothesis, $\Gamma \vdash \mathrm{senc}(n, u_b) \sim_l \mathrm{senc}(n', u'_b) : \mathsf{enc}(\mathsf{E}')$, therefore it follows from Def. 3 that $\Gamma \vdash u_b \sim_l u'_b : \mathsf{E}'$.
  - Subcase [A]×[G]: $\Gamma \vdash u_b \sim_l \mathrm{junk}(u'_b) : \mathsf{E}'$
    By the induction hypothesis, we have $\Gamma \vdash \mathrm{senc}(n, u_b) \sim_l \mathrm{senc}(u'_a, u'_b) : \mathsf{enc}(\mathsf{E}')$ and $\Gamma \vdash n \sim_l n' : l'\,\mathsf{key}$. By Def. 3, it follows from the second of these that $l < l'$ or $n = n'$. From the first one, it follows that $\Gamma \vdash u_b \sim_l u'_b : \mathsf{E}'$ as well as either $\Gamma \vdash n \sim_l u'_a : lvl(\mathsf{E}')\,\mathsf{key}$ or $\Gamma \vdash u_b \sim_l \mathrm{junk}(u'_b) : \mathsf{E}'$. In the latter case, the result is immediate. In the former case, it follows from Def. 3 that $l < lvl(\mathsf{E}')$ or $n = u'_a$. However, since $lvl(\mathsf{E}') = l'$, it must be the case that $l < lvl(\mathsf{E}')$, otherwise we would have $n = n' = u'_a$ which is prevented

by the requirement for [G] which states that $n' \neq u'_a$. The result then follows from Def. 3.

- Subcases [A]×[H,J]: $\Gamma \vdash u_b \sim_l \text{junk}(u'_b) : \mathsf{E}'$
  By the induction hypothesis, $\Gamma \vdash n \sim_l \text{junk}(n') : l'\,\mathsf{key}$, therefore it follows from Def. 3 that $l < l'$. Since $l' = \mathit{lvl}(\mathsf{E}')$, the result follows from Def. 3.
- Subcase [A]×[I]: $\Gamma \vdash u_b \sim_l \text{junk}(u'_b) : \mathsf{E}'$
  By the induction hypothesis, $\Gamma \vdash \text{senc}(n, u_b) \sim_l \text{junk}(\text{senc}(v'_a, u'_b)) : \mathsf{enc}(\mathsf{E}')$ and so it follows from Def. 3 that $\Gamma \vdash u_b \sim_l \text{junk}(u'_b) : \mathsf{E}'$.
- Subcases [B,C,D,E]×[G,H,I,J]: $\Gamma \vdash \text{junk}(u_b) \sim_l \text{junk}(u'_b) : \mathsf{E}'$
  The result follows immediately from Def. 3.

Subcase [B]×[F] is similar to subcase [A]×[G]; subcases [C,E]×[F] are similar to subcases [A]×[H,J], and subcase [D]×[F] is similar to subcase [A]×[I].

- Case T-Assign: $c : \mathsf{C} \equiv a' := e : \mathsf{cmd} \quad a' : \mathsf{E\,loc} \quad e : \mathsf{E}$
  $\langle a' := e, \phi'_1 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi'_1[a' \mapsto v] \rangle$ where $e \rightarrow_{\vartriangle \phi'_1}^* v$, and $\langle a' := e, \phi'_2 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi'_2[a' \mapsto v'] \rangle$ where $e \rightarrow_{\vartriangle \phi'_2}^* v'$. By the induction hypothesis for part (i), $\Gamma \vdash v \sim_l v' : \mathsf{E}$ and thus, in conjunction with $\Gamma \vdash \phi'_1 \sim_l \phi'_2$ and $\Gamma \vdash v_1 \sim_l v_2 : \Gamma$, the result follows from Def. 4.
- Case T-Cmds: $c : \mathsf{C} \equiv c_1 ; c_2 : \mathsf{cmd} \quad c_1 : \mathsf{cmd} \quad c_2 : \mathsf{cmd}$
  $\langle c_1 ; c_2, \phi'_1 \rangle \rightarrow_\vartriangle^* \langle \epsilon ; c_2, \phi'''_1 \rangle \rightarrow_\vartriangle \langle c_2, \phi'''_1 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi''_1 \rangle$ where $\langle c_1, \phi'_1 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi'''_1 \rangle$ and $\langle c_1 ; c_2, \phi'_2 \rangle \rightarrow_\vartriangle^* \langle \epsilon ; c_2, \phi'''_2 \rangle \rightarrow_\vartriangle \langle c_2, \phi'''_2 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi''_2 \rangle$ where $\langle c_1, \phi'_2 \rangle \rightarrow_\vartriangle^* \langle \epsilon, \phi'''_2 \rangle$. The result then follows by two applications of the induction hypothesis. □

Theorem 3 guarantees that well-typed expressions and command sequences are non-interferent. As an example, consider the following presentation of an API function for encrypting low security data stored in *msg_loc* with a key that is itself encrypted (by a master key, $\mathsf{k_m}$) and stored in a low security location, *ekey_loc*:[2]

$$\Gamma = \left\{ \begin{array}{c} \mathsf{k_m} : \top\,\mathsf{key}, \;\; \mathit{ekey\_loc} : \mathsf{enc}(\bot\,\mathsf{key})\,\mathsf{loc}, \\ \mathit{key\_loc} : \bot\,\mathsf{key\,loc}, \;\; \mathit{msg\_loc} : \bot\,\mathsf{data\,loc}, \;\; \mathit{res\_loc} : \mathsf{enc}(\bot\,\mathsf{data})\,\mathsf{loc} \end{array} \right\}$$

$$\mathit{key\_loc} := \text{sdec}(\mathsf{k_m}, !\mathit{ekey\_loc}) \;\; ; \;\; \mathit{res\_loc} := \text{senc}(!\mathit{key\_loc}, !\mathit{msg\_loc}) : \mathsf{cmd}$$

The non-interference theorem tells us that the above well-typed command sequence will not leak any information about the values of $\mathsf{k_m}$ and $!\mathit{key\_loc}$ into the low security locations.

7 Example: Wrap/Decrypt Attack

The wrap/decrypt attack ([11], Sec. 2.3) is one of the most basic attacks which a key management API can be susceptible to. In short, a sensitive key is altered in

[2] Recall that we treat all keys as high security values, and the security level associated with a key's type denotes the level of data that it may encrypt.

$$\dfrac{\dfrac{\begin{array}{c}[\Rightarrow \mathsf{E} \equiv \mathsf{enc}(\top\,\mathsf{key})]\\ x : \mathsf{enc}(\top\,\mathsf{key})\,\mathsf{loc} \in \Gamma\end{array}}{\Gamma \vdash x : \mathsf{E}\,\mathsf{loc}} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}[\Rightarrow l = \top]\\ \mathsf{k}_1 : \top\,\mathsf{key} \in \Gamma\end{array}}{\Gamma \vdash \mathsf{k}_1 : l\,\mathsf{key}} \quad \dfrac{\begin{array}{c}[\Rightarrow \mathsf{E}' \equiv \top\,\mathsf{key}]\\ \mathsf{k}_2 : \top\,\mathsf{key} \in \Gamma\end{array}}{\Gamma \vdash \mathsf{k}_2 : \mathsf{E}'} \quad \begin{array}{c}[\,\textsc{Holds}\,]\\ lvl(\mathsf{E}') = l\end{array}}{\Gamma \vdash \mathrm{senc}(\mathsf{k}_1, \mathsf{k}_2) : \mathsf{enc}(\mathsf{E}')} \quad \begin{array}{c}[\,\textsc{Holds}\,]\\ \mathsf{enc}(\mathsf{E}') <: \mathsf{E}\end{array}}{\Gamma \vdash \mathrm{senc}(\mathsf{k}_1, \mathsf{k}_2) : \mathsf{E}}}{\Gamma \vdash x \mathrel{:=} \mathrm{senc}(\mathsf{k}_1, \mathsf{k}_2) : \mathsf{cmd}}$$

Fig. 5. Successful typing derivation for the wrap command

$$\dfrac{\dfrac{\begin{array}{c}[\Rightarrow \mathsf{E} \equiv \bot\,\mathsf{data}]\\ y : \bot\,\mathsf{data}\,\mathsf{loc} \in \Gamma\end{array}}{\Gamma \vdash y : \mathsf{E}\,\mathsf{loc}} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}[\Rightarrow l = \top]\\ \mathsf{k}_1 : \top\,\mathsf{key} \in \Gamma\end{array}}{\Gamma \vdash \mathsf{k}_1 : l\,\mathsf{key}} \quad \dfrac{\begin{array}{c}[\Rightarrow \mathsf{E}' \equiv \top\,\mathsf{key}]\\ \dfrac{x : \mathsf{enc}(\top\,\mathsf{key})\,\mathsf{loc} \in \Gamma}{\Gamma \vdash x : \mathsf{enc}(\top\,\mathsf{key})\,\mathsf{loc}}\end{array}}{\Gamma \vdash\, !x : \mathsf{enc}(\mathsf{E}')} \quad \begin{array}{c}[\,\textsc{Holds}\,]\\ lvl(\mathsf{E}') = l\end{array}}{\Gamma \vdash \mathrm{sdec}(\mathsf{k}_1, !x) : \mathsf{E}'} \quad \begin{array}{c}\left[\begin{array}{c}\textsc{Does}\\ \textsc{Not}\\ \textsc{Hold}\end{array}\right]\\ \mathsf{E}' <: \mathsf{E}\end{array}}{\Gamma \vdash \mathrm{sdec}(\mathsf{k}_1, !x) : \mathsf{E}}}{\Gamma \vdash y \mathrel{:=} \mathrm{sdec}(\mathsf{k}_1, !x) : \mathsf{cmd}}$$

Fig. 6. Failed typing derivation for the decrypt command

such a way as to be able to wrap (encrypt) other sensitive keys and also decrypt public data. This typically involves altering the key's 'type' so that it is accepted by each of the two required API functions. Alternatively, two copies of the key can be obtained such that each copy has one of the two necessary types. Both of these requirements can be quite straightforward to achieve (e.g., as discussed in [7]). The outcome is that a sensitive key can be discovered by first wrapping it, then decrypting the result:

$$\begin{array}{ll} x \mathrel{:=} \mathrm{senc}(\mathsf{k}_1, \mathsf{k}_2) & \ldots\text{'wrap' } \mathsf{k}_2 \text{ with } \mathsf{k}_1 \\ y \mathrel{:=} \mathrm{sdec}(\mathsf{k}_1, !x) & \ldots\text{recover } \mathsf{k}_2 \end{array}$$

Our type system can be applied to these commands as follows:

$$\Gamma = \left\{\begin{array}{c}\mathsf{k}_1 : \top\,\mathsf{key},\ \mathsf{k}_2 : \top\,\mathsf{key},\\ x : \mathsf{enc}(\top\,\mathsf{key})\,\mathsf{loc},\ y : \bot\,\mathsf{data}\,\mathsf{loc}\end{array}\right\} \qquad \begin{array}{l} x \mathrel{:=} \mathrm{senc}(\mathsf{k}_1, \mathsf{k}_2) : \mathsf{cmd}\\ y \mathrel{:=} \mathrm{sdec}(\mathsf{k}_1, !x) : \mathsf{cmd}\end{array}$$

Figure 5 shows the typing derivation for the wrap command, and Fig. 6 shows the typing derivation for the decrypt command (unnecessary instances of the T-Sub rule have been omitted in both cases).

The first command type-checks, since $lvl(\mathsf{E}') = l$ and $\mathsf{enc}(\mathsf{E}') <: \mathsf{E}$ both hold, but the second command does not, since $\mathsf{E}' <: \mathsf{E}$ does not hold. The flaw is that a sensitive piece of data is written to a public location — the failed subtype condition indicates that the security level of the data is greater than that of the location. Note that using the definition $x : \mathsf{enc}(\bot\,\mathsf{data})\,\mathsf{loc}$ instead of $x : \mathsf{enc}(\top\,\mathsf{key})\,\mathsf{loc}$ in the above example makes the second command type-check,

$$\dfrac{\dfrac{\begin{array}{c}[\Rightarrow \mathsf{E}' \equiv l'\,\mathsf{key}]\\ \mathit{res} : l'\,\mathsf{key\ loc} \in \Gamma\end{array}}{\Gamma \vdash \mathit{res} : \mathsf{E}'\,\mathsf{loc}} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}[\Rightarrow l'' = \top]\\ \mathit{key} : \top\,\mathsf{key} \in \Gamma\end{array}}{\Gamma \vdash \mathit{key} : l''\,\mathsf{key}} \quad \dfrac{\begin{array}{c}[\Rightarrow \mathsf{E} \equiv l\,\mathsf{key}]\\ \mathit{wkey} : \mathsf{enc}(l\,\mathsf{key}) \in \Gamma\end{array}}{\Gamma \vdash \mathit{wkey} : \mathsf{enc}(\mathsf{E})} \quad \begin{array}{c}[\ \textsc{Holds}\ ]\\ \\ \mathit{lvl}(\mathsf{E}) = l''\end{array}}{\Gamma \vdash \mathrm{sdec}(\mathit{key}, \mathit{wkey}) : \mathsf{E}} \qquad \begin{array}{c}\left[\begin{array}{c}\textsc{May}\\ \textsc{Not}\\ \textsc{Hold}\end{array}\right]\\ \mathsf{E} <: \mathsf{E}'\end{array}}{\Gamma \vdash \mathrm{sdec}(\mathit{key}, \mathit{wkey}) : \mathsf{E}'}}{\Gamma \vdash \mathit{res} := \mathrm{sdec}(\mathit{key}, \mathit{wkey}) : \mathsf{cmd}}$$

Fig. 7. The unwrap command is only secure when $l\,\mathsf{key} <: l'\,\mathsf{key}$ (i.e., when $l = l'$)

but it prevents the first command from type-checking, since $\mathsf{enc}(\mathsf{E}') <: \mathsf{E}$ no longer holds.

The wrap/decrypt attack is one of a number of attacks which initially require the type of a key to be altered, therefore our type system should be able to identify when an API command may allow this to occur. One such command is 'unwrap', which takes an existing key and ciphertext corresponding to a second key encrypted under the first one, and then decrypts the ciphertext before storing the result. Figure 7 shows that our type system is indeed able to identify that the following instantiation of that command is insecure:

$$\Gamma = \left\{\begin{array}{c}\mathit{key} : \top\,\mathsf{key},\\ \mathit{wkey} : \mathsf{enc}(l\,\mathsf{key}),\ \mathit{res} : l'\,\mathsf{key\ loc}\end{array}\right\} \qquad \mathit{res} := \mathrm{sdec}(\mathit{key}, \mathit{wkey}) : \mathsf{cmd}$$

Since the security level associated with the type of a key restricts what that key can be used to encrypt and decrypt, and the instantiation of the 'unwrap' command given above allows this level to be changed (i.e., when $l \neq l'$), then it is clearly insecure. This particular flaw can be prevented in practice by including usage information for the key within the ciphertext, thereby making it possible to carry out a check which is equivalent to ensuring that l and l' are equal. However, it is then necessary to ensure that no API command allows this usage information to be modified unintentionally.

8 Related Work

Vaughan and Zdancewic [12] give a security typed language in which valid programs are guaranteed to be non-interfering; a result which is achieved via a combination of static and dynamic checks. However, they require that encrypted messages adhere to a strict format which prevents their system from being used to analyse many existing security APIs.

Laud [13] presents a weakened variant of non-interference termed 'computational independence', using static analysis to track dependencies between variables. Security is guaranteed when the public outputs are computationally independent from all of the sensitive inputs. Encryption is probabilistic and assumed to be secure with respect to a polynomially-bounded adversary. Key cycles are permitted, as the rules will identify the resulting cyclic dependencies.

Focardi and Centenaro [14] give a type system for enforcing non-interference in multi-threaded distributed programs which share common memory locations. They use *confounders* (unique values associated with each new ciphertext) as an abstraction of probabilistic encryption, and give a definition of equivalence for low security values based on the notion of *patterns* [10]. If the confounder is uniquely determined by the message and key, then their definition of indistinguishability for ciphertexts is equivalent to the one given in this paper. Their definition for memories is stronger than our one since we do not distinguish between copies of the same ciphertext and different ciphertexts created from the same key and message (doing so is only necessary when considering conditionals). However, because they deal with distributed systems where restrictions on key usage cannot be enforced, they do not associate a secondary security level with cryptographic keys which means that if a high security key is used to encrypt some low security data, the result of the subsequent decryption is forced to be high.

Bengtson et al. [15] have developed an extended typechecker for F# code that is annotated with *refinement types.* A refinement type includes a logical formula which places restrictions upon the associated term. They consider an active adversary and use a generalised version of the symbolic cryptography model. The focus of their research is on authentication and authorisation properties for security protocols, but the flexibility afforded by refinement types means that the technique may be applicable to related domains such as security API analysis. However, due to the different target domain, the underlying type system that Bengtson et al. employ is quite different from the one which we give in this paper.

9 Conclusions and Future Work

Using typing rules for analysing the security properties of cryptographic systems is not new (e.g., [16]), but it is common for restrictions to be placed upon the use of encryption and decryption, as well as on any keys involved. Consequently, certain security APIs cannot be analysed using some of these existing systems. For example, the IBM 4758 [3] has one internal master key that is used to encrypt all other keys which are then stored on the attached host, therefore rule sets in which the result of a decryption cannot be used as a key (e.g., [17]) are unable to analyse the security API for that device.

In this paper, we have presented the foundations of a type system that is designed to deal with common features of security APIs such as encrypted keys and nested encryptions. We gave a definition of indistinguishability which captures the potential for an adversary to determine that the keys used in two ciphertexts are different, even though their actual values remain unknown. We then proved that well-typed command sequences are non-interferent with respect to this definition. We also proved the type-safety of our system meaning that the type information can be ignored at run-time.

The next stage of our research is to extend our type system to include additional features present in Volpano, Smith and Irvine's original type system [1] and Volpano and Smith's extension [18] — specifically procedures, primitive operations and conditional statements. This will allow us to analyse more accurate representations of functions in widely used security APIs such as PKCS #11 [20]. Adding conditionals will require a modified definition of the indistinguishability of stores, similar to the one given by Focardi and Centenaro [14]. It should be noted that such a change will not affect our results for the indistinguishability of expressions.

Further ahead, we plan to extend our type system to deal with data integrity, since this is equally as important as data confidentiality for key management APIs, as well as permitting explicit declassification thus allowing our system to analyse an additional class of security APIs.

References

1. Volpano, D.M., Smith, G., Irvine, C.E.: A Sound Type System for Secure Flow Analysis. Journal of Computer Security 4(3), 167–187 (1996)
2. Sumii, E., Pierce, B.C.: Logical Relations for Encryption. In: Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), pp. 256–269. IEEE Computer Society Press, Los Alamitos (2001)
3. IBM 4758 PCI Cryptographic Coprocessor, `http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml`
4. nCipher nShield Hardware Security Module, `http://www.ncipher.com/en/Products/Hardware%20Security%20Modules/nShield.aspx`
5. Cortier, V., Keighren, G., Steel, G.: Automatic Analysis of the Security of XOR-Based Key Management Schemes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 538–552. Springer, Heidelberg (2007)
6. Courant, J., Monin, J.F.: Defending the Bank with a Proof Assistant. In: Proceedings of the 6th International Workshop on Issues in the Theory of Security (WITS 2006), pp. 87–98 (2006)
7. Delaune, S., Kremer, S., Steel, G.: Formal Analysis of PKCS #11. In: [19], pp. 331–344
8. Youn, P.: The Analysis of Cryptographic APIs using the Theorem Prover Otter. Master's thesis, Massachusetts Institute of Technology (May 2004)
9. Youn, P., Adida, B., Bond, M.K., Clulow, J., Herzog, J., Lin, A., Rivest, R.L., Anderson, R.J.: Robbing the Bank with a Theorem Prover. Technical Report 644, University of Cambridge Computer Laboratory (August 2005)
10. Abadi, M., Rogaway, P.: Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 3–22. Springer, Heidelberg (2000)
11. Clulow, J.S.: On the Security of PKCS #11. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 411–425. Springer, Heidelberg (2003)
12. Vaughan, J.A., Zdancewic, S.: A Cryptographic Decentralized Label Model. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy, pp. 192–206. IEEE Computer Society Press, Los Alamitos (2007)
13. Laud, P.: Handling Encryption in an Analysis for Secure Information Flow. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 159–173. Springer, Heidelberg (2003)

14. Focardi, R., Centenaro, M.: Information Flow Security of Multi-threaded Distributed Programs. In: Proceedings of the 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS 2008), pp. 113–124. ACM Press, New York (2008)
15. Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A.D., Maffeis, S.: Refinement Types for Secure Implementations. In: [19], pp. 17–32
16. Abadi, M.: Secrecy by Typing in Security Protocols. In: Ito, T., Abadi, M. (eds.) TACS 1997. LNCS, vol. 1281, pp. 611–638. Springer, Heidelberg (1997)
17. Laud, P., Vene, V.: A Type System for Computationally Secure Information Flow. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 365–377. Springer, Heidelberg (2005)
18. Volpano, D.M., Smith, G.: A Type-Based Approach to Program Security. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 607–621. Springer, Heidelberg (1997)
19. Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008). IEEE Computer Society Press, Los Alamitos (June 2008)
20. PKCS #11: Cryptographic Token Interface Standard, `http://www.rsa.com/rsalabs/node.asp?id=2133`

Separating Trace Mapping and Reactive Simulatability Soundness: The Case of Adaptive Corruption*

Laurent Mazaré[1] and Bogdan Warinschi[2]

[1] LexiFI S.A.S.
laurent.mazare@polytechnique.org
[2] University of Bristol Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol BS8 1UB, United Kingdom
bogdan@cs.bris.ac.uk

Abstract. Computational soundness is the research direction that aims to translate security guarantees with respect to Dolev-Yao models into guarantees with resepect to the stronger computational models of modern cryptography. There are essentially two different approaches that aim to achieve computational soundness. One approach is based on the so-called trace mapping theorems, and one based on reactive simulatability. In a recent paper, Backes, Dürthmuth, and Küsters have shown that the stronger requirements needed for reactive simulatability-based soundness imply that a trace mapping theorem also holds. It was left as an open problem whether there exists interesting settings where the simulatability framework breaks down but mapping theorems still exist.

In this paper we describe one such setting, and thus give a separation between the two frameworks. Specifically, we show that adaptive corruption of symmetric encryption keys (a problematic setting for simulation-based frameworks) can be smoothly treated in a mapping theorem-based soundness framework.

A crucial ingredient of our proof, and a result of independent interest, is a new (indistinguishability based) security notion for encryption. The central feature of our definition is that in addition to standard chosen-ciphertext attacks in multi-user settings, it also directly accounts for adaptive corruption of decryption keys. We show that our notion satisfies the intuitively appealing property that it is equivalent to standard security requirements on encryption.

1 Introduction

Computational soundness is an important research direction in security that aims to justify the high abstraction used by Dolev-Yao models with respect to the more concrete computational ones. Two frameworks for computational

* This work was partially funded by the European Commission through the ICT programme under Contract ICT-2007-216646 ECRYPT II.

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 193–210, 2009.

soundness have been developed in the past few years: the trace mapping-based and the reactive simulatability-based framework. Each of these frameworks establishes a connection between concrete and symbolic executions of protocols. In turn these connections allow for security properties proved in the more abstract models to be transfered to the more concrete ones.

The *reactive simulatability* approach (developed by Backes, Pfitzmann, and Waidner [BPW03a, BPW03b, BP04]) uses the well-established notion of simulation. Under this paradigm, one shows that concrete implementations can replace idealized (symbolic) versions of cryptographic primitives inconspicuously, *i.e.* without introducing any additional attacks. Such results are established through a *simulator* which, interacting solely with the abstract version of the primitives, offers lexactly the same behavior to an external arbitrary environment (in particular, a protocol executed in the presence of an adversary) as a concrete (secure) implementation. In the case of encryption, to emulate the ciphertexts created by honest parties, the simulator usually encrypts some arbitrary, randomly selected string of appropriate length. The idea is that if encryption is indeed secure, an external environment would not be able to notice the difference between simulated and honest executions.

The *trace mapping* approach (Micciancio and Warinschi [MW04]) is based on a weaker, thus easier to establish link between symbolic and concrete executions. Typically, one shows that for any protocol in a certain class, any concrete attack produces an execution trace which is the image of a trace produced by a symbolic attacker, unless the concrete attacker breaks one of the primitives used in the execution of the protocol.

Reactive simulatability imposes a stricter relation between computational and symbolic models than the trace mapping approach. Unsurprisingly, the implementation requirements needed to enforce a reactive simulatability relation are more demanding, allow for less flexibility than those for trace mapping, and require more complex symbolic models. On the flip side, reactive simulatability comes with "built-in" compositionality results that are missing for the trace mapping approach. Therefore, understanding the trade-off offered by the two frameworks is important: it allows choosing between simpler vs. complex security models, symbolic proofs, and implementations with an eye on the desired security guarantees.

An important step was recently taken by Backes, Dürmuth, and Küsters who established the first formal relation between the two frameworks [BDK07]. They show that a reactive simulatability framework for computational soundness also implies a trace mapping theorem. In this paper we further clarify the relation between the two frameworks. We identify an important and interesting setting where the tools and techniques of reactive simulatability fail, but a trace mapping theorem can still be proved. We discuss the details of our results and their significance next.

Adaptive corruption and trace mapping soundness. Technically, our main result is a computational soundness setting based on trace mapping. The novelty of the setting consists in the setting that we use: in addition to standard attacks, the adversary may adaptively decide to corrupt the symmetric keys of honest parties even *after* those keys had been used to produce encryptions.

Our trace mapping soundness theorem holds for a restricted class of protocols: we do not allow for encryption keys to be sent encrypted (they can be still sent in clear), and we do not allow ciphertext forwarding. These restrictions prevent the adversary from obtaining key-dependent encryptions (an issue we do not address in this paper) and thus in particular avoids the occurrence of encryption cycles (and their associated difficulties) [AR00, BRS03, BPS07]. Nevertheless, the class of protocols that we consider is still quite large.

More important than our technical result is its conceptual importance. A soundness result via reactive simulatability for adaptive corruption of keys seems very difficult and perhaps impossible. Indeed, an environment can easily be used to mimic the game that defines security of encryption under selective decryption attacks, a known difficult problem [DNRS03], and thus soundness would seem to require solving the selective decryption problem.

More problematic is that in simulation based settings, adaptive corruption of keys raises the so-called *commitment problem.* Recall that in such settings the simulator has to produce a simulation of the computational execution of the protocol. In doing so however, it only has access to the idealized execution. In particular, it may have to produce encryption of values that it does not apriorily know. As sketched above, in such situations the simulator may simply encrypt a random string of appropriate length. An attacker would not be able to observe the difference, unless the encryption scheme is not secure. The problem raised by adaptive corruption of keys is now clear: an adversary that corrupts a key that had been used to encrypt "garbage" would be able to tell the difference between real and simulated executions. This problem with adaptive corruption had been formally captured by Nielsen who proves that in such settings, secure systems require keys that are as long as the data to be encrypted [Nie02]. Therefore, our soundness result also entails the first separation result between trace mapping and reactive simulatability based soundness.

Security of encryption against adaptive corruption. The various security notions for encryption developed so far, capture the idea that ciphertexts reveal no partial information about the plaintext under a range of possible attacks [NY90, RS92, DDN00, BBM00]. The tacit hope underlying existent definitions is that the strongest such notion (generally accepted to be IND-CCA) captures security under *all* possible attacks that encryption should withstand when used in higher-level protocols. Unfortunately this is not the case. Existent definitions do not explicitly consider the possibility that an adversary can obtain some of the keys of the honest parties (through corruption, or possibly through legitimate means).

A second result of this paper is a novel security definition for encryption that overcomes the above problem. The central feature of our definition is that in addition to standard chosen-ciphertext attacks in multi-user settings, it also directly accounts for corruption of decryption keys. Importantly, the corruption model that we consider is adaptive: the adversary can choose which keys to corrupt *after* seeing ciphertexts under those keys. We call the resulting notions indistinguishability under chosen-plaintext (respectively, chosen-ciphertext) attacks with adaptive corruptions IND-ACCPA (respectively, IND-ACCCA) security, in short. Our definitions are given for symmetric encryption, but can be adapted for the asymmetric case straightforwardly.

Finally, we relate our notion to standard security notions for encryption. We prove the intuitively appealing property that the security of plaintexts encrypted under uncorrupted keys is not affected by corruption of other keys (as long as the corrupted keys do not encrypt related plaintexts). More formally, we prove via a tight reduction that an IND-CPA (respectively IND-CCA) secure encryption scheme is also IND-ACCPA (respectively IND-ACCCA) secure. The proof uses a subtle modification of the hybrid argument used in [BBM00].

Our new definition is a crucial ingredient in the soundness result described above. Furthermore, we expect the definition to have further applications to security proofs of larger protocols that use encryption. To clarify what we mean by this, consider some arbitrary protocol that uses encryption and imagine the following security proof by reduction. (Similar issues occur in our soundness result, so the discussion bellow is also relevant there.) Given some adversary $\mathcal{A}$ against the protocol, one constructs an adversary $\mathcal{B}$ against the encryption. The latter adversary uses access to the encryption/decryption oracles in the game defining the security of encryption to simulate the execution of the protocol for $\mathcal{A}$. During this simulation $\mathcal{B}$ needs to handle the corruption requests of $\mathcal{A}$. In the case when corruption is static (i.e. $\mathcal{A}$ decides whom to corrupt priorly to the actual execution), adversary $\mathcal{B}$ can generate the keys for all corrupt parties on its own and hand them over as responses to $\mathcal{A}$'s corrupt requests. For the simulation of the remaining honest parties $\mathcal{B}$ uses its oracles.

The case of adaptive corruption is significantly more complicated since there is no way for $\mathcal{B}$ to a priori determine which subset of keys will be corrupted by the adversary. A simple minded simulation where $\mathcal{B}$ guesses this subset succeeds only with negligible probability, so some more elaborate techniques should be employed.[1] It is clear that the simulation sketched can benefit greatly from a security game for encryption that provides $\mathcal{B}$ with the ability to corrupt keys – that part of the simulation would then become trivial – and this is precisely what our new definition offers. Our definition and equivalence result offers thus the following proof methodology: show the security via a reduction to IND-ACCPA security of the underlying encryption scheme (the simulation now can take advantage of the key corruption capabilities) and then conclude, via our equivalence proof, security of the system based on IND-CCA security. In essence, this methodology avoids repeating the hybrid argument for the equivalence between IND-CCA and IND-ACCCA (which had been done once and for all).

[1] Notice that in this discussion we assume that the reduction is to the security of encryption defined in the multi-user setting. A reduction to the security of a single key immediately lands into trouble if the adversary $\mathcal{B}$ needs to encrypt the same message under the challenge key, and under the key of some other honest party.

2 Related Work

Computational soundness. Security models for encryption that enable computationally sound symbolic analysis were recently analyzed by Backes, Pfitzmann, and Scedrov [BPS07] in the general setting of a universally composable library [BPW03b, BP04]. The attacker model that they develop combines key-related messages with dynamic corruption of keys and leads to a notion that is incomparable to ours: under their attack model the attacker can not corrupt a key after it had been used. The resulting soundness results are also different since they concern incomparable classes of protocols: unlike [BPS07] we allow adaptive corruption of keys, but use some restrictions that are not needed in that work.

Common and Cortier [CCL08] include symmetric encryption in a process calculus for security for which they prove computational soundness of an associated process equivalence. Although they do allow for nested encryption, adaptive corruption is forbidden in their calculus. A soundness result for symmetric encryption had previously been obtained by Laud [Lau04]. The framework that he considers can cope with protocols that send symmetric encryption keys after they had been used, but does not permit adaptive corruption of such keys. Furthermore, his results do not use a mapping lemma technique and thus do not yield a separation result akin to the one here.

The issue of adaptive corruption in computational soundness had also been addressed by Gupta and Shmatikov [GS06]. Their result however does not consider fully adaptive corruptions since corruption must occur prior to parties actually using their encryption key. Our security notion for encryption, and our computational soundness result, permit corruptions based on the values of ciphertexts under the secret encryption key.

Adaptively secure encryption. The security of encryption in settings where the adversary can obtain adaptively some of the decryption keys has been recognized early on as a very important cryptographic issue, and has since generated quite a bit of research. The main difference between the definition that we developed and those used in prior research on the subject is that we use indistinguishability, game-based definitions, while the latter follow simulatability-based formulations. Although quite popular due to the strength and uniformity of the resulting definitions as well as their intuitive appeal, simulation-based definitions may lead to notions which are unsatisfiable [DNRS03, Nie02].

A lot of research has been devoted to circumventing the difficulties associated to this adaptive corruption in simulation-based settings. Beaver and Haber propose a solution that uses interaction and memory erasure [BH92]. Canetti et. al. [CFGN96], followed by Beaver [Bea97], and Damgard and Nielsen [DN00] develop the concept of non-committing encryption (roughly, a ciphertext may be decrypted to arbitrary plaintexts, for appropriately chosen keys) which circumvents the commitment problem. Finally, a more recent proposal of Canetti, Halevi, and Katz [CHK05] shows that adaptive security of encryption schemes can be achieved if the decryption key is allowed to somehow evolve over time.

Given that the security requirements on encryption under simulation based definitions are stronger than those captured by our definition, and in light of the negative result in [Nie02], it should come as no surprise that the solutions outlined above, are all rather inefficient, and do not follow from standard assumptions on encryption. In contrast, our game-based notion imposes less stringent requirements on the security of encryption and can be shown to be a consequence of existing requirements on encryption. Clearly, we make no claim that we deal with adaptive security of encryption in its full generality, since the results of [Nie02] apply. We do claim however that our definition captures a reasonable level of security which still suffices for many applications where security is not defined via simulations. In some sense the situation resembles that of commitment schemes. While it is widely known that universally composable commitments do not exist [CF01], other, perfectly reasonable game-based security definitions for commitments can both be achieved and used in a wide range of applications [Gol99].

More recently, Backes, Pfitzmann, and Scedrov proposed a security definition for encryption based on indistinguishability in which they consider adversaries who can both observe key-dependent ciphertexts, and perform corruption of secret keys [BPS07]. Our definition does not consider the key-dependent message capabilities. Also, the corruption model there seems to be weaker, as it does not permit an adversary to corrupt keys which had been used to encrypt.

3 Adaptive Corruption in Indistinguishability-Based Security of Encryption

In this section we give and motivate our definition for the security of encryption when the adversary can adaptively corrupt keys.

Notation and preliminaries. We write $[k]$ for the set $\{1, 2, \ldots, k\}$. If $\mathcal{A}$ is a probabilistic algorithm we write $x \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\cdots}(\mathsf{input})$ for the result of running algorithm $\mathcal{A}$ on input input, with access to oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots$, all initialized with fresh coins.

We recall the standard definition for symmetric encryption schemes. Let $\mathcal{SE}$ be a symmetric encryption scheme defined by three algorithms $\mathcal{KG}$, $\mathcal{E}$ and $\mathcal{D}$. The key generation algorithm $\mathcal{KG}$ takes as input a security parameter η and outputs a key k. We write $k \xleftarrow{\$} \mathcal{KG}(\eta)$ for the process of generating key k. The encryption algorithm $\mathcal{E}$ is randomized. It takes as input a bit-string m and a key k and returns the encryption of m using k. We write $c \xleftarrow{\$} \mathcal{E}(k, m)$ for the process of producing the encryption c of m under k. Finally, the decryption algorithm $\mathcal{D}$ takes as input a bit-string c representing a ciphertext and a key k and outputs the corresponding plain-text. It is required that $\mathcal{D}(k, c) = m$.

Next, we recall the standard notion of $\mathsf{IND\text{-}CCA}$ security for symmetric encryption schemes. Let $\mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme.

Let $b \in \{0, 1\}$ and $\eta \in \mathbb{N}$ a security parameter. Let $\mathcal{A}$ be an adversary that has access to a left-right encryption oracle $\mathcal{E}(k, \mathcal{LR}(\cdot, \cdot, b))$, parametrized by a

```
Experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}b}}_{\mathcal{SE},\mathcal{A}}(\eta)$
    $k \xleftarrow{\$} \mathcal{KG}(\eta)$
    $d \xleftarrow{\$} A^{\mathcal{E}(k,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}(k,\cdot)}(\eta)$
    Return d
```

```
Experiment $\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}}b}_{\mathcal{A},\mathcal{SE},p}(\eta)$
    For $i = 1$ to $p(\eta)$ do $k_i \leftarrow \mathcal{KG}(\eta)$
    $\mathsf{keys} = (k_1, k_2, \ldots, k_{p(\eta)})$
    $d \xleftarrow{\$} A^{\mathcal{E}(\mathsf{keys},\cdot),\mathcal{E}(\mathsf{keys},\mathcal{LR}(\cdot,\cdot,b)),\mathcal{C}(\mathsf{keys}),\mathcal{D}(\mathsf{keys},\cdot)}(\eta)$
    Return $d$
```

Fig. 1. Experiments for defining IND-CCA and IND-ACCCA security of symmetric encryption scheme $\mathcal{SE}$. In the IND-CCA experiment the adversary is not allowed to query the decryption oracle with a ciphertext it obtains from its encryption oracle. In the IND-ACCCA experiment, the adversary is not allowed to query the decryption oracle with (i, c) if c was the result of an encryption query using key k_i. CPA versions of the above experiments are obtained by removing the decryption oracle $\mathcal{D}$.

bit b and keyed with key k. The oracle accepts as input pairs of equal-length messages (m_0, m_1) and returns the encryption of m_b. In addition, the adversary also has access to a decryption oracle $\mathcal{D}(k,\cdot)$ which on a query c returns $\mathcal{D}(k, c)$. We give the experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}b}}_{\mathcal{SE},\mathcal{A}}(\eta)$ for IND-CCA security in Figure 1.

We define the advantage of $\mathcal{A}$ against IND-CCA security of $\mathcal{SE}$ by:

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{SE},\mathcal{A}}(\eta) = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}1}}_{\mathcal{SE},\mathcal{A}}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}0}}_{\mathcal{SE},\mathcal{A}}(\eta) = 1\right]$$

We say that the encryption scheme $\mathcal{SE}$ is IND-CCA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{SE},\mathcal{A}}(\eta)$ is a negligible function of η.

3.1 Security of Encryption with Adaptive Corruption

Our starting point is the security definition that Bellare, Boldyreva, and Micali give for the security of encryption in a multi-user setting [BBM00]. We adapt their definition from the asymmetric to the symmetric case, and extend it to capture key corruptions.

In their security model, an adversary has access to polynomially many left-right encryption oracles, parametrized by the same hidden bit b, but which use independently generated encryption keys. The adversary also has access to corresponding decryption oracles. Importantly, this setup allows the adversary to see encryptions of the same message under different keys. The goal of the adversary is to determine b.

In addition to the standard attacks captured as above, we would like to allow the adversary to corrupt some of the encryption keys. One possibility is to keep the same setup, and let the adversary corrupt some of the keys used by the left-right oracles, as long as the adversary makes no queries to those oracles (since otherwise, such queries combined with key corruption lead to trivial attacks). The resulting notion would not be strong enough as the adversary cannot decide to corrupt a key once that key had been used to encrypt. Our solution is to provide the adversary with access to standard encryption oracles under the keys used by the left-right oracles which the adversary can query at will. The restriction that we impose is that ciphertexts returned by a left-right oracle are not sent to the corresponding decryption oracle.

We next formally define our new security notions. We use several oracles, which use as parameter an ordered list of keys $\mathsf{keys} = (k_1, k_2, \ldots)$. Oracle $\mathcal{E}(\mathsf{keys}, \cdot)$ allows the adversary to see encryptions of any message under the keys in keys. It accepts queries of the form $(i, m) \in (\mathbb{N} \times \{0,1\}^*)$ and returns an encryption $\mathcal{E}(k_i, m)$ if i is a valid index in the list keys. Oracle $\mathcal{E}(\mathsf{keys}, \mathcal{LR}(\cdot, \cdot, b))$ is a left-right encryption oracle parametrized by keys and a bit b. When it receives a query of the form $(i, m_0, m_1) \in (\mathbb{N} \times \{0,1\}^* \times \{0,1\}^*)$ this oracle returns an encryption $\mathcal{E}(k_i, m_b)$ if i is a valid index in set of keys, and m_0 and m_1 have the same length. The corruption oracle $\mathcal{C}(\mathsf{keys})$ gets as input an index i and returns the key k_i if the key is defined. In the chosen-ciphertext attack version, the adversary also has access to a decryption oracle $O_{\mathcal{D}}$ which on input a pair $(i, c) \in \mathbb{N} \times \{0,1\}^*$ returns $\mathcal{D}(k, c)$. The formal experiment for defining IND-ACCCA security is given in Figure 1. As before, the experiment for IND-ACCPA is obtained by removing access to the decryption oracle.

Definition 1. *For an adversary $\mathcal{A}$ against $\mathcal{SE}$ in the* IND-ACCCA *experiment in Figure 1 we define its advantage by:*

$$\mathbf{Adv}^{\mathsf{ACCCA}}_{\mathcal{A},\mathcal{SE},p}(\eta) = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}1}}_{\mathcal{A},\mathcal{SE},p}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}0}}_{\mathcal{A},\mathcal{SE},p}(\eta) = 1\right]$$

We say that scheme $\mathcal{SE}$ is IND-ACCCA *secure if for any polynomial p, and any probabilistic polynomial-time adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{ACCCA}}_{\mathcal{A},\mathcal{SE},p}(\eta)$ is a negligible function.*

Relating IND-ACCCA and IND-CCA security. Although apparently the notion that we define is stronger, due to the extra corruption capabilities, we now show that this is not the case. More precisely we prove the following theorem.

Theorem 1. *Let $\mathcal{SE}$ be a symmetric encryption scheme and p be a polynomial. Encryption scheme $\mathcal{SE}$ is* IND-CCA *secure if and only if it is* IND-ACCCA *secure.*

Clearly, security of encryption in the IND-CCA sense is a particular case of IND-ACCCA where the size of the list keys is 1, and the adversary does not have access to the corruption oracle $O_{\mathcal{C}}$ so any IND-ACCCA secure scheme is also IND-CCA secure. We prove that the converse also holds. More precisely, we show that for any adversary $\mathcal{A}$ against IND-ACCCA of $\mathcal{SE}$, there exists an adversary $\mathcal{B}$ against IND-CCA and a polynomial p such that

$$\mathbf{Adv}^{\mathsf{IND\text{-}ACCCA}}_{\mathcal{A},\mathcal{SE},p}(\eta) \leq p(\eta) \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathcal{SE}}(\eta)$$

The proof of the theorem uses a hybrid argument and is given in Appendix A. Furthermore, we can only prove that the reduction factor in the above theorem is optimal. We postpone the result for the full version of this paper.

4 Computational Soundness with Adaptive Corruptions

In this section we give our computational soundness result. As explained in the introduction, our goal is to show that adaptive corruption of ciphertexts can be treated by symbolic methods in a computationally sound manner. The result that we present in this section uses the trace mapping approach introduced Micciancio and Warinschi [MW04]. We do not attempt a strict extension of previous results that use this framework (e.g. the extension that treats asymmetric encryption and digital signature of Cortier and Warinschi [CW05]). The reason is that nested symmetric and asymmetric encryption combined with adaptive corruptions raises specific problems that are outside the goals of this paper.

Instead, we consider a simplified framework in which symmetric keys have already been distributed to parties. The class of protocols that we consider uses these keys, together with random nonces, party identities, and symmetric encryption. Furthermore, we consider protocols that do not send encryption keys over the network. We stress that keys used by protocol participants can be corrupted by the adversary. We prove that the computational traces of such protocols are the images of valid symbolic traces, with overwhelming probability over the choice of randomness used in the system. It had been shown in previous work how to use this soundness theorem to prove transference theorem of large classes of security properties [MW04, CW05]. The framework follows closely [MW04, CW05]. We highlight the differences that are due to adaptive corruption.

Protocol Syntax. We consider protocols that allow parties to exchange messages built from identities and randomly generated nonces using symmetric encryption. Consider an algebraic signature Σ with sorts ID, Key, Nonce, $\mathsf{Ciphertext}$, and Pair for respectively agent identities, keys, nonces, ciphertexts, and pairs. The sort Term is a supersort containing all other sorts, except Key. There are three operations: sk is defined on finite sets of the sort ID and return the symmetric key shared by the input identities. The other operations that we consider are pairing $\langle \_ \, , \_ \rangle : \mathsf{Term} \times \mathsf{Term} \rightarrow \mathsf{Pair}$ and symmetric encryption $\{\_\}_{\_} : \mathsf{Term} \times \mathsf{Key} \rightarrow \mathsf{Ciphertext}$. Protocols are specified using the algebra of terms constructed over the above signature from a set X of sorted variables. Specifically, $\mathsf{X} = \mathsf{X}.n \cup \mathsf{X}.a \cup \mathsf{X}.c \cup \mathsf{X}.s \cup \mathsf{X}.l$, where $\mathsf{X}.n, \mathsf{X}.a, \mathsf{X}.c$ are sets of variables of sort nonce, agent, ciphertext respectively. Furthermore, $\mathsf{X}.a$ and $\mathsf{X}.n$ are as follows. If $k \in \mathbb{N}$ is some fixed constant representing the number of protocol participants, w.l.o.g. we fix the set of agent variables to be $\mathsf{X}.a = \{A_1, A_2, \ldots, A_k\}$, and partition the set of nonce variables, by the party that generates them. Formally: $\mathsf{X}.n = \cup_{A \in \mathsf{X}.a} \mathsf{X}_n(A)$ and $\mathsf{X}_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$. This partition avoids to specify later, for each role, which variables stand for generated nonces and which variables stand for expected nonces.

The messages that are sent by participants are specified using terms in $\mathsf{T}_{\Sigma}(\mathsf{X})$, the free algebra generated by X over the signature Σ. The individual behavior of each protocol participant is defined by a *role* that describes a sequence of message receptions/transmissions. A k-party protocol is given by k such roles.

Definition 2 (Roles and protocols). *The set* Roles *of roles for protocol participants is defined by* $\mathsf{Roles} = ((\{\Pi_i\} \cup \mathsf{T}_\Sigma(\mathsf{X})) \times (\mathsf{T}_\Sigma(\mathsf{X}) \cup \{\mathsf{stop}\}))^*$. *A* k*-party protocol is a mapping* $\Pi : [k] \rightarrow \mathsf{Roles}$.

We assume that a protocol specification is so that $\Pi(j) = ((l_1^j, r_1^j), (l_2^j, r_2^j), \ldots)$, the j'th role in the definition of the protocol being executed by player A_j. Each sequence $((l_1, r_1), (l_2, r_2), \ldots) \in \mathsf{Roles}$ specifies the messages to be sent/received by the party executing the role: at step i, the party expects to receive a message conforming to l_i and returns message r_i. We stress that terms l_i^j, r_i^j are not actual messages but specify how the message that is received and the message that is output should look like.

Hypothesis over protocols. We restrict our results to the class of *executable protocols*, *i.e.* protocols where each role can be implemented by an executable program, using only the local knowledge of the corresponding agent. This requires in particular that any sent message (corresponding to some r_k^j) is always deducible from the previously received messages (corresponding to $l_1^j, \ldots, l_k^j$). We further restrict the class of protocols so that roles may not use nested encryptions (i.e. encryptions of encryptions) and may not send the keys, and encryptions of the keys used in the protocol. Notice that in particular ciphertext forwarding is not permitted as an adversary could use this ability to create nested ciphertexts. This also means that each message received by a party can be parsed completely.

Execution Models. For the protocols specified in the language described above we give symbolic and computational execution models. As usual, the symbolic execution is carried out using symbolic messages, and in the presence of a Dolev-Yao adversary. This adversary can inject messages into the communication, but can only "compute" new messages obeying the standard Dolev-Yao restrictions. In the concrete execution model, the messages that are exchanged are bit-strings and the honest parties and the adversary are p.p.t. Turing machines.

Formal Execution Model. Messages that parties exchange in the symbolic execution model are elements of the algebra T^f defined by:

$$T^f ::= a_i \mid \mathsf{sk}(a_1, \ldots, a_n) \mid n(a, j, s) \mid \langle T^f \ , T^f \rangle \mid \{T^f\}_{\mathsf{sk}(a_1, \ldots, a_n)}$$

where $a, a_1, \ldots, a_n \in \mathsf{ID}, j, s \in \mathbb{N}$.

Intuitively, the first three items correspond to identities, shared symmetric keys, and random nonces. We define the initial knowledge of an agent A by $\mathbf{kn}(A) = \{\mathsf{sk}(X) \mid A \in X\} \cup \mathsf{X}_n(A)$ *i.e.* an agent knows its shared keys, and also knows the nonces that it generates during the protocol execution.

Formally, the execution of the protocol in the symbolic world is defined by a state transition system. A *global state* of the system is given by (SId, f, H) where H is a set of terms of T^f representing the messages sent on the network and f maintains the local states of all sessions ids SId. Session ids are tuples of the form $(n, j, (a_1, a_2, \ldots, a_k)) \in (\mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k)$, where $n \in \mathbb{N}$ identifies the session, $a_1, a_2, \ldots, a_k$ are the identities of the parties that are involved in the

Initial knowledge: $\dfrac{}{S \vdash t}\ t \in S \qquad \dfrac{}{S \vdash b}\ b \in \mathsf{X}.a$

Pairing and unpairing: $\dfrac{S \vdash t_1 \quad S \vdash t_2}{S \vdash \langle t_1\ , t_2 \rangle} \qquad \dfrac{S \vdash \langle t_1\ , t_2 \rangle}{S \vdash t_i}\ i \in \{1, 2\}$

Encryption and decryption: $\dfrac{S \vdash t}{S \vdash \{t\}_{\mathsf{sk}(a_1 \ldots a_n)}} \qquad \dfrac{S \vdash \{t\}_{\mathsf{sk}(a_1 \ldots a_n)} S \vdash \mathsf{sk}(a_1 \ldots a_n)}{S \vdash t}$

Fig. 2. Deduction rules for the formal adversary

protocol and j is the index of the role that is executed in this session. Mathematically, f is a function $f : \mathsf{SId} \to ([\mathsf{X} \to T^f] \times \mathbb{N} \times \mathbb{N})$, where $f(\mathsf{sid}) = (\sigma, i, p)$ is the local state of session sid. The function σ is a partial instantiation of the variables occurring in role $\Pi(i)$ with elements from T^f and $p \in \mathbb{N}$ is the control point of the program. The transition between global states are determined by actions of the adversary. The adversary can corrupt parties, trigger the initialization of new sessions, and send messages to these sessions. The semantics of these actions is straightforward and is identical to prior work. We stress that as described in the introduction we now deal with adversaries who can adaptively corrupt parties. This is exhibited by the fact that the adversary can issue at any point a corruption request (which contrasts with prior frameworks where the corruption request was issued only in the beginning of the execution).

We write $\mathsf{SID} = \mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k$ for the set of all session ids. We define the set of *symbolic execution traces* by

$$\mathsf{SymbTr} = \left(2^{\mathsf{SID}} \times \left(\mathsf{SID} \to \left(\left[\mathsf{X} \to T^f\right] \times \mathbb{N} \times \mathbb{N}\right)\right) \times 2^{T^f}\right)^*$$

As usual in symbolic models [DY83], we restrict the messages that the adversary can send. More precisely, we consider the deduction relation $\vdash$ defined in Figure 2, and require that the messages sent by the adversary are computed from its knowledge, and the messages that it had seen so far using this deduction relation. In the figure, $S \vdash t$ means that the adversary is able to compute the term t from the set of terms S.

We remark that the rule that defines encryption is not standard in symbolic models and reflects a real capacity that computational adversary may possess: it may be possible for an adversary to create ciphertexts under an encryption key that it does not know, even if the encryption scheme is, say, IND-CCA secure. An alternative is to use the standard definition for symbolic security of encryption, but require that the scheme that is used in the implementation is both IND-CCA secure, and satisfies a version of authentication, i.e. IND-CTXT [BN00].

Definition 3. *An execution trace* $(\mathsf{SId}_1, f_1, H_1), \ldots, (\mathsf{SId}_n, f_n, H_n)$ *is said to be* valid *if the terms sent by the adversary are computed by Dolev-Yao operations. Formally, it is required that whenever the adversary sends some message t, then it is the case that* $H_i \vdash t$. *Given a protocol* Π, *we write* $\mathsf{Exec}^s(\Pi)$ *for the set of its valid symbolic execution traces.*

Concrete Execution Model. In concrete execution, the messages that are exchanged by parties are actual bit-strings that depend on the security parameter η (which determines for example the length of random nonces). We write $\mathcal{C}^\eta$ for the set of valid messages. The set $\mathcal{C}^\eta$ has subsets that contain values for agent identities, nonces, keys, ciphertexts, and pairs. We write $C^\eta.a, \mathcal{C}^\eta.n, \mathcal{C}^\eta.s, \mathcal{C}^\eta.c$, and $\mathcal{C}^\eta.p$ for these subsets, respectively. We assume that the implementation is such that each bit-string in $\mathcal{C}^\eta$ has a unique type which can be efficiently recovered via the function $\mathsf{type} : \mathcal{C}^\eta \rightarrow \{a, n, s, c, p\}$. In the concrete implementation, encryption is implemented with a symmetric scheme $\mathcal{SE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$ which we fix throughout this section. Pairing is implemented by some standard (efficiently invertible) encoding function $\langle \cdot\ , \cdot \rangle : \mathcal{C}^\eta \times \mathcal{C}^\eta \rightarrow \mathcal{C}^\eta.p$.

We represent the global state of the execution by a pair (SId, f). Here SId is the set of all session ids for the session currently executed, and f is a function that keeps track of the local state of each such session. More formally, session ids are tuples $(n, i, (a_1, a_2, \ldots, a_l))$, where $n \in \mathbb{N}$ is a unique session identifier, i is the index of the role executed in this session and $a_1, a_2, \ldots a_k \in \mathcal{C}^\eta$ are the names of the agents involved in running this session. The state function $f : \mathsf{SId} \rightarrow [\mathsf{X} \rightarrow \mathcal{C}^\eta] \times \mathbb{N} \times \mathbb{N}$, given a session id sid returns $f(\mathsf{sid}) = (\sigma, i, p)$ where σ assigns values to the variables of the program executed in this session (see the discussion regarding the execution of individual roles), i is the index of the role executed in this session and p is the program counter that keeps track of the next step to be executed in this session. Similarly to the symbolic setting, we represent the execution in this setting via transitions between global states. As in the symbolic model the adversary can corrupt parties, initiate new sessions, and send and receive messages to these sessions. Due to space limitations we defer the details for the full version of this paper.

The execution model that we described above uses randomization: the adversary is probabilistic, and the honest parties use randomization for generating nonces and encryptions. It can be shown that if the adversary $\mathcal{A}$ runs in polynomial-time, then the honest parties use a number of coins that is a polynomial in the security parameter. In the following, for a fixed adversary $\mathcal{A}$ we denote by $\{0,1\}^{p_{\mathcal{A}}(\eta)}$, resp. by $\{0,1\}^{g_{\mathcal{A}}(\eta)}$, the spaces from where the adversary, resp. the honest parties, draw the coins used in the execution. Notice that each pair of random coins $(R_{\mathcal{A}}, R_\Pi) \in \{0,1\}^{p_{\mathcal{A}}(\eta)} \times \{0,1\}^{g_{\mathcal{A}}(\eta)}$ determines a unique sequence of global states (SId_1, f_1), $(\mathsf{SId}_2, f_2), \ldots$, called the *concrete trace* determined by random coins $(R_\Pi, R_{\mathcal{A}})$. We write $\mathsf{Exec}_{\Pi(R_\Pi), \mathcal{A}(R_{\mathcal{A}})}(\eta)$ for this trace. We write $\mathcal{SId} = \mathbb{N} \times [k] \times (\mathcal{C}^\eta.a)^k$ for the set of all possible session ids. The set of all possible concrete traces is

$$\mathsf{ConcTr} = \bigcup_\eta \left(2^{\mathcal{SId}} \times (\mathcal{SId} \rightarrow ([\mathsf{X} \rightarrow \mathcal{C}^\eta] \times \mathbb{N} \times \mathbb{N}))\right).^*$$

Soundness Result. First, observe that concrete traces can be regarded as instantiations of formal traces by appropriate mappings from the abstract to the concrete representations. If we let $t^s = (\mathsf{SId}_1^s, f_1, H_1), \ldots, (\mathsf{SId}_n^s, f_n, H_n)$ and $t^c = (\mathsf{SId}_1^c, g_1), \ldots, (\mathsf{SId}_n^c, g_n)$ be a symbolic and a concrete execution trace, then we say that trace t^c is a *concrete instantiation* of t^s (or alternatively t^s is a *symbolic representation* of t^c) and we write $t^s \preceq t^c$ if there exists an injective function $c : T^f \rightarrow \mathcal{C}^\eta$ such that for every $i \in [n]$ it holds that $\mathsf{SId}_i^s = \mathsf{SId}_i^c$ and for every $\mathsf{sid} \in \mathsf{SId}_i^s$ if $f_i(\mathsf{sid}) = (\sigma^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}})$ and $g_i(\mathsf{sid}) = (\tau^{\mathsf{sid}}, j^{\mathsf{sid}}, q^{\mathsf{sid}})$ then $\tau^{\mathsf{sid}} = c \circ \sigma^{\mathsf{sid}}$, $i^{\mathsf{sid}} = j^{\mathsf{sid}}$ and $p^{\mathsf{sid}} = q^{\mathsf{sid}}$.

Informally, the soundness result that we obtain states that concrete traces of probabilistic polynomial adversaries are in fact instantiations of *valid* symbolic traces. This idea is captured formally by the following theorem.

Theorem 2. *Let Π be an executable protocol. If the concrete implementation scheme $\mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is* IND-CCA *secure and for any bit-string bs $\mathcal{D}(k, bs)$ and $\mathcal{D}(k', bs)$ cannot both succeed if $k \neq k'$ then for any p.p.t. algorithm $\mathcal{A}$*

$$\Pr\left[\exists t^s \in \mathsf{Exec}^s(\Pi) \mid t^s \preceq \mathsf{Exec}^c_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)\right]$$

is overwhelming. The probability is over the choice of randomness $(R_\Pi, R_\mathcal{A})$ used by the honest participants and the adversary.

Proof sketch: The proof of the above proposition does not use any conceptually novel ideas over prior work on trace mapping soundness [MP05, CW05]. Here we discuss it at a high level and highlight the technical role played by our security definition for encryption.

The proof proceeds by contradiction. Assuming the existence of a computational adversary $\mathcal{A}$ which with non-negligible probability produces an execution trace which is the image of a non Dolev-Yao trace, we show how to construct an adversary $\mathcal{B}$ who breaks IND-CCA security for $\mathcal{SE}$. We use Theorem 1 and we construct $\mathcal{B}$ against IND-ACCCA.

We proceed as follows: in a non Dolev-Yao trace (and thus in its computational counterpart), the adversary outputs at some point some data which is supposedly secret (it is only sent encrypted under keys that the adversary does not know). Due to the restrictions on the protocol, it can be easily seen that this secret is a nonce (had we allowed for nested encryption, this secret could have also been a ciphertext). Adversary $\mathcal{B}$ first guesses which of the secret nonces used during the execution of $\mathcal{A}$ will actually be produced by $\mathcal{A}$ (there are only polynomially many nonces, so the guess succeeds with sufficiently high probability), and then $\mathcal{B}$ proceeds to offer to $\mathcal{A}$ a simulation of the protocol with which $\mathcal{A}$ is supposed to interact. The simulation is such that the keys of the honest parties coincide with the keys used by the oracles of $\mathcal{B}$ (recall that $\mathcal{B}$ is against IND-ACCCA). $\mathcal{B}$ uses its decryption oracle to parse the messages sent by the adversary (in the case when these messages contain encryptions), and uses its encryption oracle to produce appropriate responses. The crucial aspect of the simulation is that ciphertexts which contain the secret nonce are produced using the left-right encryption oracle. Adversary $\mathcal{B}$ passes to the oracle two messages containing different values for the nonce, and obtains the encryption of the message selected according to the internal bit of the left-right oracle. Clearly, the ciphertexts obtained this way never need to be decrypted and the nonce never needs to be sent in clear (since in both cases the adversary would learn the nonce from the honest execution). At some point the adversary outputs the message that is not obtained using only Dolev-Yao computations, and which in particular contains the secret nonce. Adversary $\mathcal{A}$ then obtains the value of the nonce, and thus determines the internal bit of the left-right oracle.

During the simulation adversary $\mathcal{A}$ may decide to corrupt encryption keys. It is here where the enhanced capabilities of the adversary against encryption comes into play: $\mathcal{B}$ can easily answer such queries using access to its own key corruption oracle. Clearly, adversary $\mathcal{A}$ never asks to corrupt a key which was used to encrypt the secret nonce (as otherwise the nonce would not be secret), so all its corruption queries can be answered.

It has been already shown how to use this result to obtain general transference theorems for large classes of trace-based security notions [MW04] (e.g. entity authentication). We note that the proof of the above proposition can also be modified to yield transfer of secrecy properties from the symbolic to the computational world [CW05].

References

[AR00] Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, p. 3. Springer, Heidelberg (2000)

[BBM00] Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multiuser setting: Security proofs and improvements. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (2000)

[BDK07] Backes, M., Dürmuth, M., Küsters, R.: On simulatability soundness and mapping soundness of symbolic cryptography. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 108–120. Springer, Heidelberg (2007)

[Bea97] Beaver, D.: Plug and play encryption. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 75–89. Springer, Heidelberg (1997)

[BH92] Beaver, D., Haber, S.: Cryptographic protocols provably secure against dynamic adversaries. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 307–323. Springer, Heidelberg (1992)

[BN00] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)

[BP04] Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW 2004: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW 2004), Washington, DC, USA, p. 204. IEEE Computer Society, Los Alamitos (2004)

[BPS07] Backes, M., Pfitzmann, B., Scedrov, A.: Key-dependent message security under active attacks - BRSIM/UC-soundness of symbolic encryption with key cycles. In: Proceedings of 20th IEEE Computer Security Foundation Symposium (CSF) (June 2007); preprint on IACR ePrint 2005/421

[BPW03a] Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library (2003)

[BPW03b] Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: CCS 2003: Proceedings of the 10th ACM conference on Computer and communications security, pp. 220–230. ACM Press, New York (2003)

[BRS03] Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence of key-dependent messages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 62–75. Springer, Heidelberg (2003)

[CCL08] Cortier, V., Comon-Lundh, H.: Computational soundness of observational equivalence. In: CCS 2008: Proceedings of the 15th ACM conference on Computer and communications security, pp. 109–118 (2008)

[CF01] Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)

[CFGN96] Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multiparty computation. In: SToC 1996: Proceedings of the 28th annual ACM Symposium on Theory of Computing, pp. 639–648. ACM Press, New York (1996)

[CHK05] Canetti, R., Halevi, S., Katz, J.: Adaptively-secure, non-interactive public-key encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 150–168. Springer, Heidelberg (2005)

[CW05] Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)

[DDN00] Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. 30(2), 391–437 (2000)

[DN00] Damgaard, I., Nielsen, J.B.: Improved non-committing encryption schemes based on a general complexity assumption. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 432–450. Springer, Heidelberg (2000)

[DNRS03] Dwork, C., Naor, M., Reingold, O., StockmeyerMagic, L.: Magic functions. Journal of the ACM 50(6), 852–921 (2003)

[DY83] Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)

[Gol99] Goldreich, O.: Modern cryptography, probabilistic proofs and pseudorandomness. Springer, Berlin (1999)

[GS06] Gupta, P., Shmatikov, V.: Key confirmation and adaptive corruptions in the protocol security logic. In: Proceedings of FLOC Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis – FCS-ARSPA 2006 (2006)

[Lau04] Laud, P.: Symmetric encryption in automatic analyses for confidentiality against active adversaries. In: Proc. 25th IEEE Symposium on Security & Privacy, pp. 71–85 (2004)

[MP05] Micciancio, D., Panjwani, S.: Adaptive security of symbolic encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 169–187. Springer, Heidelberg (2005)

[MW04] Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)

[Nie02] Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002)

[NY90] Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: ACM Symposium on Theory of Computing, pp. 427–437 (1990)

[RS92] Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)

A Proof of Theorem 1

We prove the theorem in two steps. First, we show that the $\mathsf{IND\text{-}CCA}$ and $\mathsf{IND\text{-}ACCCA}$ notions are equivalent whenever the $\mathsf{IND\text{-}ACCCA}$ adversary is given access to encryption/decryption oracles that use a single key. Then, we leverage on this result to prove the general case.

Lemma 1. *Let $\mathcal{SE}$ be a symmetric encryption scheme. Then for any polynomial time adversary $\mathcal{A}$ against* $\mathsf{IND\text{-}ACCCA}$*, there exists a polynomial time adversary $\mathcal{B}$ against* $\mathsf{IND\text{-}CCA}$ *such that:*

$$\mathbf{Adv}^{\mathsf{IND\text{-}ACCCA}}_{\mathcal{A},\mathcal{SE},1}(\eta) = \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathcal{SE}}(\eta)$$

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}ACCCA}$. We construct an adversary $\mathcal{B}$ against $\mathsf{IND\text{-}CCA}$ as follows. Adversary $\mathcal{B}$ (which has access to the left-right encryption oracle $\mathcal{E}(k, \mathcal{LR}(\cdot,\cdot,b))$ and corresponding decryption oracle $\mathcal{D}(k,\cdot)$) runs $\mathcal{A}$ as a subroutine and answers its queries as follows.

1. When $\mathcal{A}$ sends a query (m_0, m_1) to its left-right oracle, $\mathcal{B}$ sends $(1, m_0, m_1)$ to its own oracle and forwards the answer to $\mathcal{A}$.
2. When $\mathcal{A}$ sends a query m to its normal encryption oracle, adversary $\mathcal{B}$ forwards (m, m) to its left-right encryption oracle and obtains some answer c. It then records the pair (m, c) and forwards the answer to $\mathcal{A}$.
3. When $\mathcal{A}$ sends c to its decryption oracle, then $\mathcal{A}$ searches its list of recorded pairs for an entry (m, c). If such an entry exists then it sends m to $\mathcal{B}$. Otherwise, it sends c to its decryption oracle and forward the answer to $\mathcal{A}$.
4. Finally, if $\mathcal{A}$ queries its corruption oracle, then $\mathcal{B}$ outputs 1 as its guess and finishes its execution.

Let Ev_b be the event that $\mathcal{A}$ uses its corruption oracle in the $\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}b}}$ game and let $\overline{\mathsf{Ev}_b}$ the complementary event. The first important observation is that Ev_b does not depend on b. This is true because $\mathcal{A}$ is not allowed to make use of its left-right oracle if it corrupts the key used by this oracle. In the remainder of this proof we write Ev for both Ev_0 and Ev_1. Due to the same reason, it is the case that $\Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}1}} = 1 \mid \mathsf{Ev}\right] = \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}0}} = 1 \mid \mathsf{Ev}\right]$. We therefore have that:

$$
\begin{aligned}
&\mathbf{Adv}_{\mathcal{A},\mathcal{SE},1}^{\mathsf{IND\text{-}ACCCA}}(\eta) = \\
&= \Pr[\mathsf{Ev}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}1}}(\eta) = 1 \mid \mathsf{Ev}\right] - \Pr[\mathsf{Ev}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}0}}(\eta) = 1 \mid \mathsf{Ev}\right] + \\
&\quad \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}1}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] - \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}0}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] \\
&= \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}1}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] - \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}0}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right]
\end{aligned}
$$

To finish up the analysis, we calculate the advantage of adversary $\mathcal{B}$ in the $\mathsf{IND\text{-}CCA}$ game:

$$
\begin{aligned}
&\mathbf{Adv}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA}}(\eta) = \\
&= \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}1}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}0}(\eta) = 1\right] \\
&= \Pr[\mathsf{Ev}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}1}(\eta) = 1 \mid \mathsf{Ev}\right] - \Pr[\mathsf{Ev}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}0}(\eta) = 1 \mid \mathsf{Ev}\right] \\
&\quad + \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}1}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] - \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{B},\mathcal{SE}}^{\mathsf{IND\text{-}CCA\text{-}}0}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] \\
&= \Pr[\mathsf{Ev}] - \Pr[\mathsf{Ev}] + \\
&\quad \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}1}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] - \Pr[\overline{\mathsf{Ev}}] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{SE}}^{\mathsf{IND\text{-}ACCCA\text{-}0}}(\eta) = 1 \mid \overline{\mathsf{Ev}}\right] \\
&= \mathbf{Adv}_{\mathcal{A},\mathcal{SE},1}^{\mathsf{IND\text{-}ACCCA}}(\eta)
\end{aligned}
$$

Lemma 2. *Let $\mathcal{SE}$ be a symmetric encryption scheme and p be a polynomial. Then for any probabilistic polynomial time adversary $\mathcal{A}$ against* $\mathsf{IND\text{-}ACCCA}$*, there exists a probabilistic polynomial time adversary $\mathcal{B}$ against* $\mathsf{IND\text{-}ACCCA}$ *such that:*

$$\mathbf{Adv}_{\mathcal{A},\mathcal{SE},p}^{\mathsf{IND\text{-}ACCCA}}(\eta) \leq p(\eta) \cdot \mathbf{Adv}_{\mathcal{B},\mathcal{SE},1}^{\mathsf{IND\text{-}ACCCA}}(\eta)$$

Proof. Let $\mathcal{SE}$ be a symmetric encryption scheme and p be a polynomial. Then we build $p(\eta)$ adversaries $\mathcal{B}_j$ against $\mathsf{IND\text{-}CPA}$ that use $\mathcal{A}$ as a sub-routine (j ranges between 1 and $p(\eta)$).

Adversary $\mathcal{B}_j$ works as follows. Recall that $\mathcal{B}_j$ works in the $\mathbf{Exp}_{\mathcal{B}_j,\mathcal{SE},1}^{\mathsf{IND\text{-}ACCCA\text{-}b}}(\eta)$ experiment, so it has access to a left-right encryption oracle keyed with some key k, a standard encryption oracle keyed with the same key, a decryption oracle keyed with k, and a corruption oracle (which when called returns k). $\mathcal{B}_j$ generates $p(\eta)$ keys $k_1, k_2, \ldots, k_{p(\eta)}$, using $\mathcal{KG}$ and simulates the environment of $\mathcal{A}$. The queries of $\mathcal{A}$ are answered as follows:

1. On query $\mathcal{E}(i, m)$ for a standard encryption, $\mathcal{B}_j$ proceeds as follows. If $i \neq j$ then it simply sends an encryption $\mathcal{E}(k_i, m)$. Otherwise, it sends $(1, m)$ to its own standard encryption oracle and forwards the answer to $\mathcal{A}_j$.

2. On a corruption query i, $\mathcal{B}_j$ returns k_i if $i \neq j$. Otherwise it calls its own corruption oracle and forwards the answer to $\mathcal{A}$.
3. On a decryption oracle (i, c), $\mathcal{B}_j$ returns $\mathcal{D}(k_i, c)$ if $i \neq j$; otherwise it sends the query $(1, c)$ to its own decryption oracle and forwards the answer to $\mathcal{A}$.
4. Finally, on a left-right query $(i, (m_0, m_1))$, $\mathcal{B}_j$ answers with $\mathcal{E}(k_i, m_0)$ if $i < j$, and with $\mathcal{E}(k_i, m_1)$ if $i > j$. Otherwise (i.e. when $i = j$) $\mathcal{B}_j$ sends $(1, (m_0, m_1))$ to its left-right encryption oracle and forwards the answer to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs a bit d, then $\mathcal{B}_j$ outputs d and halts.

If follows from the way we set up the simulation that the output of $\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}1}}_{\mathcal{B}_1,\mathcal{SE},1}(\eta)$ is identically distributed with $\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}1}}_{\mathcal{A},\mathcal{SE},p(\eta)}(\eta)$, that is:

$$\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}b}}_{\mathcal{B}_1,\mathcal{SE},1}(\eta) = 1\right] = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}b}}_{\mathcal{A},\mathcal{SE},p(\eta)}(\eta) = 1\right]$$

and similarly, $\mathcal{B}_{p(\eta)}$ where bit b is 0 is identical to the experiment involving $\mathcal{A}$ where bit b equals 0.

$$\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}0}}_{\mathcal{B}_{p(\eta)},\mathcal{SE},1}(\eta) = 1\right] = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}0}}_{\mathcal{A},\mathcal{SE},p(\eta)}(\eta) = 1\right]$$

Moreover, we have that:

$$\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}0}}_{\mathcal{B}_j,\mathcal{SE},1}(\eta) = 1\right] = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}1}}_{\mathcal{B}_{j+1},\mathcal{SE},1}(\eta) = 1\right]$$

We then get that:

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{IND\text{-}ACCCA}}_{\mathcal{A},\mathcal{SE},p}(\eta) = \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}1}}_{\mathcal{A},\mathcal{SE},p}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}ACCCA\text{-}0}}_{\mathcal{A},\mathcal{SE},p}(\eta) = 1\right] \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}1}}_{\mathcal{B}_1,\mathcal{SE},1}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}0}}_{\mathcal{B}_{p(\eta)},\mathcal{SE},1}(\eta) = 1\right] \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}1}}_{\mathcal{B}_1,\mathcal{SE},1}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}0}}_{\mathcal{B}_1,\mathcal{SE},1}(\eta) = 1\right] + \\
&\quad \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}1}}_{\mathcal{B}_2,\mathcal{SE},1}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}0}}_{\mathcal{B}_2,\mathcal{SE},1}(\eta) = 1\right] + \\
&\quad \ldots \\
&\quad \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}1}}_{\mathcal{B}_{p(\eta)},\mathcal{SE},1}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA\text{-}0}}_{\mathcal{B}_{p(\eta)},\mathcal{SE},1}(\eta) = 1\right] \\
&= \sum_{j=1}^{p(\eta)} \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B}_j,\mathcal{SE},1}(\eta) \leq p(\eta) \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathcal{SE},1}(\eta)
\end{aligned}$$

In the above, adversary $\mathcal{B}$ is the adversary that randomly samples an integer j in $[1, p(\eta)]$ and executes adversary $\mathcal{B}_j$.

How Many Election Officials Does It Take to Change an Election?

P.Y.A. Ryan

University of Luxembourg

Abstract. Electronic voting technologies have been quite widely deployed in a number of democracies. In the US for example, approximately 30% of the electorate used *Direct Recorded Electronic* (DRE) machines in the 2004 and 2008 presidential election. These systems have come in for significant criticism leading many experts to conclude that all digital technology used in voting must be flawed. While these critiques are fully justified and documented, to conclude that all voting technology must be irredeemably flawed is not in our view justified. The purpose of this note is to argue that, while it is undoubtedly true that careless introduction of information technology can undermine democracy, high assurance, verifiable voting schemes do exist and are worthy of serious consideration. Furthermore, such verifiable schemes can provide higher levels of trustworthiness than traditional pencil and paper, hand counting systems. The challenge remains to convince the various stakeholders of the trustworthiness of such schemes.

1 Introduction

The history of democracy is littered with instances of the outcome of elections being being altered by corruption of the process. Human ingenuity knows no bounds when it comes to devising ways to subvert voting systems. The US has a long, and not so proud, history of technological innovation in voting systems dating back to the end of the 18th century with the lever machines and followed by punch-cards, optical scanners, touch screens. For every one of these people quickly devised techniques to subvert the outcome. Excellent descriptions of this can be found in [7] and [8]. Indeed, Gumbel speaks of the "fallacy of the technological fix".

More recently, information technology has been widely deployed in various stages of elections: in maintaining an electoral role, in vote capture and for vote counting. In the US for example, approximately 30% of the electorate used so-called DRE (Directly Recorded Electronic) devices to cast and record their votes. Here, the voters choices are (supposedly) directly recorded in computer memory. Many experts have voiced serious concerns about such devices: the quality of the security engineering of all such systems that have been made available for examination has been shown to be abysmal, for example [11]. Indeed the whole approach can be argued to be fundamentally flawed: the accuracy of the count depends critically on the correctness of the software running on the devices at the

P. Degano and L. Viganò (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 211–221, 2009.

time of the election. Rivest and Wack, [15], have argued applying the principle of *software independence* to voting systems: in essence that no change in the software should result in an undetectable change in the outcome.

The suppliers of such DRE devices typically make claims about the verification and testing of the code but the openness and credibility of certification process is questionable. Typically the code is proprietary and copyright protected and so no available for general scrutiny. The recent California Secretary of State's "Top-to-bottom" report demonstrates that a number of voting machines that had been "certified" were not fit for purpose. This throws into question the whole certification process for voting technologies, in the US at least.

Many experts have gone on to conclude from these observations that any use of information technology in voting systems must undermine their trustworthiness. We fully concur with the view that introducing systems whose integrity depends on proprietary code that is not available for inspection must be inadmissible. However, we take issue with the claim that the lack of trustworthiness of many existing systems necessarily implies that no information technology can provide trustworthy elections. Indeed, we will argue that trustworthy and practical voting schemes already exist and that they can provide greater levels of assurance with weaker trust assumptions that the traditional paper and hand counting approach.

1.1 Verify the Election, Not the System

In summary, we are advocating the serious consideration of voting schemes designed to provide a high degree of transparency and auditability, within the constraints of the privacy requirements. The correctness of the outcome of the election should not depend on the correct behaviour of the code running during the election. Rather it should be possible to detect any error that could lead to a change in the tally.

Good candidates for such verifiable voting schemes do exist, several of which are conceptually and technologically sufficiently simple to be viable for use in real elections. In the remainder of the paper I will outline a number of the most interesting and promising such schemes.

1.2 Remote Voting

To be clear, I am not advocating remote, e.g., internet voting. Virtually any form of remote voting, whether it be postal, phone, internet, is vulnerable to threats of coercion and vote buying. It is extremely difficult to enforce the isolation of the booth in the remote context. Ingenious technical approaches to countering coercion threats in remote voting have been proposed, for example [10]. Even if some of these might be regarded as sound from a technical point of view, it is not clear that the electorate at large would feel sufficiently comfortable with them for them to be truly effective. A (technically) coercion resistent scheme that is not understood and trusted by the majority of the electorate is not in fact coercion resistent.

1.3 Resilience

I am not advocating a further point to be made clear is that, whilst I assert that the integrity of the count must not rely on the correct behaviour of the code, we are not advocating ignoring verification and testing. On the contrary, verification and testing must be our first line of defence and it is essential that every effort be made to ensure that the election runs smoothly, in particular, to make it as difficult as possible to undermine the smooth accurate execution of the election.

Thus, verification and testing is necessary but not sufficient: we need a second line and more solid line of defence based on verification of the election itself.

2 The Challenge of High-Assurance Voting Systems

Designing voting systems that are both trustworthy and trusted is immensely challenging. We need to ensure that two conflicting requirements are meet simultaneously: accuracy and ballot privacy. Clearly either would be trivial in isolation: for accuracy a simple show of hands is enough, for secrecy a constant function that always return the same outcome regardless of votes cast is enough. Alternatively, if we are prepared to place our trust in a third party, then again the problem is quite trivial. However, our goal will be to guarantee both these properties with minimal trust assumptions. Ideally we would like to eliminate trust, in the sense of dependence, entirely. In practice it seems that it is not feasible to reduce dependence to zero.

The challenge is made even more daunting by also requiring the systems to be sufficiently simple to easily usable and readily trusted by the majority of voters. Advances have been made in recent years, mainly in schemes that make heavy use of cryptographic techniques to reconcile these conflicting requirements. It is essential that voters have sufficient understanding of the security mechanisms for them to trust the guarantees they provide, and be motivated to contribute to the dependability. Cryptography, by its very nature is rather sinister and forbidding to the majority of the human race. A system that a cryptographer claims is "transparent", in the sense of highly auditable, may be totally opaque to most voters. Thus we must find ways to either avoid the use of cryptography, or ensure that it's use is simple enough and can be sufficiently clearly explained, perhaps vis suitable metaphors, to instill widespread trust.

Another fundamental problem with elections, in contrast to most other critical systems, is that there is no "god's eye" view that would tell us the correct answer.

3 A Brief History of Verifiable Voting

In this section I give a very brief outline of recent developments in verifiable voting. This is necessarily very brief and makes no pretense of scholarship.

Verifiable, secret elections can be regarded as a special case of the secure, distributed computation problem: a number of entities each know secret values and they wish to compute some function of these values in a verifiable fashion in such a way as not to reveal any of the secret values. Some very elegant schemes have been proposed that take this approach but these typically require each voter to have computational, in particular cryptographic capabilities. This is fine in theory and perhaps reasonable in some contexts, e.g. elections for officers of the International Association for Cryptologic Research perhaps, but not a reasonable assumption for most political elections. More recent developments have striven to come up with schemes that minimise the capabilities required of the voters and make the voter experience as simple and familiar as possible. It is these schemes that I will focus on here.

The first suggestion that cryptographic techniques could be applied to voting systems appears to be Chaum's 1981 paper that introduced the idea of anonymising mixes, [3]. In 1994, Benaloh and Tunistra, [2], introduce the notion of coercion-resistance along with a scheme using homomorphic tabulation that satisfies it. Later, Chaum [4] and Neff [13] introduced schemes that could be regarded as more practical than previous schemes. The original Prêt à Voter scheme, [17] and [16] was inspired by the Chaum scheme, replacing the visual cryptography by the candidate permutation concept. Chaum has subsequently adopted this concept in his new PunchScan scheme, [1]. Chaum subsequently proposed Scantegrity II, [6] which will be outlined below. Recently, Rivest has proposed *ThreeBallot*, [14] that provides voter-verifiability without using cryptography.

4 Evaluating Secure Voting Systems

The question of the title is of course rather tongue-in-cheek, but it does point to a serious issue: how does one evaluate and compare often vastly different voting systems and technologies? One possible rough measure is to estimate the number of people who would have to collude in order to corrupt an election. In practice coming up with such estimates is extremely difficult: we need good threat models, people fall into different categories (voters, election officials, supplier technicians, cryptographic experts, etc.). Voting systems are examples par excellence of socio-technical systems: it is not enough to examine the technical core, the code, the cryptographic algorithms etc, we need also to take account of the behaviour of the humans and their interactions with the technical mechanisms.

In this section we discuss the strengths and weakenesses of a representative set of voting systems.

4.1 Pen-and-Paper

In the case of a conventional pen-and-paper system with hand counting, like the one used in the UK, it can be argued that it would take quite a large number of colluding agents to alter the outcome, especially given the First-Past-The-Post system, though the number drops as the election margin drops. In the UK anyone can request to be an observer of the election process. This lends a significant level of transparency to the process, but clearly no observer will be able to observe more that a tiny fraction of execution of a general election. Many ways have been found to corrupt votes in a virtually undetectable fashion. Ballot boxes can be manipulated in transit or storage. Jones, [9], observes that virtually all ballot box locks and seals are quite easy to manipulate and fake. A piece of pencil lead secreted under a thumb nail can be used to surreptitiously spoil ballot forms during counting, and so on.

4.2 Touch Screen Machines

For the touch screen devices used in recent US presidential elections, it appears that just one corrupt technician or official could quite easily swing an election. Indeed it has been argued that with some systems, a voter with a little knowhow could corrupt an arbitrary number of votes. Furthermore, if done with a little subtlety, the rigging will be virtually undetectable.

Until recently, such vulnerabilities were essentially theoretical, with no documented exploitation. Recently however we have seen the first documented case of ballot corruption with electronic voting machines. In Clay County Kentucky a number of officials have been indited for large scale corruption of votes, [20]. The attack was socio-technical in nature, exploiting weaknesses in the design of the interface and altering guidance to the voters to fool them into failing to finalise their vote, allowing the officials to go into the booth after and “correct” and finalise the vote for them.

4.3 Voter-Verifiable Paper Audit Trails

A common response to the concerns about the lack of auditability of touch screen machines is to propose the incorporation of a Voter-Verifiable Paper Audit Trail (VVPAT) mechanism. This is essentially a printer attached to the side of the machine that prints the ballot in a way that the voter can examine and approve if correct, or challenge if incorrect. Typically the printed ballot is shown under glass to the voter to prevent tampering, the so called *Mercuri method*, [12]. This indeed has the benefit of providing an independent audit trail that can be invoked if the electronic count is called into question. Such mechanisms have been incorporated in a number of DRE style products. In fact it has a number of serious shortcoming:

- fallability of a paper audit
- failure of voters to properly verify the ballot
- challenging the printed ballot may violate the voter’s privacy
- lack of clarity regarding when the paper audit trail should be invoked

Nonetheless, VVPAT schemes are regarded as falling within the definition of software independent.

4.4 Cryptographic Schemes

The final category I want to discuss is that of cryptographically based schemes. These strive to provide a notion of *Voter-verifiability* though the term is potentially misleading as it is quite different from the sense used in VVPAT systems. The idea here is to provide voters with the means to confirm to their own satisfaction that their vote is accurately included in the count while not providing a way to prove to a third party how they voted. A first glance this may seem impossible and indeed some pretty nifty cryptographic magic is required to pull it off. The key idea is to provide the voter with a receipt that carries their vote in encrypted or encoded form. This is done in such a way that nobody can extract the vote but a threshold set of Trustees can extract the votes and perform the count in a way that maintains ballot secrecy. I will outline a number of such schemes below, for the moment I make a number of observations:

Such schemes appear to give a high degree of trustworthiness as a consequence of the high degree of auditability of the whole process. Assurance arguments for voting systems can usefully be broken down into three elements:

- cast as intended
- recorded as cast
- counted as recorded

For cryptographic schemes what this means is that:

Votes must be faithfully encoded in the receipts. Arguably this is the most delicate part of the process. Voters cannot be expected to check the correct encryption of their vote in there receipt, nor would we wish to give them this capability as this would enable them to prove their vote to a coercer or vote buyer. Consequently we must use indirect means to convince the voters of the correctness of the encryption. I will discuss some ways to achieve this in the sections on particular schemes, but note that this is arguably the most challenging aspect of the whole endeavor. We need mechanisms that are technically sound but also sufficiently simple for voters to understand and trust.

We need to ensure that all legitimately cast receipts, and only these, are included in the tabulation phase. This is where the voter verification comes into play: all receipts should be posted to some form of public Bulletin Board (BB) and voters can check that their receipt appears accurately. If it does not they can protest and they hold a receipt to substantiate their complaint. Leaving aside for the moment issues of voter diligence in checking their receipts and ballot stuffing of the BB, this should serve to ensure that all legitimately cast votes are entered into the tabulation.

Finally, we need to ensure that all posted receipts are correctly translated back into votes. This phase is almost totally technical in nature and rather well understood. A number of established techniques are known for verifiably tabulating encrypted ballots in an anonymous way. In a sense, by introducing the encrypted receipts we have transformed the problem into one that is purely mathematical and hence fully verifiable. The problems of course remain at the edges: in ensuring that the transformation into the purely mathematical domain is faithful, and furthermore, universally seen as faithful.

5 Voter-Verifiable Schemes

In this section I outline a number of recent proposals for practical, verifiable voting.

5.1 Prêt à Voter

The Prêt à Voter approach to verifiable voting, [5,18,19], is distinguished by the particularly simple way that the voter makes their choice and this is translated into an encrypted receipt. The key innovation of the Prêt à Voter approach is the way votes are encoded using a randomised candidate list. An important observation about this way of encoding the vote is that, in contrast to previous schemes, there is no need for the voter to communicate their vote to an encryption device. What is encrypted is the information that defines the frame of reference for any given ballot form, and this can be computed in advance, before the ballot has been associated with any voter let alone a vote choice.

At the polling station, the voter pre-registers and chooses at random a ballot form from a pile of forms individually sealed in envelopes. An example form is shown in Figures 1.

In the booth, the voter removes the ballot from its envelope and makes her selection in the usual way by placing a cross in the right hand column against the candidate of choice, or, in the case of a Single Transferable Vote (STV) system for example, she marks her ranking against the candidates. Once the selection has been made, she detaches and discards the left hand strip that carries the candidate order. The remaining right hand strip now constitutes the receipt, as shown in Figure 2.

Candidates	Vote
Obelix	
Idefix	
Asterix	
Panoramix	
	$7rJ94K$
Destroy	Retain

Fig. 1. Prêt à Voter ballot form

Your Vote
X
$7rJ94K$
Retain

Fig. 2. Prêt à Voter ballot receipt (encoding a vote for "Idefix")

Anne now exits the booth with this receipt, registers with an official and casts her receipt in the presence of the official. The ballot receipt is placed against an optical reader or similar device that records the cryptographic value at the bottom of the strip, that we will refer to henceforth as the ballot *onion*, and an index value i indicating the cell into which the X was marked. A digital signature is computed over the onion and index values and this signature is printed on Anne's receipt.

The digitized copies of the receipts are transmitted to a central tabulation server which posts them to a secure WBB. Voters are encouraged to visit this WBB and confirm that their receipt appears correctly and, if their receipt does not appear, or appears incorrectly (i.e., with the X in the wrong position), they can appeal.

After a suitable period, assuming that checks on the posted receipts are satisfactory or at least suitably resolved, we can start the tabulation process. For this we employ standard, verifiable, anonymising tabulation techniques. I omit the details here, they can be found in, for example, [19].

5.2 Three-Ballot

ThreeBallot is an interesting new scheme has proposed by Rivest, [14], that achieves voter-verifiability and unconditional privacy without using any cryptography. In essence, voters cast three ballots in such a way as to allocate two votes to their candidate of choice and exactly one vote to all other candidates. All ballots carry unique, pure random serial numbers.

All three ballots are cast, recorded and posted to a secure WBB. At the time of casting, the voter retains a copy of one out of their three ballots, chosen arbitrarily. The copy should be created in such a way to ensure that the system does not learn which has been retained. This chosen form they can check against the WBB. As a result, an attempt to corrupt a ballot stands a 1/3 chance of being detected.

Obelix		Obelix	X	Obelix	
Idefix		Idefix		Idefix	X
Asterix	X	Asterix		Asterix	X
Panoramix		Panoramix	X	Panoramix	
762067		4567023		3633209	

Fig. 3. ThreeBallot ballot form (showing a vote for Asterix)

Obelix	X
Idefix	
Asterix	
Panoramix	X
4567023	

Fig. 4. ThreeBallot ballot receipt

The tabulation is easy to perform, involves no cryptography and universally verifiable. Suppose that there are n voters. Votes for each candidate are totalled over all the $3n$ posted ballots. n is subtracted from the total for each candidate to give the votes cast for each candidate. Note that a receipt does not reveal how the vote was cast.

This scheme is remarkable in that it achieves a degree of voter-verifiability without the use of cryptography. Unfortunately, it seems that the scheme, at least as it currently conceived, is probably not suitable for real elections. The voting rules are not trivial and clearly they need to be enforced: if a voter is allowed to cast three votes against a candidate or indeed none against another, fairness will be violated. It is not clear how to enforce the voting rules in such a way as to maintain vote secrecy. It is also not clear how to provide the voter with a copy of just one ballot whilst ensuring that the system cannot learn which the voter has chosen.

The scheme is also vulnerable to subtle forms of coercion, for a example a coercer might require a voter to cast their vote according to a certain pattern. If ballots corresponding to this pattern do not appear on the WBB then punishment follows. This can be countered by not allowing the voter the freedom to choose an initial distribution of "null" crosses by having a device allocate a random distribution to which the voter then adds their own, casting X. This too has problems, for example, how to ensure that the device really does allocate the initial Xs randomly and does not record any information.

Thus, the scheme is of immense theoretical importance but probably not practical as it stands.

5.3 Scantegrity II

Scantegrity II is a recent scheme proposed by Chaum, [6], that provides verifiability while being compatible with existing US optical scanner machines. The concept is interesting in that here the voter is not provided with a receipt, as with previous voter-verifiable schemes. Rather the voter gets to learn a random code associated with her choice which she can note down and subsequently use to check that her vote has been correctly recorded. An ingenious use of invisible ink ensures that the voter only learns the code associated on her ballot with her choice and no other codes on the ballot. Scratch strips could also be used to ensure that only the code for the chosen candidate is revealed. The point of this is that if she finds that a different code is recorded for her ballot and she protests, there is a mechanism to reveal the alternative codes for her ballot. If it turns out the code she claims to have noted is indeed a valid code for this ballot then it is likely that the system has tried to corrupt her vote. Alternatively, if the code she claims is not a valid code for her ballot then it is likely that she is merely trying to discredit the voting system by making up an alternative code. Thus, the fact that she is denied knowledge of the alternate codes means that if she claims a different valid code than that recorded then her claim should be regarded as evidence of corruption.

The ballot forms are essentially the standard forms used in optical scan systems with the difference that random codes are written in invisible ink in the bubbles against the candidates. When a voter fills in the bubble of her choice, using a specially provided pen, the code is revealed. All other codes should remain hidden as long as only one bubble is filled. If more than one bubble (for a given race) is filled this is detected by the optscan software and the ballot rejected.

Information defining the codes is stored in a secret data base which is used to perform the tabulation against recorded codes in a verifiable fashion, details in [6]. The scheme has a number of advantages: it is compatible with existing technology in the US and so for voters already familiar with opscan machines the voting experience is virtually unchanged, aside from the optional step of verifying their codes. The outcome can be computed from the codes but the scheme also produces as a side effect a VVPAT that can be invoked if necessary.

6 Conclusions

There has been much discussion about the merits or demerits of introducing digital technology into the process of voting. Many argue that moving to digital technology is simply too dangerous and it is better to stick with tried and tested pen-and-paper along with hand counting. Whilst the critiques of many existing voting products and technologies are well-founded, the extrapolation of these observations to conclude that all technologies must inevitably be flawed is unfounded. Indeed many of these polemics appear unaware of, or prefer to ignore, the existence of high-assurance, verifiable schemes. There is also a tendency to conflate the problems of information technologies and those of remote forms of voting, in particular internet voting. In this note I have sought to clarify these issues and put the case for software-independent, verifiable voting systems.

References

1. `http://punchscan.org/index.php`
2. Beneloh, J., Tuinstra, D.: Receipt-free secret-ballot elections. In: Symposium on Theory of Computing, pp. 544–553. ACM, New York (1994)
3. Chaum, D.: Untraceable mail, return addresses and digital pseudonyms. Communications of the ACM 24(2), 84–88 (1981)
4. Chaum, D.: Secret-ballot receipts: True voter-verifiable elections. IEEE Security and Privacy 2(1), 38–47 (2004)
5. Chaum, D., Ryan, P.Y.A., Schneider, S.: A practical, voter-verifiable election scheme. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005)
6. Chaum, D., et al.: Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes. Technical report. In: Proceedings of USENIX-ACCURATE EVT (2008)
7. Gumbel, A.: Steal this vote! Nation Books (2005)

8. Jones, D.W.: A brief illustrated history of voting (2003), `http://www.cs.uiowa.edo/~jones/voting/pictures`
9. Jones, D.W.: Threats to voting systems (2005), `http://vote.nist.gov/threats/papers/threatstovotingsystems.pdf`
10. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant Electronic Elections. In: Proceedings of the 2005 ACM workshop on Privacy in the electronic society (November 2005)
11. Kohno, T., Stubblefield, A., Rubin, A.D., Wallach, D.S.: Analysis of an electronic voting system. In: Symposium on Security and Privacy. IEEE, Los Alamitos (2004)
12. Mercuri, R.: A better ballot box? IEEE Spectrum Online (October 2002)
13. Neff, A.: A verifiable secret shuffle and its application to e-voting. In: Conference on Computer and Communications Security, pp. 116–125. ACM, New York (2001)
14. Rivest, R.L.: The three ballot voting system (2006), `http://theory.lcs.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf`
15. Rivest, R.L., Wack, J.P.: On the notion of "software independence" in voting systems. Transactions of the Royal Society (2008)
16. Ryan, P.Y.A.: Towards a dependability case for the chaum voting scheme. In: DIMACS Workshop on Electronic Voting – Theory and Practice (2004)
17. Ryan, P.Y.A.: A variant of the chaum voting scheme. Technical Report CS-TR-864, University of Newcastle upon Tyne (2004)
18. Ryan, P.Y.A.: A variant of the chaum voting scheme. In: Proceedings of the Workshop on Issues in the Theory of Security, pp. 81–88. ACM, New York (2005)
19. Ryan, P.Y.A., Schneider, S.: Prêt à Voter with Re-encryption Mixes. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 313–326. Springer, Heidelberg (2006)
20. Schneier, B.: Election fraud in kentucky (2009)