Andrei Sabelfeld (Editor)

Proceedings of

FCS'04

Workshop on Foundations of Computer Security

Turku Centre *for* Computer Science

TUCS General Publication
No 31, June 2004

Proceedings

Foundations of Computer Security

Affiliated with LICS'04 and ICALP'04

Turku, Finland
July 12–13, 2004

Featuring a *Sub-Workshop on Logical Foundations of an Adaptive Security Infrastructure*, organized by Leo Marcus

Edited by
Andrei Sabelfeld

Table of Contents

Security Protocols II

Language-Based Security I

Language-Based Security II

Invited Talk

Workshop on Logical Foundations of an Adaptive Security Infrastructure

Preface

Computer security is an established field of computer science of both theoretical and practical significance. In recent years, there has been increasing interest in logic-based foundations for various methods in computer security, including the formal specification, analysis and design of cryptographic protocols and their applications, the formal definition of various aspects of security such as access control mechanisms, mobile code security and denial-of-service attacks, and the modeling of information flow and its application to confidentiality policies, system composition, and covert channel analysis.

This workshop continues a tradition, initiated with the Workshops on Formal Methods and Security Protocols — FMSP — in 1998 and 1999, then with the Workshop on Formal Methods and Computer Security — FMCS — in 2000, and finally with the LICS satellite Workshop on Foundations of Computer Security — FCS — in 2002 and 2003, of bringing together formal methods and the security community. The aim of the workshop this year is to provide a forum for continued activity in this area, to bring computer security researchers in contact with the LICS and ICALP community, and to give LICS and ICALP attendees an opportunity to talk to experts in computer security.

FCS received 45 submissions. The Program Committee selected 13 of them for presentation as the outcome of the reviewing process.

The contributions of many people have made the workshop a success. The Program Committee took reviewing and selection seriously and provided much useful feedback in the reviews. Many thanks are due to Martín Abadi, the FCS invited speaker, for instantly agreeing to take up an FCS invited talk. Phil Scott and Mika Hirvensalo, our connections to LICS and ICALP provided extremely efficient help for FCS to run smoothly. This year FCS features a Sub-Workshop on Logical Foundations of an Adaptive Security Infrastructure. Dedicated efforts of Leo Marcus, who organized this event, were rewarded by cutting-edge papers and talks. Most of all, we are thankful to the authors and the attendees who made this workshop an inspiring and fruitful event.

Andrei Sabelfeld
FCS'04 Program Chair

Program Committee

Tuomas Aura, Microsoft Research Cambridge, UK
Gilles Barthe, INRIA, France
Michele Bugliesi, University of Venice, Italy
Iliano Cervesato, ITT Industries, USA
Mads Dam, KTH/SICS, Sweden
Sabrina De Capitani Di Vimercati, University of Milan, Italy
Joshua Guttman, MITRE, USA
Naoki Kobayashi, Tokyo Institute of Technology, Japan
Mogens Nielsen, BRICS, Denmark
Valtteri Niemi, Nokia Research Center, Finland
Andrei Sabelfeld (chair), Chalmers, Sweden
Andre Scedrov, University of Pennsylvania, USA
Robin Sharp, DTU, Denmark
Vitaly Shmatikov, SRI, USA

Session I

Mobility

Typing migration control in $lsd\pi$.

Francisco Martins* António Ravara**

Abstract

This paper presents a type system to control the migration of code between sites in a concurrent distributed framework. The type system constitutes a decidable mechanism to ensure specific security policies, which control remote communication, process migration, and channel creation. The approach is as follows: each network administrator specifies sites privileges, and a type system checks that the processes running at those sites, as well as the composition of the sites, respect these policies. At runtime, well-typed networks do not violate the security policies declared for each site.

1 Introduction

Framework. A natural and simple framework to study distributed mobile systems is DPI, the distributed π-calculus of Hennessy and Riely [4], which extends the π-calculus [7, 8] by locating processes on a (flat) network of *sites*—named places where computation occurs. Communication only happens within sites (to avoid "global synchronisation"), but processes may migrate from one to another. However, DPI presents no notion of location of a resource: a global entity is responsible for allocating and managing memory.

DiTyCO, a distributed extension of the TyCO programming language [5, 12], addresses this drawback. It follows the DPI model in what respects the notions of locality (or *site*) and communication, but uses a lexically scoped regime for names that rules process migration. The main motivations of DiTyCO are: (1) a *network-aware style of programming* in the sense that the time and the place associated with the creation of new resources (sites and channels) is explicit in the syntax of the programs; and (2) an easier implementation, based on current technology, by providing the compiler with information to generate code for the creation, and the access to these resources. Migration of resources is a subset of the resource access operations, and thus is clearly bounded in the program's source code. Clients do not interact remotely with a server. Rather, they move to the site of the server and interact locally, eliminating the costs of

*Department of Mathematics, University of Azores, Portugal. E-Mail: fmartins@di.fc.ul.pt

**CLC and Department of Mathematics, Instituto Superior Técnico, Lisboa, Portugal. E-Mail: aravara@ist.utl.pt

maintaining long remote sessions between clients and servers. The latest release of the language DiTyCO is available on the TyCO project site (http://www.ncc.up.pt/tyco/).
Aim. DiTyCO lacks a security mechanism to control the usage of important resources such as memory and *cpu* cycles. This work addresses such lacuna. Our main motivation is the control of code mobility by means of a simple, decidable, and low complexity type system. The system checks the integrity and the consistency of user-declared security policies, guaranteeing that well-typed networks are free of (runtime) security violation errors. Our approach consists in a simple set-based static analysis where the network administrators associate security policies to the sites they supervise, and by this mean, tailor the allowed interaction in a network between sites that know each other.

To focus on the problem, we develop our work using a lexically scoped distributed version of the pi-calculus—the $lsd\pi$ calculus [11]—that is the theoretical basis of DiTyCO. This calculus seems more suitable to study code migration in a distributed setting than other proposals, since the migration is triggered by channels themselves, rather than using explicit migration primitives (for an overview of distributed mobile calculi see a deliverable of an EU project [1]). We choose a monadic version of a flat calculus because the stress of our work is purely in the control of process migration rather than on communication, or on hierarchical issues. Indeed, DiTyCO is a good choice since it is a simple setting that lead us to a clear understanding of the security issues underlying code migration, and let us settle the basis for reasoning about resource usage.

To control the migration of processes between sites, we use three security policies, to control respectively *remote communication, code migration,* and *channel creation*. These policies are directly related to the actions of the calculus, and therefore are independent from each other. *Remote communication* refers to the ability of a thread to send a message to a resource located at a distant site. *Code migration*, in turn, means that a thread may cross site boundaries, exiting its current site and entering a new one. This operation can be understood, from the source site point of view, as an upload of code. Finally, *channel creation* represents the ability of a thread to create a new channel in a foreign site. Mastering channel creation is important because (1) if a thread is able to create a channel on a remote site it means, as discussed later, that it would be able to migrate code to that destination, and (2) it may also give rise to a denial of service attack, if the source site creates an arbitrarily large number of channels in the destination site, consuming important resources, such as memory.

The definition of a security policy for a given site consists on the enumeration of the sites allowed to perform the monitored actions. Thereafter, the type system checks whether these policies are followed by each process running in the site, and by the other sites in the network that is being checked. Notice that we assume a closed world. This system is a tool to declare how code in other known sites may affect the computation of a given site, and to verify the compositionality of all these sites. The type system checks if the processes running at a given site respect its security

$$\text{simple channels} \quad a, b, c, x, y, z \in \mathcal{C} \qquad \text{sites} \quad r, s, t \in \mathcal{S}$$
$$\text{sets of sites} \quad S, R, T \subseteq \mathcal{S} \quad \text{channels} \quad u ::= a \mid a@s \quad \text{values} \quad v ::= () \mid u$$
$$\text{processes} \quad P, Q ::= \mathsf{0} \mid u\,!\,\langle v \rangle \mid u?(x : S)\ P \mid u?*(x : S)\ P \mid P \,|\, Q \mid (\boldsymbol{\nu}\, u)\ P$$
$$\text{networks} \quad N, M ::= \mathsf{0} \mid s_G[P] \mid N \parallel M \mid (\boldsymbol{\nu}\, a@s)\ N$$

$$\begin{array}{rcll} \Gamma & ::= & \{s_1 : (\varphi_1, G_1), \ldots, s_n : (\varphi_n, G_n)\} & \text{(typings)} \\ \varphi & ::= & \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\} & \text{(site types)} \\ G & ::= & \{\mathrm{rem} : S_1, \mathrm{mig} : S_2, \mathrm{new} : S_3\} & \text{(site policies)} \\ \gamma & ::= & \mathrm{ch}(\gamma)@S^t \mid \mathrm{val} & \text{(channel types)} \\ t & ::= & o \mid i \mid b & \text{(site tags)} \end{array}$$

Figure 1: Syntax of $lsd\pi$.

policies, and if all the sites one wants to compose in a network will interoperate without violating each other policies.

Outline. Next section presents the syntax of the calculus, and of the type system. Section 3 is devoted to the operation semantics of the calculus. The main section of the paper is Section 4, where we introduce the type system, argue about some examples of networks that should be rejected by the type system, formalise a notion of runtime error suitable for our setting and purpose, and prove type safety. Section 5 concludes the paper, presents related work, and points directions for future research.

2 The calculus

This section briefly describes the calculus as well as its types, and some examples of networks, hinting informally the semantics of $lsd\pi$. A thorough presentation of the calculus is elsewhere [11].

Syntax. The $lsd\pi$ calculus extends the pi-calculus [8] distributing processes over flat networks of named sites. Communication occurs only within a site. Resources, located at creation time, maintain their location throughout the computation. Unique to the calculus is the notion of lexical scoping, a well-known feature from main-stream programming languages like, for instance, Pascal, C, Java, or ML. In $lsd\pi$ this means that a channel may be addressed by its simple name—the channel—when it is at home, or by its located (or global) name—the pair channel–host site—whenever the reference is made from a foreign site.

The syntax of the calculus used in this paper is described in Figure 1. Fix a denumerable set of *simple channels*, $\mathcal{C}$, ranged over by a, b, c, x, y, z, and a denumerable set of *sites*, $\mathcal{S}$, ranged over by r, s, t, disjoint from $\mathcal{C}$. *Channels*, may be simple—a—or located—$a@s$—, referring to a channel a from a site s. Let $\mathcal{N} = \mathcal{C} \cup \mathcal{S}$ be the set of

the resource identifiers. *Processes* are the standard π-calculus processes, apart from the input process, $u?(x : S)\ P$, as well as the replicated input, $u?*(x : S)\ P$, which mention a finite set of sites S. This set plays an important role in the assurance of security policies: it restricts the channels that may instantiate x, enumerating the sites allowed to host them. Notice, however, that in $lsd\pi$ sites are not first class citizens, in the sense that they are not passed around. The claim is that there is no purpose to reveal a site location. Instead, disclosing a port at a site (a located channel) is all that is needed to establish a link to it.

Sites constitute the basic building blocks for networks. A network $s_G[\![P]\!]$ denotes a site s, where a process P is running, with security policies bound by G. This set G defines the interactions allowed between s and the other sites in the surrounding network. It is intended to be written by the site administrator, who thus declares in this way the site security policies. The section on types presents these sets of security policies in detail. Networks are putted together using the *network parallel construct* $N \parallel M$. We use a different symbol from parallel processes (*c.f.* $P \mid Q$) to stress the fact that there is no communication between networks. The interaction between networks occurs through explicit migration of processes among sites. The remaining constructs allow us to restrict located channels in a network, $(\boldsymbol{\nu}\, a@s)\ N$, and $\mathsf{0}$ denotes an inactive network.

Notice the absence of an operator to create sites, which are thus constants. This scenario fits the present situation of DiTyCO. Allowing to create and pass around sites significantly complicates the technical work, since one has to take into account dynamic changes of site policies. An extension of the work reported here to incorporate this aspect has been developed and is presented elsewhere [6].

As an example, the following $lsd\pi$ term

$$r_{G_1}[\![a@s\,!\,\langle b\rangle]\!] \parallel s_{G_2}[\![a?(x : S)\ P]\!]$$

represents a network consisting of two sites r and s with security policies G_1 and G_2, respectively (that we do not detail now). The output process running at site r is willing to deliver a message to the channel a from site s. In s the channel a declares which sites may remotely communicate with it and instantiate x. So, if r is in S, the process migrates first from r to s and then communicates with a within s. Notice that process $a@s\,!\,\langle b\rangle$ gives a clear understanding of the sites each name belongs to: the located channel $a@s$ is hosted by s; the simple channel b is from r. The above network reduces in two steps to

$$r_{G_1}[\![\mathsf{0}]\!] \parallel s_{G_2}[\![P[b@r/x]]\!]$$

where r is now explicitly mentioned in the reference to channel b, since it "left home".

Types. Our approach consists in the specification of the security policies at site level, possibly by the site security administrator, that set up a kind of "border control" between the site and the neighbouring network. We consider three sorts of policies: *remote communication*, *process migration*, and *name creation*. Each policy is related

to an action of the calculus, and we proceed by enumerating the names of the sites that are allowed to perform these actions. Therefore, we relate remote communication, process migration, and name creation with the ability to output, input, and create channels, respectively.

A *typing* is a mapping from site names to pairs of site types and site policies. *Site types* are mappings from channel, the free names of the site, to channel types. A channel type records the type of the argument and the sites where this channel may be used. We need this information to be able to check security policies. For instance, to type an output process $x\,!\,\langle v \rangle$ running at site s, we require that the sites of the channels that may instantiate x allow s to remote communicate with them. The val type has a unique value, $()$, and denotes a channel that carries no other channels.

The *tags*, i for *input*, o for *output*, and b for *both*, are part of a subtyping relation (inspired on the proposal of Pierce and Sangiorgi [10]) on the set of sites that may instantiate a given channel (defined in section 4). For example, this subtyping relation says that it is safe to use a channel of type $\mathrm{ch}(\gamma)@\{s,r\}^o$ whenever it is possible to use a channel of type $\mathrm{ch}(\gamma)@\{s\}^o$.

3 Semantics

The operational semantics of the calculus is presented following Milner *et al* [8]. We first define a *congruence relation* between processes and networks that simplifies the *reduction relation* introduced thereafter.

Free names. The notions of free and of bound names, as well as the substitution relation that lies on top of them, present some subtleties, introduced by located identifiers and by lexical scoping, which deserve some attention. Binders capture identifiers. The conceptual ideas behind bindings are the following.

- At network level, the binding of a located channel, entails the binding of free occurrences of not only a located channel anywhere in the network, but also the occurrences as a simple channel at its host site.

$$(\boldsymbol{\nu}\, a@r)\ \Big(s\mathbf{[}\cdot a\cdot a@r\cdot\mathbf{]}\,\|\,r\mathbf{[}\cdot a\cdot a@r\cdot\mathbf{]}\Big)$$

- At site level, the binding of a channel (simple or located) only binds the free occurrences (simple or located, respectively) of this channel.

$$s\Big[(\boldsymbol{\nu}\, a)\ (\cdot a\cdot a@s\cdot)\Big] \qquad s\Big[(\boldsymbol{\nu}\, a@s)\ (\cdot a\cdot a@s\cdot)\Big]$$

A paper on $lsd\pi$ presents complete definitions and explanations on free names (function fn not shown in this paper), on bound names, and on substitution [11].

Structural congruence. The *structural congruence relation* is the least congruence

1. $N \equiv M \qquad \text{if } N \equiv_\alpha M$
2. $((N \parallel M) \parallel M') \equiv (N \parallel (M \parallel M')), \quad (M \parallel N) \equiv (N \parallel M), \quad (N \parallel \mathsf{0}) \equiv N$
3. $((\boldsymbol{\nu}\, a@s)\ N) \parallel M \equiv (\boldsymbol{\nu}\, a@s)\ (N \parallel M) \qquad \text{if } a@s \notin \mathrm{fn}(M)$

   $(\boldsymbol{\nu}\, a@r)\ (\boldsymbol{\nu}\, b@s)\ N \equiv (\boldsymbol{\nu}\, b@s)\ (\boldsymbol{\nu}\, a@r)\ N$

   $(\boldsymbol{\nu}\, a@r)\, s_G[\![P]\!] \equiv s_G[\![(\boldsymbol{\nu}\, a@r)\ P]\!] \text{ if } (r \neq s \wedge s \in G_r(\mathrm{new})) \vee (r = s \wedge a \notin \mathrm{fn}(P))$

   $(\boldsymbol{\nu}\, a@s)\ s_G[\![P]\!] \equiv s_G[\![(\boldsymbol{\nu}\, a)\ P]\!] \qquad \text{if } a@s \notin \mathrm{fn}(P)$
4. $s_G[\![a@s\,!\,\langle v\rangle]\!] \equiv s_G[\![a\,!\,\langle v\rangle]\!], \quad s_G[\![a@s?(x:S)\ P]\!] \equiv s_G[\![a?(x:S)\ P]\!]$
5. $P \equiv Q \qquad \text{if } P \equiv_\alpha Q$
6. $((P \mid Q) \mid R) \equiv (P \mid (Q \mid R)), \quad (P \mid Q) \equiv (Q \mid P), \quad (P \mid \mathsf{0}) \equiv P$
7. $((\boldsymbol{\nu}\, u)\ P) \mid Q \equiv (\boldsymbol{\nu}\, u)\ (P \mid Q) \qquad \text{if } u \notin \mathrm{fn}(Q)$

   $(\boldsymbol{\nu}\, u)\ (\boldsymbol{\nu}\, u')\ P \equiv (\boldsymbol{\nu}\, u')\ (\boldsymbol{\nu}\, u)\ P$
8. $(\boldsymbol{\nu}\, a)\ \mathsf{0} \equiv \mathsf{0}$

Figure 2: Structural congruence on processes and networks.

relation closed under the rules in Figure 2 The first rules are fairly standard. Networks are congruent up to α-renaming; the parallel composition operator for networks is taken to be commutative and associative, with $\mathsf{0}$ being the neutral element.

Scope extrusion rules, however, deserve a more detailed analysis. In the third rule of group 3, if we are creating a remote channel ($r \neq s$), then the remote site r must grant permission to create the name. (the set G_r defines the security policies for site r.) When we create a local name, then there must not exist a simple channel with the same name in P. A similar concern is expressed in forth rule of the same group. The reason for these side conditions becomes clear in the following examples. These two networks should not be in the congruence relation

$$(\boldsymbol{\nu}\, a@s)\ s_G[\![a\,!\,\langle\cdot\rangle \mid a@s\,!\,\langle\cdot\rangle]\!] \not\equiv s_G[\![(\boldsymbol{\nu}\, a@s)\ a\,!\,\langle\cdot\rangle \mid a@s\,!\,\langle\cdot\rangle]\!]$$

since the (left-hand side) binder, at network level, binds both the simple channel, and the located channel, whereas the binder at process level only binds the located channel.

Similarly,

$$(\boldsymbol{\nu}\, a@s)\ s_G[\![a\,!\,\langle\cdot\rangle \mid a@s\,!\,\langle\cdot\rangle]\!] \not\equiv s_G[\![(\boldsymbol{\nu}\, a)\ a\,!\,\langle\cdot\rangle \mid a@s\,!\,\langle\cdot\rangle]\!]$$

the (right-hand side) binder, at site level, only binds the simple channel.

The last three rules rename located channels to simple channels (and vice-versa) when the channels are mentioned from their home site.

In $lsd\pi$ calculus there is also a rule for splitting and regrouping sites: $s_G[\![P]\!] \parallel s_G[\![Q]\!] \equiv s_G[\![P \mid Q]\!]$. However, in this version of the calculus, where sites are constant, the rule is covered by the migration rules RN-MIGI, RN-MIGO, and RN-MIGR ahead (see Figure 3).

Structural congruence on processes presents a similar set of rules when compared to structural congruence on networks. The only remark concerns rule 8 where "garbage collection" is only allowed locally. One could think that a more general rule also stands, say $(\boldsymbol{\nu}\, u)\ \mathsf{0}$, but this is not the case, since $\mathrm{fn}((\boldsymbol{\nu}\, u)\ \mathsf{0}) \neq \mathrm{fn}(\mathsf{0})$ when u is a located channel. Recall that s is free in $(\boldsymbol{\nu}\, a@s)\ \mathsf{0}$. Similar reasoning applies also to networks, where garbage collection would also have the malicious effect of erasing site policy annotations.

Reduction. We use contexts for processes and networks to simplify the reduction relation.

Definition 1 (Reduction contexts).

$$\begin{array}{lcl} E & ::= & [\cdot] \mid (E \mid P) \mid (\boldsymbol{\nu}\, u)\ E \\ F & ::= & [\cdot] \mid (F \parallel N) \mid (\boldsymbol{\nu}\, a@s)\ F \end{array}$$

When moving a process from a site to another one, we need to translate the free channels of that process to take into account the location the process will end up in: those local to the source site become remote, and those belonging to the target site become simple as they reached home.

Definition 2 (Translation of identifiers). *Let $A \subseteq \mathcal{N}$, $A \cap \mathcal{C} = \{a_1, \ldots, a_n\}$ and $A \cap \mathcal{C}@r = \{b_1@r, \ldots, b_m@r\}$. Then,*

$$\sigma_{rs}(P, A) = P\{a_1@s/a_1, \ldots, a_n@s/a_n, b_1/b_1@r, \ldots, b_m/b_m@r\}\,.$$

Notation: Let $P\sigma_{rs}$ abbreviate the result of applying a name translation σ_{rs} to the process P, affecting its free names: $P\sigma_{rs} = \sigma_{rs}(P, \mathrm{fn}(P))$.

Definition 3 (Reduction relation). *The rules in figure 3 inductively define the reduction relation on $lsd\pi$ terms.*

Rule RP-COMM is the *communication* rule of the calculus. It is defined only locally and it is the standard asynchronous π-calculus communication rule. Rule RP-COMR defines the communication for replicated inputs. Rules RN-MIGI, RN-MIGO, and RN-MIGR allow for processes to migrate across sites. When an input or output operation is carried out over a remote resource then, since communication only arises locally, the process migrates to the host site of the resource. In order to keep channels lexically

$$\text{RP-Comm}\quad a?(x:T)\ P \mid a\,!\,\langle v\rangle \to P[v/x]$$

$$\text{RP-Comr}\quad a?*(x:T)\ P \mid a\,!\,\langle v\rangle \to a?*(x:T)\ P \mid P[v/x]$$

$$\text{RN-Migo}\quad s_{G_1}[\![P]\!] \parallel r_{G_2}[\![a@s\,!\,\langle v\rangle \mid Q]\!] \to s_{G_1}[\![P \mid (a@s\,!\,\langle v\rangle)\sigma_{rs}]\!] \parallel r_{G_2}[\![Q]\!], \quad r \neq s$$

$$\text{RN-Migi}\quad s_{G_1}[\![P]\!] \parallel r_{G_1}[\![(a@s?(x:A)\ Q) \mid R]\!] \to s_{G_1}[\![P \mid (a@s?(x:A)\ Q)\sigma_{rs}]\!] \parallel r_{G_2}[\![R]\!], \quad r \neq s$$

$$\text{RN-Migr}\quad s_{G_1}[\![P]\!] \parallel r_{G_1}[\![(a@s?*(x:A)\ Q) \mid R]\!] \to s_{G_1}[\![P \mid (a@s?*(x:A)\ Q)\sigma_{rs}]\!] \parallel r_{G_2}[\![R]\!], \quad r \neq s$$

$$\text{RP-Cont}\quad \frac{P \to Q}{E[P] \to E[Q]} \qquad \text{RP-Str}\quad \frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

$$\text{RN-Site}\quad \frac{P \to Q}{s_G[\![P]\!] \to s_G[\![Q]\!]} \qquad \text{RN-Cont}\quad \frac{N \to M}{F[N] \to F[M]}$$

$$\text{RN-Str}\quad \frac{N \equiv N' \quad N' \to M' \quad M' \equiv M}{N \to M}$$

Figure 3: Reduction rules.

scoped, we use function σ to translate the free names of the migrating process. If we are moving from site r to site s, then σ_{rs} transforms the references to channels from r into located channels, since they will be mentioned from s, and makes references to channels from s into simple channels.

Rules RP-Str, and RN-Str introduce structural congruence into the reduction relation that are crucially used to bring processes and networks into the form requested by the left-hand-side of axioms RP-Comm, RN-Migo, RN-Migi, and RN-Migr. Finally, rules RP-Cont, RN-Site, and RN-Cont allow for reduction to happen within process and network contexts.

The next example illustrates a download of code from a server site srv requested by client site cl. Assume that $r \notin \mathrm{fn}(Q)$.

$$cl[\![(\boldsymbol{\nu}\, req)\ dl@srv\,!\,\langle req\rangle \mid req\,!\,\langle\rangle \mid P]\!] \parallel srv[\![(dl?*(r)\ r?()\ Q) \mid R]\!]$$

The client issues a new request to the dl (download) resource of the server by communicating a fresh channel req. The server upon the received request migrates process Q to the server using the acquired channel. Finally, the client fires the downloaded process. Reduction is as follows. Security annotations were deliberately omitted since they play no role in reduction.

$$cl[(\boldsymbol{\nu}\, req)\ dl@srv\,!\,\langle req\rangle \mid req\,!\,\langle\rangle \mid P] \parallel srv[dl?*(r)\ r?()\ Q \mid R] \quad \rightarrow$$

(RN-STR, RN-MIGO)

$$(\boldsymbol{\nu}\, req@cl)\ cl[req\,!\,\langle\rangle \mid P] \parallel srv[dl?*(r)\ r?()\ Q \mid R \mid dl\,!\,\langle req@cl\rangle] \quad \rightarrow$$

(RP-COMR)

$$(\boldsymbol{\nu}\, req@cl)\ cl[req\,!\,\langle\rangle \mid P] \parallel srv[dl?*(r)\ r?()\ Q \mid req@cl?()\ Q \mid R] \quad \rightarrow$$

(RN-MIGI)

$$(\boldsymbol{\nu}\, req@cl)\ cl[req\,!\,\langle\rangle \mid P \mid req?()\ Q\sigma_{srv,cl}] \parallel srv[dl?*(r)\ r?()\ Q \mid R] \quad \rightarrow$$

(RP-COMM, RN-STR)

$$cl[(\boldsymbol{\nu}\, req)\ Q\sigma_{srv,cl} \mid P] \parallel srv[dl?*(r)\ r?()\ Q \mid R]$$

4 Type system

The type system we present in this section enforces the user-defined security policies in $lsd\pi$ networks. We guarantee that, at runtime, well-typed networks do not violate the specified security policies.

Examples. In the following, we present some examples of erroneous networks that should be caught by the type system's sieve. Consider, in all examples, that sites denoted by $r, s,$ and t represent distinct locations.

A remote communication error occurs whenever an output to a located channel is performed from a site not belonging to the **rem** policy of the remote site. The next two examples elucidate this situation.

Example 1. Consider the network $s_{\{\mathrm{rem}:\{t\}\}}[P] \parallel r_{G_1}[a@s\,!\,\langle x\rangle]$. The output process running at site r is willing to send a remote message to site s; however, this action is not allowed, since r is not mentioned in the **rem** policy of s. The inclusion of r in the policies of site s fix the problem: $s_{\{\mathrm{rem}:\{t,r\}\}}[P] \parallel r_{G_1}[a@s\,!\,\langle x\rangle]$.

Example 2. To type-check the network $s_{\{\mathrm{rem}:\{r\}\}}[a?(x:\{r,t\})\ x\,!\,\langle c\rangle] \parallel r_{\emptyset}[a@s\,!\,\langle b\rangle]$ correctly, site s must be able to remote communicate both with site r, and site t. This is easily seen from the arguments of the input process at site s. Analysing the input continuation, $x\,!\,\langle c\rangle$, we conclude that the process could remote communicate with site r (and with site t as well), which does not concede any remote communication privilege at all. If r grants rem privileges to s, the network $s_{\{\mathrm{rem}:\{r\}\}}[a?(x:\{r,t\})\ x\,!\,\langle c\rangle] \parallel r_{\{\mathrm{rem}\{s\}\}}[a@s\,!\,\langle b\rangle]$ type-checks.

The control of code migration is performed using the policy keyword—**mig**—specifying which sites are allowed to upload code.

$$\gamma \leq \gamma \qquad \frac{\gamma_1 \leq \gamma_2 \quad \gamma_2 \leq \gamma_3}{\gamma_1 \leq \gamma_3} \qquad \frac{t \leq t \quad b \leq i \quad b \leq o \quad R \subseteq S}{S^o \leq R^o} \qquad \frac{t \leq t \quad b \leq i \quad b \leq o \quad S \subseteq R}{S^i \leq R^i} \qquad \frac{t \leq t \quad b \leq i \quad b \leq o \quad t \leq t'}{S^t \leq S^{t'}} \qquad \frac{\gamma_1 \leq \gamma_2 \quad S^t \leq R^{t'}}{\mathrm{ch}(\gamma_1)@S^t \leq \mathrm{ch}(\gamma_2)@R^{t'}}$$

Figure 4: Subtyping relation.

Example 3. The network $s_{\{\mathrm{mig}:\{t\}\}}[\![P]\!] \parallel r_{G_1}[\![a@s?(x : S)\ b\,!\,\langle x\rangle]\!]$ is rejected because site s denies migration of code from site r as it is intended by process $a@s?(x : S)\ b\,!\,\langle x\rangle$. Including r in the policies of site s overcomes the problem: $s_{\{\mathrm{mig}:\{t,r\}\}}[\![P]\!] \parallel r_{G_1}[\![a@s?(x : S)\ b\,!\,\langle x\rangle]\!]$

The creation of remote channels is controlled by the policy keyword **new**, and enumerating the sites authorised to create remote channels.

Example 4. The network $s_{\{\mathrm{new}:\{t\}\}}[\![P]\!] \parallel r_{G_1}[\![(\boldsymbol{\nu}\, a@s)\ \ a@s\,!\,\langle b\rangle]\!]$ fails to type check because site s denies creation of remote channels (as well as remote communications) from site r. Regardless the actions of process P, that we are not taking into account for the example, the network $s_{\{\mathrm{new}:\{t,r\},\mathrm{rem}:\{r\}\}}[\![P]\!] \parallel r_{G_1}[\![(\boldsymbol{\nu}\, a@s)\ \ a@s\,!\,\langle b\rangle]\!]$ is well typed .

The following example shows a more trickier situation resulting from code migration.

Example 5. Consider the network $s_G[\![b@r?(x : S)\ \ a@s\,!\,\langle x\rangle]\!] \parallel r_{\{\mathrm{mig}:\{s\}\}}[\![0]\!]$. The remote message $a@s\,!\,\langle x\rangle$ is going to run at site r, since it is the continuation of a process that migrates from s to r. Although r grants migration privileges to s, it does not allow for remote communications from s, and therefore the network should be rejected. Fix the security fault including $\mathrm{rem} : \{s\}$ into the policies for site r.

A different kind of communication error arises when a process is trying to pass on information about a non-reliable site. Considering a mailing system as an example, we can decide to grant a certain machine to deliver messages, but restrict the sites where these messages come from.

Example 6. The network $s_{\{\mathrm{rem}:\{r\}\}}[\![a?(x : \{t\})\ 0]\!] \parallel r_{G_1}[\![a@s\,!\,\langle b\rangle]\!]$ is illegal, since the output process, $a@s\,!\,\langle b\rangle$, at site r, is communicating information about a channel from site r, but the input process running at site s only admits information that mentions channels from t (indicated by $x : \{t\}$). Network $s_{\{\mathrm{rem}:\{r\}\}}[\![a?(x : \{t, r\})\ 0]\!] \parallel r_{G_1}[\![a@s\,!\,\langle b\rangle]\!]$ presents no security faults.

Subtyping. The binary relation $\leq$ on types is defined following Pierce and Sangiorgi [10], and is the least relation closed under the rules in figure 4. Intuitively, the subtyping relation allows for the inclusion of site identifiers (of where a channel can be

$$\text{SS-CHL}\quad \Gamma \vdash_s a@r : \Gamma(r)_1(a) \qquad\qquad \text{SS-CHS}\quad \Gamma \vdash_s a : \Gamma(s)_1(a)$$

Figure 5: Typing channels.

located in) when we perform outputs, and allows for the exclusion of site identifiers when we perform inputs. If a channel is used both for input and output its type is fixed. For example, considering that $\gamma_1 \leq \gamma_2$, then $\mathrm{ch}(\gamma_1)@\{s,r,t\}^o \leq \mathrm{ch}(\gamma_2)@\{s,r\}^o$, and $\mathrm{ch}(\gamma_1)@\{s\}^i \leq \mathrm{ch}(\gamma_2)@\{s,r,t\}^i$.

Tags i, o, and b denote, respectively, that a channel is used for input, for output, or for both input and output purposes. The relation is defined conventionally: covariant for inputs, contravariant for outputs, and invariant when a channel is used for inputs and outputs.

Typing. Figures 5, 6, and 7 present, respectively, the typing rules for channels, processes, and networks.

We record types for sites.[1] Therefore, the types of the free channels are kept within the types of the sites where they belong to. Types for located channels are directly fetched from their identifiers; to access types for simple channels we need extra information identifying their host site. We keep track of this information in the typing judgements for channels. In fact, the judgement $\Gamma \vdash_s v : \gamma$, means that if v is a simple channel, its host site is s. Of course, if v is located, it already has all the information needed for typing. Look up in figure 5 for the typing rules for channels.

Judgements for processes, $\Gamma \vdash_{s,S} P$, besides the identity of the current site s, record also the set of sites where P might be hosted at runtime. This is a crucial information for checking security policies because S give us the sites where events (remote communication, code migration, or channel creation) take place. We proceed by explaining the typing rules for processes that can be found in figure 6.

An output process $a@r\,!\,\langle v\rangle$ is well-typed if (1) the type of a, located at r, is a channel type having as arguments a supertype of the type of v, and if (2) site r allows any site where the output process might be located at runtime to remote communicate with it. Following is an example of an instance of the output rule

$$\frac{\begin{array}{cc}\Gamma \vdash_s v : \mathrm{ch}(\gamma)@\{s\}^b & \mathrm{ch}(\gamma)@\{s\}^b \leq \mathrm{ch}(\gamma)@\{s,r\}^i \\ \{t\} \subseteq \{s,t\} = \Gamma(r)_2(\mathrm{rem}) & \Gamma(r)_1(a) = \mathrm{ch}(\mathrm{ch}(\gamma)@\{s,r\}^i)@\{r\}^b\end{array}}{\Gamma \vdash_{s,\{t\}} a@r\,!\,\langle v\rangle}$$

On the other hand, if the output is performed over a simple channel a, we require every site where a may be located to give permission for remote communications from the sites where the process, $a\,!\,\langle v\rangle$, may be at runtime. It may not be possible

[1] A word on notation: let $\Gamma(s)_1$ and $\Gamma(s)_2$ be the first and second projections of the pair $\Gamma(s) = (\varphi, G)$.

$$\textsc{SP-Outl}\quad \frac{\Gamma(r)_1(a) = \mathrm{ch}(\gamma_1)@\{r\}^b \quad \Gamma \vdash_s v : \gamma_2 \quad \gamma_2 \leq \gamma_1 \quad S \subseteq \Gamma(r)_2(\mathrm{rem})}{\Gamma \vdash_{s,S} a@r\,!\,\langle v\rangle}$$

$$\textsc{SP-Outs}\quad \frac{\Gamma(s)_1(a) = \mathrm{ch}(\gamma_1)@R^b \quad \Gamma \vdash_s v : \gamma_2 \quad \gamma_2 \leq \gamma_1 \quad S \subseteq \Gamma(r)_2(\mathrm{rem}),\ \forall r \in R}{\Gamma \vdash_{s,S} a\,!\,\langle v\rangle}$$

$$\textsc{SP-Inpl}\quad \frac{\Gamma(r)_1(a) = \mathrm{ch}(\gamma_1)@\{r\}^b \quad \Gamma(s)_1(x) = \mathrm{ch}(\gamma_2)@T^b \quad \Gamma \vdash_{s,\{r\}} P \quad \mathrm{ch}(\gamma_2)@T^b \leq \gamma_1 \quad S \subseteq \Gamma(r)_2(\mathrm{mig})}{\Gamma \setminus x@s \vdash_{s,S} a@r?(x : T)\ P}$$

$$\textsc{SP-Inps}\quad \frac{\Gamma(s)_1(a) = \mathrm{ch}(\gamma_1)@R^b \quad \Gamma(s)_1(x) = \mathrm{ch}(\gamma_2)@T^b \quad \Gamma \vdash_{s,R} P \quad \mathrm{ch}(\gamma_2)@T^b \leq \gamma_1 \quad S \subseteq \Gamma(r)_2(\mathrm{mig}),\ \forall r \in R}{\Gamma \setminus x@s \vdash_{s,S} a?(x : T)\ P}$$

$$\textsc{SP-Nil}\quad \Gamma \vdash_{r,S} \mathsf{0} \qquad \textsc{SP-Par}\quad \frac{\Gamma \vdash_{s,S} P \quad \Gamma \vdash_{s,S} Q}{\Gamma \vdash_{s,S} (P \,|\, Q)}$$

$$\textsc{SP-Ress}\quad \frac{\Gamma(s)_1(a) = \mathrm{ch}(\gamma)@S^b \quad \Gamma \vdash_{s,S} P}{\Gamma \setminus a@s \vdash_{s,S} (\boldsymbol{\nu}\, a)\ P}$$

$$\textsc{SP-Resl}\quad \frac{\Gamma(r)_1(a) = \mathrm{ch}(\gamma)@\{r\}^b \quad \Gamma \vdash_{s,S} P \quad S \setminus s \subseteq \Gamma(r)_2(\mathrm{new})}{\Gamma \setminus a@r \vdash_{s,S} (\boldsymbol{\nu}\, a@r)\ P}$$

Figure 6: Typing processes.

to determine the site that hosts a channel at compile time (consider a communication over a channel received as a parameter). When typing a process $x\,!\,\langle v\rangle$ where the channel name x can be instantiated with either a simple channel or a located channel, one uses the rule SP-Outs. The sites that may send channels to instantiate x must be in the annotation of the input which introduces the parameter x.

To type an input process, $\Gamma \vdash_{s,S} a@r?(x : T)\ P$, we type P resolving simple channels to site s and considering P as running in r, since $a@r$ triggers the migration of P from s to r. Notation $\Gamma \setminus x@s$ denotes the removal of channel x, located at site s, from the typing environment Γ. Bear in mind that simple channels remain bound to s by lexical scoping. We check that the migration operation to r is allowed from every site where the process may be located at runtime. The following network type-checks,

$$s_{\emptyset}[a@r?(x : \{t\})\ x\,!\,\langle c\rangle] \parallel r_{\{\mathrm{mig}:\{s\}\}}[a\,!\,\langle b@t\rangle] \parallel t_{\{\mathrm{rem}:\{r\}\}}[\mathsf{0}]$$

The typing rules for the inaction process, the parallel process, and the creation of local channels are fairly standard. The creation of remote channels requires the

$$\textsc{SN-Nil}\quad \Gamma \vdash 0 \qquad\qquad \textsc{SN-Par}\quad \frac{\Gamma \vdash N \qquad \Gamma \vdash M}{\Gamma \vdash (N \parallel M)}$$

$$\textsc{SN-Net}\quad \frac{\begin{array}{c}\Gamma \vdash_{s,\{s\}} P \qquad\qquad \Gamma(s)_2 = G \\ \Gamma(s)_1(a) = \mathrm{ch}(\gamma)@\{s\}^b,\ \forall a \in \mathrm{dom}(\Gamma(s)_1)\end{array}}{\Gamma \vdash s_G[\![P]\!]}$$

$$\textsc{SN-Resl}\quad \frac{\begin{array}{c}\Gamma \vdash N \qquad\qquad S \setminus r \subseteq \Gamma(r)_2(\mathrm{new}) \\ S \text{ is the set of sites where } a@r \text{ occurs free in } N\end{array}}{\Gamma \setminus a@r \vdash (\boldsymbol{\nu}\, a@r)\ N}$$

Figure 7: Typing networks.

authorisation from the site where the channel is being created. The authorisation must be issued to all the sites in S.

The typing rules for networks can be found in figure 7. Rule SN-NET types a located process in a site, $s_G[\![P]\!]$. Process P must be well typed under type assumptions Γ, where simple channels are considered resources from site s, and the processes is running in s. Moreover, we demand that the network policies defined at network level match exactly the policies formulated in Γ—no process is allowed to forge security policies. The remaining premises assures that visible resources of the site, addressed only by simple channels, are indeed located at the site.

The handling of resource restrictions at network level is more delicate than at site level, since we lack information about the site that has created the resource. Therefore, we require that the site hosting the resource must concede creation permissions to every site that uses the resource. Notice in the following example that since $a@s$ is free in site r, site s must grant remote creation privileges to r.

$$(\boldsymbol{\nu}\, a@s)\ s_{\{\mathrm{new}:\{r\},\mathrm{rem}:\{r\}\}}[\![0]\!] \parallel r_{\emptyset}[\![a@s\,!\,\langle b\rangle]\!]$$

where $S = \{r\}$, $\Gamma(s) = \{(\emptyset, \{\mathrm{new} : \{r\}, \mathrm{rem} : \{r\}\})\}$, and $\Gamma(r) = \{(b : \mathrm{ch}(\gamma)@\{r\}^b, \emptyset)\}$.

5 Results

On the type system. Subtyping is a preorder.

Lemma 1. *The relation $\leq$ is a preorder.*

The type and subtype system rules are syntax oriented, so an algorithm to compute types (in polynomial time) can be found just by reading the rules backward. Notice that there are no recursive types.

Theorem 1 (Decidability of the type system). *Given Γ, and N, the problem of verifying whether $\Gamma \vdash N$ is decidable.*

Reduction preserves the typability of processes and of networks. This result uses a standard substitution lemma, together with subject congruence.

Lemma 2. *If* $\Gamma \vdash P$, *and* $P \equiv Q$, *then* $\Gamma \vdash Q$.

Proof. We proceed by induction on the type derivation, analysing the last rule applied. We just sketch case 7. (figure 2, page 8), where the last typing rule applied is SP-RESL, to illustrate the use of the side condition $s \in G_1(\mathrm{new})$ when $a@s \notin \mathrm{fn}(P)$. Consider the case where $(\boldsymbol{\nu}\, a@r)\ s_G[\![P]\!] \equiv s_G[\![(\boldsymbol{\nu}\, a@r)\ P]\!]$.

By hypothesis, $\Gamma \vdash (\boldsymbol{\nu}\, a@r)\ s_G[\![P]\!]$. Assuming $s \neq r$, we need to consider two subcases:

- $a@r \in \mathrm{fn}(P)$

  Therefore, by rule SP-RESL, $s \in \Gamma(r)_2(\mathrm{new})$ and so, by SP-RESL and SN-NET, we prove that $\Gamma \vdash s_G[\![(\boldsymbol{\nu}\, a@r)\ P]\!]$.

- $a@r \notin \mathrm{fn}(P)$

  In this case $s \notin S$, but the side condition $s \in G_1(\mathrm{new})$ ensures that r gives permission to create a local channel from s and so we can apply SP-RESL and SN-NET safely.

The case when $s = r$ is not relevant because it is always possible to create a channel in the site that hosts it. Notice that we exclude the current site when checking the sites that must grant the new policy. □

Theorem 2 (Subject reduction).

1. *If* $\Gamma \vdash_{s,S} P$, *and* $P \to Q$, *then* $\Gamma \vdash_{s,S} Q$.
2. *If* $\Gamma \vdash N$, *and* $N \to M$, *then* $\Gamma \vdash M$.

Proof. (sketch) By induction on the typing derivation of $\Gamma \vdash_{s,S} P$ and of $\Gamma \vdash N$. We proceed by case analysis on the reduction relation and examine the last typing rule of the typing derivation. The proof is straightforward. □

Runtime errors. Our type system guarantee that well-typed networks do not violate the specified security policies. In what follows we formalise the notion of runtime error.

Definition 4 (runtime errors). *Assume that* $r \neq s$. *Let* $\mathcal{E} = \{N | N \to^\star \nu u_1 \ldots \nu u_k (M' \parallel M)\}$, *and* M *of the form*

$$r_{G_1}[\![P]\!] \parallel s_{G_2}[\![a@r\,!\,\langle v\rangle]\!], \qquad s \notin G_1(\mathrm{rem}) \tag{1}$$

$$r_{G_1}[\![P]\!] \parallel s_{G_2}[\![a@r?(x:T)\ P]\!], \qquad s \notin G_1(\mathrm{mig}) \tag{2}$$

$$s_G[\![(a?(x:T)\ P) \mid a\,!\,\langle b@r\rangle]\!], \qquad r \notin T \tag{3}$$

$$r_{G_1}[\![P]\!] \parallel s_{G_2}[\![(\boldsymbol{\nu}\, a@r)\ P]\!], \qquad s \notin G_1(\mathrm{new}) \tag{4}$$

Networks that violate a security policy are in the set $\mathcal{E}$. The following result states that well-typed networks do not belong to $\mathcal{E}$.

Theorem 3 (type safety). *If* $\Gamma \vdash N$, *and* $N \rightarrow^{\star} M$, *then* $M \notin \mathcal{E}$.

Proof. The proof is straightforward and proceeds by absurd. □

6 Conclusions

Summary. We present a type system to control the migration of code in a concurrent distributed calculus where a network has a fixed number of sites that know each other. This setting fits DiTyCO, a distributed implementations of the language TyCO, which presently lacks resource access control. We monitor three security policies: remote communication, process migration, and channel creation, corresponding to the actions of the calculus, these policies enabling us to control code migration. The security policies are defined by the site administrators, following an intuitive and easy approach. For each site, its administrator specifies what operations other sites are allowed to perform. The type system checks if the processes running at those sites respect its security policies, and then checks if all the sites in the network will interoperate without violating each other policies. Specifically, we prove subject reduction, define runtime errors, and then prove type safety.

The current setting allow us to focus on the security policies for resources. We start with a subset of $lsd\pi$ calculus, with a fixed number of sites, and present a non-trivial yet simple and low-cost solution based on typing and subtyping relations. The system checks that processes running at given sites respect their security policies, and that sites in a network interoperate correctly. We prove subject reduction and type safety.

Further work. It is our understanding that this work settles the ground basis for further developments along two main directions: (a) the definition of security policies at resource level and therefore be able to refine the interaction between sites; (b) and the ability to adjust security policies dynamically.

We have already extended this system to deal with an arbitrary number of sites in networks with dynamic topology [6].

Related work. Other approaches to resource security in distributed mobile calculi comprise DPI [3, 4] and Klaim [2, 9]. See [1] for a general survey on concurrent mobile calculus, type systems, and security policies. DPI possesses an explicit objective construct to code—the **go** primitive. The control of migration is found along three aspects: a keyword **mig**, a subtype relation, and the ability to communicate site names. If a process "sees" the **mig** keyword as part of the type of a site, then it may migrate code to that site. The subtype relation, together with the capability to communicate site names, allows a site to tailor the information (*e.g.* resource names, control keywords) that the target site would be able to use. From a programming point of view, this approach does not seem very attractive since security annotations are spread along

the code and it is difficult to understand what actions are really allowed to execute. It is not clear how to implement type inference.

Klaim uses a capability type system to control operations on tuple spaces. Each site defines the actions that other sites can perform. There is a correspondence between the capabilities and the calculus's actions. For the migration primitive (*eval*) the type specifies also the security restrictions that the migrating process should obey. The Klaim approach is similar to ours in the sense that security policies are declared at site level, but differs substantially when we consider the way policies are programmed and checked. Notice that the Klaim type system is far more complex than ours is, although it provides roughly the same guarantees. One main distinction concerns the place where the security policies are defined: security policies in Klaim talk about what operations a site may perform on other sites, whereas in our framework each site talks about what actions it allows others to perform on it. From the site administrator point of view this looks more appropriate.

Moreover, our system is tailored to the particular aspects of lexical scoped settings.

Acknowledgements. This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (via CLC and the project MIMO, POSI/CHS/39789/2001), and by the EU FEDER (via CLC) and the EU IST proactive initiative FET-Global Computing (projects Mikado, IST–2001–32222, and Profundis, IST–2001–33100).

References

[1] G. Boudol, I. Castellani, F. Germain, and M. Lacoste. Models of distribution and mobility: State of the art. Mikado Deliverable D1.1.1, 2002.

[2] D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *Proc. of 30th ICALP'03*, volume 2719 of *LNCS*, pages 119–132. Springer-Verlag, 2003.

[3] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 2003.

[4] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Journal of Information and Computation*, 173:82–120, 2002.

[5] L. Lopes, A. Figueira, F. Silva, and V. Vasconcelos. A concurrent programming environment with support for distributed computations and code mobility. In *IEEE CLUSTER'00*, pages 297–306, 2000.

[6] F. Martins and V. Vasconcelos. Controlling security policies in a distributed environment. DI/FCUL TR 04–1, Department of Informatics, University of Lisbon, April 2004.

[7] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. Springer-Verlag, 1993.

[8] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, 1992.

[9] R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theoretical Computer Science*, 1(240):215–254, 2000.

[10] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[11] A. Ravara, A. Matos, V. Vasconcelos, and L. Lopes. Lexically scoping distribution: what you see is what you get. In *FGC'03*, volume 85(1) of *ENTCS*, July 2003.

[12] V. Vasconcelos, L. Lopes, and F. Silva. Distribution and mobility with lexical scoping in process calculi. In *HLCL'98*, volume 16 (3) of *ENTCS*. Elsevier Science Publishers, 1998.

Session II

Access Control

Model-checking Access Control Policies

Dimitar P. Guelev* Mark Ryan* Pierre Yves Schobbens**

Abstract

We present a model of access control which provides fine-grained data-dependent control, can express permissions about permissions, can express delegation, and can describe systems which avoid the root-bottleneck problem. We present a language for describing goals of agents; these goals are typically to read or write the values of some resources. We describe a decision procedure which determines whether a given coalition of agents has the means (possibly indirectly) to achieve its goal. We argue that this question is decidable in the situation of the potential intruders acting in parallel with legitimate users and taking whatever temporary opportunities the actions of the legitimate users present. Our technique can also be used to synthesise finite access control systems, from an appropriately formulated logical theory describing a high-level policy.

Introduction

In a world in which computers are ever-more interconnected, *access control systems* are of increasing importance in order to guarantee that resources are accessible by their intended users, and not by other possibly malicious users. Access control systems are used to regulate access to resources such as files, database entries, printers, web pages. They may also be used in less obvious applications, such as to determine whether incoming mail has access to its destination mailbox (spam filtering), or incoming IP packets to their destination computers (firewalls).

We present a model of access control which has among others the following features:

*School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK

**Institut d'Informatique, Facultés Universitaires de Namur, Rue Grandgagnage 21, 5000 Namur, Belgium

- Access control may be dependent on the data subject to control. This is useful in certain applications, such as the conference paper review system described below, or stateful firewalls, databases, etc. In [7], this is called *conditional authorisation*.
- Delegation of access control is easily expressed. This helps to avoid the root bottleneck, whereby root or the owner of a resource is required in order to make access control changes, and the insecurity caused by investing too much power in a single agent.
- Permissions for coalitions to act jointly can be expressed.

A key feature of our model is that *permissions* are functions of *state variables*, and therefore may change with the state. Because the ability to change the state is itself controlled by permissions, one can, in particular, express *permissions about permissions*. This allows us easily to devolve authority downwards, thus avoiding the root bottleneck, and to express delegation.

A potential problem of sophisticated access control systems, such as those which can be described using our model, is *indirect paths*. It might be that the system denies immediate access to a resource for a certain agent, but it gives the agent indirect possibilities by allowing it to manipulate permissions. Hence, there could be a sequence of steps which the agent can execute, in order to obtain access to the resource. We are interested in verifying access control systems to check whether such indirect paths exist.

Example 1 Consider a *conference paper review system*. It consists of a set of papers, and a set of agents (which may be authors, programme-committee (PC) members, etc). The following rules apply:

- The chair appoints agents (if they agree to it) to become PC members. PC members can resign unilaterally.
- The chair assigns papers for reviewing to PC members.
- PC members may submit reviews of papers that they have been assigned.
- A PC member a may read b's review of a paper, if the paper has not been assigned to a, or the paper has been assigned to a, and she has already submitted her own review.
- PC members may appoint sub-reviewers for papers which they have been assigned. Sub-reviewers may submit reviews of those papers. The PC member can withdraw the appointment of sub-reviewers.
- Authors should be excluded from the review process for their papers.

Each of these rules is a read access or a write access by one or more agents to a resource. We formalise this example in the next section, and use it as a running example through the paper. Statements 3 and 4 illustrate the dependency of write access and read access (respectively) on the current state. Statement 5 shows how permissions about permissions are important; here, the PC member has write permission on the data expressing the sub-reviewers' write permission on reviews.

Model checking such an access control system will answer questions such as: *can an author find out who reviewed her paper? Can a reviewer of a paper read someone else's review, before submitting his own?* We answer the second question in Example 9.

The main part of this paper presents a simple language for programming access, a propositional language for specifying access goals, and an accessibility operator which denotes that a given goal is achievable by means of a program in the programming language and can be used to formulate access control policies. We propose axioms which lead to the expressibility of this operator in propositional logic and to decision procedures for it. These procedures allow access control policies to be checked and behaviour that violates them to be proposed as counterexample to imperfect implementations of policies. Furthermore, the propositional expressibility of the accessibility operator entails that implementations of policies formulated with it can be automatically synthesised. We also show that it is decidable whether the execution of a certain program by one coalition provides another coalition with temporary opportunities that are sufficient for the achievement of a certain goal, given that the second coalition can interleave its actions with the actions of the first one. A Prolog implementation of one of the possible decision procedures for our accessibility operator (together with examples) is available on the web [8].

Structure of the paper. We first define our model of access control formally, show how Example 1 can be encoded in it and point to some properties of our model known to be important from the literature. Then we introduce the simple programming language which expresses the procedures that coalitions of agents can use to access systems and define a class of goals that can be pursued by coalitions of agents. For every concrete system it is decidable whether a coalition can achieve a given goal of this class by running a program. We argue that the techniques developed in detail for the simple programming language can be straighforwardly extended to languages based on high-level access actions. In the concluding section we explain how these techniques lead to algorithms for model checking access control policies on existing systems and synthesising systems which implement given policies. In Appendix A we describe an alternative way to decide the achievability of goals by model-checking appropriate μ-calculus formulas.

1 Access control systems

We denote the set of propositional formulas φ built using the variables p from some given vocabulary P by $\mathbf{L}(P)$. We adopt $\Rightarrow$ and $\bot$ as basic in the construction of these formulas and regard $\top$, $\neg$, $\wedge$, $\vee$ and $\Leftrightarrow$ as abbreviations. We denote the set of the variables occurring in a formula $\varphi \in \mathbf{L}(P)$ by $\mathrm{Var}(\varphi)$.

Definition 2 An *access control system* is a tuple $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$, where P is a set of propositional variables as above, Σ is a set of *agents*, and $\mathtt{r}$ and $\mathtt{w}$ are mappings of type $P \times \mathcal{P}_{\mathit{fin}}(\Sigma) \to \mathbf{L}(P)$, where $\mathcal{P}_{\mathit{fin}}(\Sigma)$ stands for the set of the finite subsets of Σ. The mappings $\mathtt{r}$ and $\mathtt{w}$ are required to satisfy

$$A' \subset A \text{ implies } \vdash \mathtt{r}(p, A') \Rightarrow \mathtt{r}(p, A) \text{ and } \vdash \mathtt{w}(p, A') \Rightarrow \mathtt{w}(p, A). \qquad (1)$$

The requirement (1) reflects that a coalition A has the abilities of all of its sub-coalitions A'.

The state of an access control system $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w} \rangle$ is determined by the truth values of the variables $p \in P$, denoted by 0 and 1. States are models for $\mathbf{L}(P)$ as a propositional logic language. We represent the states of S by the subsets of P, $s \subseteq P$ representing the state at which the variables which evaluate to 1 are those in s. We denote the truth value of formula φ at state s by $\varphi(s)$. Truth values of formulas are defined in the usual way.

Given $p \in P$, $A \subset_{\mathit{fin}} \Sigma$ and $s \subseteq P$, coalition A has the right to read or overwrite p at state s iff $\mathtt{r}(p, A)(s) = 1$ or $\mathtt{w}(p, A)(s) = 1$, respectively. The definitions of $\mathtt{r}$ and $\mathtt{w}$ are assumed to be known to all agents $a \in \Sigma$. Agents, however, may lack the permission to access variables in the formulas that $\mathtt{r}$ and $\mathtt{w}$ produce, and therefore be unable to decide what is permitted at certain states.

Example 3 Consider the Conference paper review system again. Let $\mathtt{Papers}$ and $\mathtt{Agents}$ be fixed sets, let the function

$$\mathtt{author} : \mathtt{Papers} \times \mathtt{Agents} \to \{\bot, \top\}$$

be fixed, and the constant $c : \mathtt{Agents}$ denote the chairperson of the programme committee. Let P contain the variables

$\mathtt{pcmember}(a)$	a is a PC member
$\mathtt{reviewer}(p, a)$	paper p is assigned to PC member a
$\mathtt{subreviewer}(p, a, b)$	paper p is assigned to sub-reviewer b by PC member a
$\mathtt{submittedReview}(p, a)$	a review of p has been submitted by sub-reviewer a
$\mathtt{review}(p, a)$	the review of p from sub-reviewer a

for each $a \in \mathtt{Agents}$ and $p \in \mathtt{Papers}$. Let $\mathtt{pcmember}(c)$ hold (initially) and $\mathtt{r}$ and $\mathtt{w}$ be defined as follows:

$\mathtt{pcmember}(.)$ The set of PC members is known to everyone.
$\mathtt{r}(\,\mathtt{pcmember}(a),\, A\,) \rightleftharpoons \top$.
A PC member may be appointed by a joint action of the chair and the candidate, and may resign unilaterally:
$\mathtt{w}(\,\mathtt{pcmember}(a),\, A\,) \rightleftharpoons \{a, c\} \subseteq A \vee (a \in A \wedge \mathtt{pcmember}(a))$.

Here and below we use definition schemata, which become well-formed formulas over P when the agents and coalitions occurring in them become instantiated. In particular, $\mathtt{author}(p, a)$ stands for a propositional constant for each pair $p \in \mathtt{Papers}$, $a \in \mathtt{Agents}$.

$\mathtt{reviewer}(.,.)$ Reviewers are known to the PC members, except the authors of the respective paper:
$\mathtt{r}(\mathtt{reviewer}(p, a), \{x\}) \rightleftharpoons \mathtt{pcmember}(x) \wedge \neg\mathtt{author}(p, x)$
The chairperson c may assign a paper p to a PC member a who is not an author of p, if a accepts. Both c and a may resign reviewership of p unilaterally, unless a has already assigned p to a sub-reviewer:

$\mathtt{w}(\mathtt{reviewer}(p,a),A) \rightleftharpoons$

$$\left(\begin{array}{c}(\mathtt{pcmember}(a) \wedge \{a,c\} \subseteq A \wedge \neg\mathtt{author}(p,a) \wedge \neg\mathtt{reviewer}(p,a)) \vee \\ ((a \in A \vee c \in A) \wedge \mathtt{reviewer}(p,a) \wedge \neg \bigvee_{b \in \mathtt{Agents}} \mathtt{subreviewer}(p,a,b))\end{array}\right)$$

$\mathtt{subreviewer}(.,.,.)$ The review status of a paper is known to PC members who are not authors of the paper, and to the respective sub-reviewers:
$\mathtt{r}(\mathtt{subreviewer}(p,a,b),\{x\}) \rightleftharpoons (\mathtt{pcmember}(x) \wedge \neg\mathtt{author}(p,x)) \vee x = b$
A reviewer a may assign a paper p to at most one sub-reviewer b, who is not an author of p, and has not been assigned p by another reviewer. (To review p personally, a must become his/her own sub-reviewer.) A reviewer may revoke sub-reviewership, and a sub-reviewer may resign, unless a review has already been submitted:
$\mathtt{w}(\mathtt{subreviewer}(p,a,b),A) \rightleftharpoons$

$$\left(\begin{array}{c}\left(\begin{array}{c}\{a,b\} \subseteq A \wedge \mathtt{reviewer}(p,a) \wedge \neg\mathtt{author}(p,b) \wedge \\ \neg \bigvee_{d \in \mathtt{Agents}} (\mathtt{subreviewer}(p,a,d) \vee \mathtt{subreviewer}(p,d,b))\end{array}\right) \vee \\ (\mathtt{subreviewer}(p,a,b) \wedge (a \in A \vee b \in A) \wedge \neg\mathtt{submittedReview}(p,b))\end{array}\right)$$

$\mathtt{submittedReview}(.,.)$ Whether a review on a paper has been submitted is known to PC members, except the authors of the paper:
$\mathtt{r}(\mathtt{submittedReview}(p,a),\{x\}) \rightleftharpoons \mathtt{pcmember}(x) \wedge \neg\mathtt{author}(p,x)$
A subreviewer may submit a review once. (This makes the current value of $\mathtt{review}(p,a)$ final.)
$\mathtt{w}(\mathtt{submittedReview}(p,a),\{x\}) \rightleftharpoons$

$$x = a \wedge \bigvee_{b \in \mathtt{Agents}} \mathtt{subreviewer}(p,b,x) \wedge \neg\mathtt{submittedReview}(p,x)$$

$\mathtt{review}(.,.)$ PC member a can read reviews of a paper p, provided a is not its author and does not have a review outstanding for p.
$\mathtt{r}(\mathtt{review}(p,a),\{x\}) \rightleftharpoons$

$$\left(\begin{array}{c}\mathtt{pcmember}(x) \wedge \neg\mathtt{author}(p,x) \wedge \mathtt{submittedReview}(p,a) \wedge \\ (\bigvee_{b \in \mathtt{Agents}} \mathtt{subreviewer}(p,b,x) \Rightarrow \mathtt{submittedReview}(p,x) \vee x = a)\end{array}\right)$$

A sub-reviewer may update the contents of his review until he/she makes it final by setting $\mathtt{submittedReview}$ to 1:
$\mathtt{w}(\mathtt{review}(p,a),\{x\}) \rightleftharpoons$

$$x = a \wedge \bigvee_{b \in \mathtt{Agents}} \mathtt{subreviewer}(p,b,x) \wedge \neg\mathtt{submittedReview}(p,x)$$

The formulas $\mathtt{r}(.,.)$ and $\mathtt{w}(.,.)$ in 2-4 which are defined about singleton coalitions extend to bigger coalitions by monotonicity.

The purpose of this example is to illustrate our model and syntax. It becomes clear in Example 9 that the design of the system specified above is not flawless. It admits violating some well-established practices of conference management.

We extend $\mathtt{r}$ to a mapping from $\mathbf{L}(P) \times 2^{\Sigma}$ to $\mathbf{L}(P)$ by putting $\mathtt{r}(\varphi, A) \rightleftharpoons \bigwedge_{p \text{ occurs in } \varphi} \mathtt{r}(p, A)$.

An access control system $\langle P, \Sigma, \mathtt{r}, \mathtt{w} \rangle$ is *finite*, if P and Σ are finite. In this paper we study finite access control systems. We only consider systems whose resources are sets of boolean variables; for example, the review of a paper was represented as a boolean, which is more crude than the reviews from most conferences.

1.1 Comparison with other models

Several formal models of access control have been published. The influential early work [9] proposed a model for access control with a matrix containing the current rights of each agent on each resource in the modelled system. The actions allowed include creating and destroying agents and resources and updating the matrix of the access rights. The possibility to carry out an action is defined in terms of the rights as described in the matrix. Given the generality of that model, it is not surprising that the problem of whether an agent can gain access to a resource, called the *safety problem*, is not decidable. This can be largely ascribed to the possibility to change the sets of agents and resources in the model. In our model, the sets of agents and resources are fixed.

The formulas $\mathtt{r}(p, A)$ and $\mathtt{w}(p, A)$ may be considered as the values of the cells of an access matrix

	...	Coalition A	...
...	...	...	...
Resource p	...	$\mathtt{r}(p, A),\ \mathtt{w}(p, A)$	...
...	...	...	...

which for each particular state s of the modelled system corresponds to a matrix of the form from [9] describing the rights of reading and writing at that state. Unlike [9], entries in the matrix are updated by actions specifically for that purpose, whereas in our model coalitions update *general purpose* state variables, which in turn affect the value of the formulas $\mathtt{r}(.,.)$ and $\mathtt{w}(.,.)$. This allows the modelling of *automatic* dependencies between the contents of the access control system, if viewed as a database, and the rights of its users. The special case in which every particular right can be manipulated by a dedicated action can be modelled in our system by choosing a dedicated propositional variable $q_{\mathtt{x},p,A}$ for each triple $\mathtt{x} \in \{\mathtt{r}, \mathtt{w}\}$, $p \in P$ and $A \subseteq \Sigma$ and defining $\mathtt{x}(p, A)$ as $q_{\mathtt{x},p,A}$. Then changing the right $\mathtt{x}$ of coalition A on p can be made independently for each triple $\mathtt{x}, p, A$. In this case, however, special care needs to be taken to avoid infinite digressions like $q_{\mathtt{x},p,A}$, $q_{\mathtt{y},q_{\mathtt{x},p,A},B}$, $q_{\mathtt{z},q_{\mathtt{y},q_{\mathtt{x},p,A},B},C}$, ...

An analysis of formal models is given in [7]. Desirable properties highlighted in the literature include:

- *Conditional authorisations* [7]. Protection requirements may need to depend on the evaluation of conditions. As shown by the example above, this is a central feature of our model.
- *Expressibility of joint action* [10, 1]. Some actions require to be executed jointly by a coalition of agents, such as the appointment of an agent to the programme committee in the example above, which requires the willingness both of the chair and the candidate.
- *Delegation mechanisms*. In particular, permission to delegate a privilege should be independent of the privilege [3]. Delegation mechanisms may be classified according to permanence, transitivity and other criteria [4].
- *Support for open and closed systems* [7]. In open systems, accesses which are not specified as forbidden are allowed. Thus, the default is that actions are allowed. In closed systems, the default is the opposite: actions which are not expressly allowed are forbidden.
- *Expressibility of administrative policies* [7]. Administrative policies specify who may add, delete, or modify the permissions of the access control system. The are "one of the most important, although less understood" aspects of access control, and "usually receive little consideration" [7]. In our model, they are fully integrated, as the conference paper review example shows.
- *Avoidance of root bottleneck.* Called 'separation of duty' in [7], this property refers to the principle that no user should be given enough privilege to misuse the system on their own. Models should facilitate the design of systems having this property.
- *Support for fine-and coarse-grained specifications* [7]. Fine-grained rules may refer to specific individuals and specific objects, and these should be supported. But allowing *only* fine-grained rules would make a model unusable; some coarse-grained mechanisms such as roles must also be supported. Our model supports fine-grained rules. It relies on a higher-level language such as the language of predicates used in the example above to express coarse-grained rules.

Our model satisfies all these properties, except the last one. It is not meant to be a language for users. It represents a low-level model of access control, which we can use to give semantics to higher-level languages such as RBAC [12], OASIS [2], and the calculus of [1].

2 Programs in systems with access control

In this section we introduce a simple language which can be used to program access to systems as we described above. Programs α in it have the syntax

$$\alpha ::= \texttt{skip} \mid p{:=}\varphi \mid \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \alpha \mid (\alpha;\alpha) \tag{2}$$

and the usual meaning. It can be shown that adding a *loop* statement, e.g. $\texttt{while}\ \varphi\ \texttt{do}\ \alpha$ to this language would have no effect on its ultimate expressive power. This follows from our choice to model only finite state systems. We do not include loops in (2), because our concern is the mere existence of programs with certain properties.

2.1 Semantics of programs

We define the semantics of programs in (2) as functions from states to states. This can be regarded as a *denotational semantics* for (2), as known from the literature (see, e.g., [11]). The ingredient of this semantics that is specific and most important to our study is a mapping describing executability of programs as the subject of access restrictions. The notation below is introduced to enable the concise definition of the semantics. Let $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ be a fixed access control system for the rest of this section.

Definition 4 Substitutions are functions of the type $P \to \mathbf{L}(P)$. We record substitutions f in the form $[f(p)/p : p \in P]$. We often write $[f(p)/p : p \in Q]$ where $Q \subset P$ to denote $\lambda p.\mathbf{if}\ p \in Q\ \mathbf{then}\ f(p)\ \mathbf{else}\ p$. If $Q = \{p_1, \ldots, p_n\}$, then we sometimes denote $[f(p)/p : p \in Q]$ by $[f(p_1)/p_1, \ldots, f(p_n)/p_n]$.

A substitution f is extended to a function of type $\mathbf{L}(P) \to \mathbf{L}(P)$ by the clauses $f(\bot) = \bot$ and $f(\varphi \Rightarrow \psi) = f(\varphi) \Rightarrow f(\psi)$. We omit the parentheses in $f(\varphi)$ for $\varphi \in \mathbf{L}(P)$. Given substitutions f and g, fg denotes $[fg(p)/p : p \in P]$. $\exists p\varphi$ stands for $[\bot/p]\varphi \vee [\top/p]\varphi$. $\forall p\varphi$ stands for $\neg\exists p\neg\varphi$. If $\mathrm{Var}(\varphi) = \{p_1, \ldots, p_n\}$, then $\exists\varphi$ and $\forall\varphi$ stand for $\exists p_1 \ldots \exists p_n\varphi$ and $\forall p_1 \ldots \forall p_n\varphi$, respectively.

Let $\mathbf{P}$ be the set of all programs in P. The function $[\![.]\!] : \mathbf{P} \to (P \to \mathbf{L}(P))$ is defined by the clauses:

$$\begin{array}{rcl}
[\![\mathtt{skip}]\!] & = & [p/p : p \in P] = [\,] \\
[\![p := \varphi]\!] & = & [\varphi/p] \\
[\![\mathtt{if}\ \varphi\ \mathtt{then}\ \alpha\ \mathtt{else}\ \beta]\!] & = & [(\varphi \wedge [\![\alpha]\!](p)) \vee (\neg\varphi \wedge [\![\beta]\!](p))/p : p \in P] \\
[\![(\alpha;\beta)]\!] & = & [\![\alpha]\!][\![\beta]\!]
\end{array}$$

Proposition 5 *If S grants all the access α attempts, then the run of α from state $s \subseteq P$ takes S to state $\{p : ([\![\alpha]\!](p))(s) = 1\}$.*

Every particular step of the execution of a program can be carried out only if the respective coalition has the necessary access rights. E.g., for an assignment $p := \varphi$ to be executed, the coalition needs the right to overwrite p and read the variables occurring in φ. We define this by means of the function $[\![.,.]\!] : 2^{\Sigma} \times \mathbf{P} \to \mathbf{L}(P)$. $[\![A, \alpha]\!]$ evaluates to a formula which expresses whether the coalition A may execute

the program α. $[\![.,.]\!]$ is defined by the clauses:

$$\begin{array}{rcl} [\![A, \mathtt{skip}]\!] & = & \top \\ [\![A, p := \varphi]\!] & = & \mathtt{r}(\varphi, A) \wedge \mathtt{w}(p, A) \\ [\![A, \mathtt{if}\ \varphi\ \mathtt{then}\ \alpha\ \mathtt{else}\ \beta]\!] & = & \mathtt{r}(\varphi, A) \wedge (\varphi \Rightarrow [\![A, \alpha]\!]) \wedge (\neg\varphi \Rightarrow [\![A, \beta]\!]) \\ [\![A, (\alpha; \beta)]\!] & = & [\![A, \alpha]\!] \wedge [\![\alpha]\!][\![A, \beta]\!] \end{array}$$

Proposition 6 *S will grant coalition $A \subseteq \Sigma$ to execute program α from state s iff $[\![A, \alpha]\!](s) = 1$.*

Despite its ultimate simplicity, the language (2) can describe every deterministic and terminating algorithm for access to a system the considered type, as long as it is assumed that a failed access attempt can only bring general failure, and cannot be used to, e.g., draw conclusions on the state of a system for the purpose of further action. This restriction can be lifted. See the more general setting outlined in Subsections 4.2 and 4.3.

2.2 Programs which obtain access

Let $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ be a fixed access control system again, and let $\mathbf{P}$ be the set of programs (2) in the vocabulary P. Given a state $s \subseteq P$ and a $p \in P$, the truth values $\mathtt{r}(p, A)(s)$ and $\mathtt{w}(p, A)(s)$ indicate whether A can read and write p, respectively, *in state s*. However, it may be that A currently does not have some permission, but that A can change the state in order to obtain it. In this section we define $\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$, which denote A's ability to read/write φ by a possibly lengthy sequence of steps. Such sequences can be encoded as programs of the form (2). The ultimate ability for A to obtain the truth value of $\varphi \in \mathbf{L}(P)$ can be understood as the ability of A to run a program $\alpha \in \mathbf{P}$ that works out the value of φ and copies it into some variable p_0 such that $\mathtt{r}(p_0, A) = \mathtt{w}(p_0, A) = \top$. It can be expressed in terms of $[\![\alpha]\!]$ and $[\![A, \alpha]\!]$ as follows:

$$\mathtt{R}_A\varphi \rightleftharpoons (\exists\alpha \in \mathbf{P})\forall([\![A, \alpha]\!] \wedge ([\![\alpha]\!](p_0) \Leftrightarrow \varphi)) \tag{3}$$

Similarly, the ability of A to drive the system into a state where some $\varphi \in \mathbf{L}(P)$ has a truth value of A's choosing, can be expressed by the formula

$$\mathtt{W}_A\varphi \rightleftharpoons (\exists\alpha_\top, \alpha_\bot \in \mathbf{P})\forall([\![A, \alpha_\top]\!] \wedge [\![A, \alpha_\bot]\!] \wedge [\![\alpha_\top]\!]\varphi \wedge [\![\alpha_\bot]\!]\neg\varphi) \tag{4}$$

The universal closures $\forall$ in (3) and (4) express that α, $\alpha_\top$ and $\alpha_\bot$ are runnable and produce the stated results from all initial states. Note that $\mathtt{R}_A$ and $\mathtt{W}_A$ allow for destructive behaviour of the programs involved. Obtaining the desired goal may involve changing the state, possibly in a way which A cannot undo. In the next section, we consider a more expressive goal language in which we restrict the search to programs which are not destructive.

The formulas (3) and (4) determine the ability of A to execute a program which would achieve the goal of reading or writing φ. Quantifier prefixes like $(\exists\alpha \in \mathbf{P})$ make it hard to evaluate (3) and (4) directly. However, if S is finite, these formulas have purely propositional equivalents, and therefore can be computed mechanically, because there are only finitely many different programs in the vocabulary P modulo semantical equivalence. Of course, the enumerating all these programs in order to evaluate $(\exists\alpha \in \mathbf{P})$ is very inefficient. In Section 3 we treat $\mathtt{R}_A$ and $\mathtt{W}_A$ as special cases of a more general accessibility operator. In an appendix of [6] we describe a way to evaluate this operator, and consequently, $\mathtt{R}_A$ and $\mathtt{W}_A$, without resorting to quantifier prefixes of the form $(\exists\alpha \in \mathbf{P})$, which is more efficient.

3 A general accessibility operator

Extracting information and driving a system into a state with some desired property are only the simplest goals of access. One goal cannot be treated without regard for others, because achieving a goal may have destructive side effects which prevent another goal from being achieved. That is why achieving composite goals sometimes needs to be planned with all their subgoals in mind at the same time. In this section, we consider a language for describing more refined kinds of access. Our language allows us to express boolean combinations of goals. Expressible goals include preserving the truth value of some formulas while reading or setting the truth values of others. Preservation is understood as restoring the original value of the formula in question upon the end of activities, and not necessarily keeping the value constant throughout the run of a program.

The accessibility operator in this language is written in the form $A(\Phi, \psi)$ where A is a coalition, Φ is a list of formulas in $\mathbf{L}(P)$ that A wants to read, and ψ is a *goal formula* with the syntax

$$\psi ::= \bot \mid \top \mid \mathbf{p} \mid \psi \wedge \psi \mid \psi \vee \psi \tag{5}$$

where $\mathbf{p}$ denotes an atomic goal of one of the following forms:

- $\overline{\varphi}'$, where $\varphi \in \mathbf{L}(P)$; this is the goal of making φ true.
- $\overline{\varphi}$, where $\varphi \in \mathbf{L}(P)$; this is the goal of "realising" that φ is true.

$\top$ and $\bot$ stand for a trivial goal, which calls for no action, and an unachievable goal, respectively. The goal $\psi_1 \vee \psi_2$ is regarded as achieved if either ψ_1 or ψ_2 are. The goal $\psi_1 \wedge \psi_2$ is achieved if both ψ_1 and ψ_2 are. Atomic goals of the form $\overline{\varphi}$ may fail even if A manages to obtain the truth value of φ, in case it turns out to be 0. On the other hand a goal of the form $\overline{\varphi} \vee \overline{\neg\varphi}$ can be assumed achieved without any action.

Example 7 The expression $A(\langle p\rangle,\ (\overline{q} \wedge \overline{q}') \vee (\overline{\neg q} \wedge \overline{\neg q}'))$ denotes that A wants to read p and *preserve* the truth value of q. If $\mathtt{r}(p, A) = q$ and $\mathtt{r}(q, A) = \mathtt{w}(q, A) = \top$, then A can achieve its goal by means of the program

$$\mathtt{if}\ q\ \mathtt{then}\ p_0{:=}p\ \mathtt{else}\ (q{:=}\top;\ p_0{:=}p;\ q{:=}\bot)$$

where p_0 is a variable dedicated to storing the value of p. Note that the program restores the value of q after temporarily setting it to 1 in order to gain access to p in the else clause of the conditional statement. The goal described by the simpler expression $A(\langle p\rangle, \top)$, which does not require q to be restored, can be achieved by the simpler program $(q{:=}\top;\ p_0{:=}p)$.

The formula ψ in $A(\Phi, \psi)$ can express an arbitrary relation $R(\overline{p_1}, \ldots, \overline{p_n}; \overline{p_1}', \ldots, \overline{p_n}')$ between the initial values $\overline{p_1}, \ldots, \overline{p_n}$ and the final values $\overline{p_1}', \ldots, \overline{p_n}'$ of the variables $p_1, \ldots, p_n$ of the system as a requirement for A to satisfy. The main difficulty in implementing the relation R in our setting is not in computing R, but to the planning of the actions needed to access the variables.

Example 8 Let $P = \{p_1, p_2, p_3\}$, $A \subseteq \Sigma$, $\mathtt{r}(p_1, A) = \neg p_2$, $\mathtt{w}(p_1, A) = p_2$, $\mathtt{r}(p_2, A) = \mathtt{w}(p_2, A) = \top$, $\mathtt{r}(p_3, A) = p_1$ and $\mathtt{w}(p_3, A) = \neg p_1$. *From any state, can A achieve a state in which the value of p_3 is inverted?* Yes; for example, this program samples the variables in order to determine what it can do, and inverts the value of p_3. A can run it from any state.

```
if p2 then (
    p1:=⊤;
    if p3 then (p1:=⊥; p3:=⊥) else (p1:=⊥; p3:=⊤)
)
else if p1 then
    if p3 then (p2:=⊤; p1:=⊥; p3:=⊥) else (p2:=⊤; p1:=⊥; p3:=⊤)
else (
    p2:=⊤; p1:=⊤;
    if p3 then (p1:=⊥; p3:=⊥) else (p1:=⊥; p3:=⊤)
)
```

The program (except for the formatting) was produced by our implementation [8].

In general, the goal $A(\Phi, \psi)$ expresses the ability of the coalition A to execute a program which reads the values of formulas in Φ, while changing the values of formulas in order to make the relation represented by ψ hold. The simple goals expressed by $\mathtt{R}_A\varphi$ and $\mathtt{W}_A\varphi$ can be expressed in this language:

$$\mathtt{R}_A\varphi \Leftrightarrow A(\{\varphi\}, \top), \qquad \mathtt{W}_A\varphi \Leftrightarrow A(\emptyset, \overline{\varphi}') \wedge A(\emptyset, \overline{\neg\varphi}').$$

In the appendix of [6] we show that the possibility (for A) to achieve $A(\Phi, \psi)$ can be decided mechanically and, if $A(\Phi, \psi)$ is achievable, a program which can be used (by A) to achieve it can be synthesised.

To demonstrate this, we add the superscripts V, T, K to goal expressions. $A^{V,T,K}(\Phi, \psi)$ expresses the existence of a program α which A can execute to read the formulas from Φ and enforce the relation represented by ψ, *provided that the initial state s of the system satisfies $s \cap V = T$* and *without going through any of the states in the list*

of states K. We use the superscript triple V, T, K to express achievability of *sub*goals which can arise after some action that brings partial knowledge of the state of the system has already been taken. The list K is used to prevent considering moving to states which have already been explored. Now the original form $A(\Phi, \psi)$ can be viewed as the special case $A^{\emptyset,\emptyset,\emptyset}(\Phi, \psi)$, in which nothing is assumed about initial states. Further details are given in an appendix of [6].

Sometimes goals involve enabling the achievement of further goals. A natural way to formulate and to enable reasoning about such goals is to allow *nested occurrences* of the accessibility operator A in goal formulas ψ:

$$\psi ::= \bot \mid \top \mid \mathbf{p} \mid A(\Phi, \psi) \mid \psi \wedge \psi \mid \psi \vee \psi \tag{6}$$

Example 9 For the conference paper review system, the question of whether reviewer a of paper p can read reviewer b's review before submitting his own, may be written as:

$$\{a, b, c\}(\emptyset, \overline{\texttt{submitted}(p, b) \wedge \neg \texttt{submitted}(p, a)}' \wedge \{a\}(\langle \texttt{review}(p, b) \rangle, \overline{\texttt{submitted}(p, a)}')).$$

This formula asks: *is it possible for a, b and the chair c to reach a state s in which b has submitted his review of p but a has not yet submitted hers, and from there a may read b's review and then submit hers?* If this formula holds, we can synthesise a program for $\{a, b, c\}$ to enable $\{a\}$ to achieve $(\langle \texttt{review}(p, b) \rangle, \overline{\texttt{submitted}(p, a)}')$ from such an s. Surprisingly, the answer is "yes". PC member a can read b's review, then become appointed a subreviewer by c and submit her own review.

Since we define the achievability of a goal by a coalition as its ability to plan its actions for achieving the goal in the form of a program, one coalition can enable another coalition to achieve a goal by taking the system to a state which allows the second coalition to achieve the goal and, most importantly, passing the second coalition the knowledge of this state needed to justify its plan for achieving the goal. If Φ is the empty list $\langle\rangle$, then $A^{V,T,K'}(\Phi, B(\Phi', \psi'))$ means that A can reach a state s in which A's knowledge of s will be sufficient for B to achieve (Φ', ψ'). In case $\Phi' \neq \langle\rangle$, we assume that it is possible to achieve $A^{V,T,K'}(\Phi, B(\Phi', \psi'))$ by (I) *A sharing with B its knowledge* of a reached s described by appropriate V and T upon passing the control to B and then (II) B reading the formulas from Φ *for* A. That is why we have

$$B^{V,T,\{T\}}(\Phi' * \Phi, \psi') \Rightarrow A^{V,T,K'}(\Phi, B(\Phi', \psi'))$$

where $\Phi' * \Phi$ denotes the concatenation of Φ' and Φ. Since K is irrelevant to the description of the knowledge of coalition A on S, it does not appear on the left of $\Rightarrow$ above. Appendix A of [6] covers the extended syntax (6).

4 Some generalisations

Here we outline some more general forms of the model of access control described in the previous sections and how our results about this model extend to these forms.

4.1 Concurrent access

Now let coalitions A and B be running programs α and β, respectively. Let the individual steps of α and β be interleaved. Let B have priority over A in the following sense: B can choose to execute as many steps of β as it wishes, then allow the next step of α to be made and regain control. Let β pass control to α for one step by executing the special command `sleep`. Intuitively, this means that B can monitor the behaviour of A to the extent it can read the variables A updates and take advantage of whatever access rights A grants B as a side effect of pursuing its own goals. Let us denote this form of parallel composition of α and β by $\alpha\|\beta$. We define $[\![\alpha\|\beta]\!]$ for α and β of the form $(\gamma;\mathtt{skip})$ for the sake of technical convenience:

$[\![\alpha\|\mathtt{skip}]\!] = [\![\alpha]\!]$;
$[\![\alpha\|(p := \varphi;\beta)]\!] = [\varphi/p][\![(\alpha\|\beta)]\!]$;
$[\![\alpha\|(\mathtt{if}\ \varphi\ \mathtt{then}\ \beta_1\ \mathtt{else}\ \beta_2;\beta_3)]\!] =$
$\quad [(\varphi \wedge [\![\alpha\|(\beta_1;\beta_3)]\!](p)) \vee (\neg\varphi \wedge [\![\alpha\|(\beta_2;\beta_3)]\!](p))/p : p \in P]$;
$[\![\alpha\|((\beta_1;\beta_2);\beta_3)]\!] = [\![\alpha\|(\beta_1;(\beta_2;\beta_3))]\!]$;
$[\![\mathtt{skip}\|(\mathtt{sleep};\beta)]\!] = [\![\beta]\!]$;
$[\![(p := \varphi;\alpha)\|(\mathtt{sleep};\beta)]\!] = [\varphi/p][\![(\alpha\|\beta)]\!]$;
$[\![(\mathtt{if}\ \varphi\ \mathtt{then}\ \alpha_1\ \mathtt{else}\ \alpha_2;\alpha_3)\|(\mathtt{sleep};\beta)]\!] =$
$\quad \left[\begin{array}{l}(\varphi \wedge [\![(\alpha_1;\alpha_3)\|(\mathtt{sleep};\beta)]\!](p)) \vee \\ (\neg\varphi \wedge [\![(\alpha_2;\alpha_3)\|(\mathtt{sleep};\beta)]\!](p))\end{array} /p : p \in P\right]$;
$[\![((\alpha_1;\alpha_2);\alpha_3)\|(\mathtt{sleep};\beta)]\!] = [\![(\alpha_1;(\alpha_2;\alpha_3))\|(\mathtt{sleep};\beta)]\!]$.

The clause $[\![\mathtt{skip}\|(\mathtt{sleep};\beta)]\!] = [\![\beta]\!]$ states that `sleep` has no effect when the program run by A has nothing left to do. To express this about subsequent possible occurrences of `sleep` in β, we extend $[\![.]\!]$ by putting $[\![\mathtt{sleep}]\!] = [\![\mathtt{skip}]\!]$. The executability $[\![A,B,\alpha\|\beta]\!]$ of $\alpha\|\beta$ by A and B can be defined like in the case of programs run by individual coalitions. We skip the definition here.

Now let $p_B \in P$ satisfy $\mathtt{w}(p_B,B) = \top$ and $\mathtt{w}(p_B,A) = \bot$ and assume that B is trying to take whatever opportunities appear while A is executing α, in order to obtain a copy of the truth value of ψ in p_B by executing β. It is natural to assume that B takes the advantage of doing as many things as it wishes between every two updates A does. That is why the actions of A and B in the course of their executing α and β respectively are interleaved as in $\alpha\|\beta$. We denote the set of all programs that can possibly have occurrences of `sleep` by $\mathbf{P}_{\mathtt{sleep}}$.

Using $[\![.\|.]\!]$ and $[\![.,.,.\|.]\!]$, we can describe, e.g. what it means for coalition B to be able to read some property φ of the state of the system by means of running program β in parallel with program α being run by coalition A:

$$\mathtt{R}_B(\varphi,A,\alpha) \rightleftharpoons (\exists\beta \in \mathbf{P}_{\mathtt{sleep}})\forall([\![A,B,\alpha\|\beta]\!] \wedge ([\![\alpha\|\beta]\!](p_0) \Leftrightarrow \varphi)) \qquad (7)$$

A fixed α implies an upper bound of the number of `sleep` statements that β may need to execute in a sequence in order to let α complete its execution. This entails that, much like in the case of a single coalition accessing the system, there are finitely

many β modulo equivalence with respect to their effect on the system, including their interaction with the fixed interleaved α. This means that the quantifier prefix $(\exists\beta \in \mathbf{P}_{\mathtt{sleep}})$ in (7) can be eliminated and, therefore, $\mathtt{R}_B(\varphi, A, \alpha)$ can be calculated. The more efficient approach described in [6] can be applied to this setting too.

4.2 Access control with arbitrary atomic actions

So far our model allows only simple assignments to boolean variables as the atomic actions. This brings the level of abstraction down and makes some natural things difficult to program. For example, consider the system $S = \langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ where $P = \{p_1, p_2\}$ and $\mathtt{w}(p_1, \Sigma) = \mathtt{w}(p_2, \Sigma) = p_1 \wedge p_2$. Then Σ can overwrite each of p_1 and p_2 at state $\{p_1, p_2\}$, but can never change the values of both variables, because once one of the values becomes 0, the writing permission is lost. Hence, there is no way to permit Σ the transition from state $\{p_1, p_2\}$ to state $\emptyset$ without also allowing Σ to change some of the states $\{p_1\}$ and $\{p_2\}$, or even to leave S in one of these states and never proceed towards $\emptyset$. This means that coalitions cannot be forced to maintain integrity constraints, like, e.g., $p_1(s) = p_2(s)$ for all $s \subseteq 2^P$, keep logs, or be saved from painting themselves into a corner. This restriction can be removed by introducing high-level atomic actions instead of the single variable assignments. In this section we argue that the technique developed for assignments as the atomic transformations on system state generalise to arbitrary atomic actions.

Let our programming language have the atomic statements $\mathtt{a}_1, \ldots, \mathtt{a}_k$, each denoting some transformation on the state. Let $[\![\mathtt{a}_i]\!]$ and $[\![A, \mathtt{a}_i]\!]$ denote the substitution which represents the transformation performed by $\mathtt{a}_i$ in the sense of Proposition 5, and the rights required for A to execute $\mathtt{a}_i$ in the sense of Proposition 6, respectively. $[\![\mathtt{a}_i]\!]$ and $[\![A, \mathtt{a}_i]\!]$ were defined in terms of $\mathtt{r}$ and $\mathtt{w}$ for the case of $\mathtt{a}_i$ being assignments in Section 2. Now we assume that these are *given* substitutions and formulas for $\mathtt{a}_1, \ldots, \mathtt{a}_k$. Let us replace $\mathtt{w}$ by the mappings $[\![., \mathtt{a}_i]\!] : 2^\Sigma \to \mathbf{L}(P)$, $i = 1, \ldots, k$, in systems which control access by atomic actions $\mathtt{a}_1, \ldots, \mathtt{a}_k$. Let us retain $\mathtt{r}$, which determines reading permissions. We obtain access control systems of the form

$$\langle P, \Sigma, \mathtt{r}, \lambda A.[\![A, \mathtt{a}_1]\!], \ldots, \lambda A.[\![A, \mathtt{a}_k]\!]\rangle.$$

Since for every subsitution $[\![\mathtt{a}_i]\!]$ there is an α_i of the form (2) such that $[\![\mathtt{a}_i]\!] = [\![\alpha_i]\!]$, we can reproduce the results from Sections 2-3 and the appendix of [6] about such systems. This possibility shows that, despite its restrictions, the language (2) and the techniques for it from Sections 2.2-3 have a fundamental role in the assembly of the respective machinery for the analysis of finite access control described in terms of high-level actions.

4.3 General knowledge states

So far we have allowed only pairs of the form V, T to represent the knowledge states of coalitions. A knowledge state can be viewed as a nonempty set of system states. Pairs

V, T can represent only some such sets. In general, any set of system states described by a satisfiable formula from $\mathbf{L}(P)$ can be viewed as a knowledge state. This leads to a natural generalisation of $A^{V,T,K}(\ldots)$ to the form $A^{\chi,K'}(\ldots)$ where K' is a sequence of formulas $\chi_1, \ldots, \chi_k$ from $\mathbf{L}(P)$ such that $\vdash \chi_{i+1} \Rightarrow \chi_i$ and $\nvdash \chi_i \Rightarrow \chi_{i+1}$, $i < k$, and χ_k is χ.

General knowledge states can be used to deal with the assumption that a failed access attempt ony causes the attempting coalition A to learn that the (generally arbitrary) formula $\mathtt{x}(p, A)$ does not hold, where $\mathtt{x} \in \{\mathtt{r}, \mathtt{w}\}$ denotes the attempted action and p is the accessed variable.

Conclusions

We conclude by listing some problems whose solutions can be derived from the techniques developed in this paper.

Model checking (Synthesis of attacks). Given a concrete access control system of the form $\langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ a recursive equation for $A(\Phi, \psi)$ from the appendix of [6] provides an algorithm to calculate the ability of a coalition A to achieve a general goal combining reading and writing variables, and, if there is such ability, to synthesise a program for A to achieve the goal. Hence it can be checked whether the system permits various forms of legitimate access, leak of data or attacks which can be written as goals of the form (Φ, ψ). In Subsection 4.1 we show that the same problem is decidable in the situation of the potential intruders acting in parallel with legitimate users and taking whatever temporary opportunities the actions of legitimate users present.

Synthesis of access control systems. Given a set of propositional variables P, a set of agents Σ and an access control policy formulated as a logical theory about $A(.,.)$ for $A \subseteq \Sigma$ on systems which have their state described in terms of the variables from P, it can be decided whether an access control system of the form $\langle P, \Sigma, \mathtt{r}, \mathtt{w}\rangle$ which implements this policy exists and, if so, definitions for its remaining components $\mathtt{r}$ and $\mathtt{w}$ can be proposed. This can be done by developing the equation from [6] into full propositional definitions of the instances of $A(.,.)$ involved in the formulation of the policy and establishing the satisfiability of the policy with respect to the applications of $\mathtt{r}$ and $\mathtt{w}$ at the respective states of the system treated as propositional variables. If the policy turns out to define a satisfiable restriction on $\mathtt{r}$ and $\mathtt{w}$, any particular pair of mappings $\mathtt{r}$ and $\mathtt{w}$ which satisfies this restriction can be chosen to complete the access control system in a way which implements the given policy.

In Section 4 we argued that the results from Sections 2-3 can be reproduced for systems of a general form where access is based on an arbitrary set of high-level actions. A representation of the respective access operator $A(.,.)$ like that in the appendix of [6] for the basic case can be assembled from the components used in this basic case. We proposed a way to reason about goals which involve enabling some further goals to be achieved. We also showed how to generalise the form of knowledge states of coali-

tions used in Sections 2-3 and thus allow to describe that coalitions know arbitrary constraints on the states of systems and that coalitions can learn from failures.

The algorithms which follow from the appendix of [6] are not optimal. Results on the complexity of the problems on the class of all access control systems of the considered form might be practically unrepresentative, because instances of extreme complexity usually have little in common with typical real cases. That is why it would be interesting to describe subclasses which exhibit the kinds of regularity typical for real access control systems first.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] Jean Bacon, Ken Moody, and Walt Yao. Access control and trust in the use of widely distributed services. *Lecture Notes in Computer Science*, 2218:295+, 2001. Also: *Software Practice and Experience* 33, 2003.

[3] O. Bandmann, M. Dam, and B. Firozabadi. Constrained delegations. In *Proc. IEEE Symposium on Security and Privacy*, pages 131–142, 2002.

[4] E. S. Barka. *Framework for Role-Based Delagation Models*. PhD thesis, George Mason University, 2002.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[6] M. D. Ryan D. P. Guelev and P. Y. Schobbens. Model-checking access control policies. `http://www.cs.bham.ac.uk/~dpg/fullaclpaper.ps/`, April 2004.

[7] Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Access control: principles and solutions. *Software Practice and Experience*, 33:397–421, 2003.

[8] D. P. Guelev. Prolog code supporting "Model-checking access control policies". `http://www.cs.bham.ac.uk/~dpg/mcacp/`, November 2003.

[9] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. In *Proceedings of the fifth symposium on Operating systems principles*, pages 14–24. ACM Press, 1975.

[10] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[11] H. Riis Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992.

[12] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

A Model-checking $A^{V,T,\{T\}}(.,.)$ in the propositional μ-calculus

Alternatively, $A^{V,T,\{T\}}(\langle\varphi_1,\ldots,\varphi_l\rangle,\psi)$ can be calculated by model-checking a formula in the propositional μ-calculus (see e.g. [5]). Assume there are no subgoals of the form $B(\Phi',\psi')$ in ψ for the sake of simplicity. Consider a system, whose state space is the set of quadruples V_0,T_0,V,T such that $T_0\subseteq V_0\subseteq V\subseteq P$ and $T\subseteq V$, where P is the vocabulary of a fixed access control system as above. A quadruple V_0,T_0,V,T represents a state of knowledge of A which consists of the fact $V_0\cap s_0=T_0$ about the initial state s_0 of S and the fact $V\cap s=T$ about the current state s of S. The meaning of V,T is as in Section 3. Hence the quadruple V_0,T_0,V,T represents A's knowledge of both the initial and the current state. (The addition V_0,T_0 is not needed in these axioms, because they prescribe to simplify φ in subgoals of the form $\overline{\varphi}$ and in formulas to read from Φ immediately each time the value of a variable becomes known.)

Consider the μ-calculus language with the modalities $\langle\mathtt{sample}\ p\rangle$, $\langle p{:=}\bot\rangle$ and $\langle p{:=}\top\rangle$ for each $p\in P$. Let the corresponding accessibility relations $R_{\mathtt{sample}\ p}$, $R_{p:=\bot}$ and $R_{p:=\top}$ be defined by the clauses

$$R_{\mathtt{sample}\ p}(V_0,T_0,V,T;V_0',T_0',V',T')\leftrightarrow$$
$$\left(\begin{array}{c}\mathtt{r}(p,A)\wedge p\notin V\wedge V_0'=V_0\cup\{p\}\wedge V'=V\cup\{p\}\wedge\\ \left(\begin{array}{c}T_0'=T_0\setminus\{p\}\wedge T'=T\setminus\{p\}\vee\\ T_0'=T_0\cup\{p\}\wedge T'=T\cup\{p\}\end{array}\right)\end{array}\right)$$

$$R_{p:=\bot}(V_0,T_0,V,T;V_0',T_0',V',T')\leftrightarrow$$
$$\mathtt{w}(p,A)\wedge V_0'=V_0\wedge T_0'=T_0\wedge V'=V\cup\{p\}\wedge T'=T\setminus\{p\}$$

$$R_{p:=\top}(V_0,T_0,V,T;V_0',T_0',V',T')\leftrightarrow$$
$$\mathtt{w}(p,A)\wedge V_0'=V_0\wedge T_0'=T_0\wedge V'=V\cup\{p\}\wedge T'=T\cup\{p\}$$

Each of these relations represents an action on behalf of A in which A increases its knowledge and/or changes the current state. The knowledge of A is sufficient to establish that its goal is already achieved iff the formula

$$\forall[\sigma_{V_0,T_0}\varphi/\overline{\varphi},\sigma_{V,T}\varphi/\overline{\varphi}':\varphi\in\mathbf{L}(P)]\psi\wedge\bigwedge_{i=1}^{l}(\forall\sigma_{V_0,T_0}\varphi_i\vee\forall\sigma_{V_0,T_0}\neg\varphi_i)$$

is valid. Let the set of states from which A can reach a satisfactory state be X. Then the implications

$$\langle p{:=}\bot\rangle X\Rightarrow X,\quad\langle p{:=}\top\rangle X\Rightarrow X,\quad[\mathtt{sample}\ p]X\Rightarrow X,\quad p\in P\qquad(8)$$

The $[\cdot]$ in the last formula means that A should be prepared for any outcome of the sampling. The least solution of the system of inclusions (8) is the set of the knowledge

states in which A can make a plan to reach a satisfactory state without fail. Hence $A^{V,T,\{T\}}(\langle\varphi_1,\ldots,\varphi_l\rangle,\psi)$ is equivalent to the satisfaction of

$$\mu X.\left(\begin{array}{c}\left(\forall[\sigma_{V_0,T_0}\varphi/\overline{\varphi},\sigma_{V,T}\varphi/\overline{\varphi}':\varphi\in\mathbf{L}(P)]\psi\wedge\bigwedge_{i=1}^{l}(\forall\sigma_{V_0,T_0}\varphi_i\vee\forall\sigma_{V_0,T_0}\neg\varphi_i)\right)\vee\\ \bigvee_{p\in P}([\texttt{sample}\ p]X\vee\langle p{:=}\bot\rangle X\vee\langle p{:=}\top\rangle X)\end{array}\right)$$

at state $\emptyset,\emptyset,V,T$.

Inheritance hierarchies in the Or-BAC model and application in a network environment

Frédéric Cuppens[1] Nora Cuppens-Boulahia[1] Sylvain Gombault[1]
Alexandre Miège[1,2]

Abstract

Role hierarchy was first introduced in the Role Based Access Control (RBAC) model. Inheritance of permissions is associated with this hierarchy. This is useful to design security policies in a modular way. In this paper, we extend this approach in the context of the Organization Based Access Control (Or-BAC) model. We first define hierarchies of roles, views and activities and formally model inheritance mechanism associated with each hierarchy. We then define hierarchy of organizations. We show that this provides efficient means to derive policies of security components from corporate security policies specification. We illustrate our approach in the context of network security policy, in particular to configure firewalls.

1 Introduction

The inheritance mechanism was suggested in object oriented programming as an efficient way to design an application in a modular way. A similar mechanism is used in RBAC [14] when a hierarchy of roles is defined and associated with inheritance of permissions. The role hierarchy is a useful mean to structuring the security policy specification.

However, the concept of role hierarchy is not free of ambiguity [11, 12, 5]. Some of these ambiguities are directly related to the concept of role itself. If we consider the examples suggested in [14], there are actually several different interpretations of roles. Basically, a role allows a subject who is assigned to this role to perform some particular activities. This is the case of roles such as *physician*, *nurse* or *medical secretary*. However, in some examples, a role is related to a given *organization* [13]. For instance, we may consider roles such as *nurse in a cardiological department* or *nurse in a reanimation team*. This is interesting because permissions assigned to a given role may change from one organization to another. For example, a nurse may have different permissions if she performs her activities in a cardiological department or in a reanimation team. A role may also correspond to the activity of leading a given

organization. Examples of such roles may be *director of an hospital* or *head of a cardiological department*.

As we shall see in the following, these different interpretations of the role concept do not behave similarly with respect to permission inheritance. This is why it is important to have a model that provides means to make explicit such differences.

We shall analyze this problem in the context of the Or-BAC model [10]. This model is centered on the concept of *organization*. In Or-BAC, an organization corresponds to any entity that is in charge of managing a set of security rules (permissions or prohibitions). For instance, a given hospital is an organization. A concrete security component, such as a firewall, may be also viewed as an organization since it manages a set of security rules.

Role definition in Or-BAC is always related to a given organization. This is useful to avoid some ambiguities when defining role hierarchies and associating them with permission inheritance.

The Or-BAC model considers two other concepts, namely *activity* and *view*. A security policy assigned to a given organization is defined as permissions (or prohibitions) for roles to perform activities on views. In the following, we shall suggest defining hierarchies of activities and views and model inheritance mechanism associated with these hierarchies.

Finally, in Or-BAC, we can also define hierarchy of organizations. This possibility provides a very efficient mean to structuring the security policy specification, starting with high level organization such as an hospital and finishing with concrete security components such as a firewall.

In this paper, we aim to analyze and formally model inheritance of permissions and prohibitions through these different hierarchies. The remainder of this paper is organized as follows. Section 2 recalls main concepts of Or-BAC. Section 3 presents the various hierarchies respectively associated with roles, activities and views. Section 4 studies organization hierarchies. In section 5, we summarize how to specify a security policy in Or-BAC when hierarchies are used. Section 6 shows how our approach applies to network security policy specification. Finally, section 7 concludes and suggests several issues to this work.

2 Or-BAC

2.1 Basic concepts of Or-BAC

Or-BAC [10] is an access control model based on the organization concept. In Or-BAC, different organizations can specify their own access control policy using eight basic sets of entities: Org(anization), $Role$, $Activity$, $View$, $Subject$, $Action$, $Object$ and $Context$. Basic predicates used in Or-BAC to model relationships between these eight entities are summarized in figure 1.

As mentioned in the introduction, an organization is any entity that manages a set of security rules. A subject is an active entity that may be assigned to a role. We

shall assume that $Org \subseteq Subject$ so that a role may be assigned to an organization. For instance, roles "casualty department" or "rescue team" may be assigned to some organizations.

By means of the entity $Role$, we are able to structure the subjects and to update easily security policies when new subjects are added to the system. Since we have also to structure the objects and to add new objects to the system, a similar entity regarding objects is needed: the entity $View$. Roughly speaking, as in relational databases, a view corresponds to a set of objects that satisfy a common property. Another entity is used to abstract actions: the entity $Activity$. Seeing that roles associate subjects that fulfil the same functions and views correspond to sets of objects that satisfy a common property, activities will join actions that share the same principles.

Subjects, objects and actions may have attributes. This is modelled by a set of binary predicates having the form $att(ent, val)$ where ent is a subject, an object or an action and val is the value of attribute att. For instance, if med_27 is a medical record, then $name(med\_27, John)$ means that med_27 is John's medical record.

We assume that $Subject \subseteq Object$ so that we can define views of subjects that we call *groups*. In Or-BAC, there is a clear difference between a role and a group. Permissions are assigned to roles whereas a group is simply a set of subjects that have some common properties. However, it is sometimes useful to assign the same role to every subject belonging to a given group. For this purpose, we can use the predicate $G\_Empower(org, group, role)$ and specify the following rule:

- GE: $\forall org, \forall group, \forall role, \forall subject,$
  $Use(org, subject, group) \wedge G\_Empower(org, group, role)$
  $\rightarrow Empower(org, subject, role)$

Since the Or-BAC model allows the administrator to specify that some permission or prohibition only applies in specific *contexts*, we also introduce the entity $Context$. Contexts are defined by logical rules whose conclusion is the predicate $Define$ (see figure 1 that gives the example of the *working_hours* context). We say that a context c in organization org is defined by condition $cond$ when there is a rule having the form: $Define(org, s, \alpha, o, c) \leftarrow cond$. Specifying contexts in Or-BAC is further analyzed in [7].

Predicate name	Domain	Description
$Relevant\_role$	$Org \times Role$	If org is an organization and r a role, then *Relevant_role(org,r)* means that playing role r is defined in organization org. Ex: $Relevant\_role(H, physician)$
Relevant_activity	$Org \times Activity$	If org is an organization and a is an activity, then *Relevant_activity(org,a)* means that performing activity a is defined in organization org. Ex: $Relevant\_activity(H, consult)$
$Relevant\_view$	$Org \times View$	If org is an organization and v is a view, then *Relevant_view(org,v)* means that using view v is defined in organization org. Ex: $Relevant\_view(H, medical\_record)$
$Empower$	$Org \times Subject \times Role$	If org is an organization, s a subject and r a role, then $Empower(org, s, r)$ means that org empowers subject s in role r. Ex: $Empower(H, John, physician)$
$Consider$	$Org \times Action \times Activity$	If org is an organization, α is an action and a is an activity, then $Consider(org, \alpha, a)$ means that org considers that action α falls within the activity a. Ex: $Consider(H, "SELECT", consult)$
Use	$Org \times Object \times View$	If org is an organization, o is an object and v is a view, then $Use(org, o, v)$ means that org uses object o in view v. Ex: $Use(H, med\_27, medical\_record)$
$Define$	$Org \times Subject \times Action \times Object \times Context$	If org is an organization, s a subject, α an action, o an object and c a context, then $Define(org, s, \alpha, o, c)$ means that within organization org, context c holds between subject s, action α and object o. Ex: $\forall s, \forall \alpha, \forall o, Define(H, s, \alpha, o, working\_hours) \leftarrow (08:00 \leq time(GLOBAL\_CLOCK) \wedge time(GLOBAL\_CLOCK) \leq 19:00)$

Figure 1. Basic predicates of Or-BAC

2.2 Permission and prohibition

Permissions and prohibitions in Or-BAC are defined with predicates defined in figure 2. The access control policy is specified at two different levels: an abstract level that specifies permissions and prohibitions between role, activity and view, and a concrete level where permissions and prohibitions between subject, action and object are derived. These two levels are related as follows. In a given organization org, a subject s is permitted to perform an action α on an object o if (1) s is empowered to play a given role r in org and (2) α implements a given activity a in org and (3) o is used in a given view v by org. If these three conditions are satisfied and if (4) the organization

Predicate name	Domain	Description
$Permission$	$Org \times Role \times Activity \times View \times Context$	If org is an organization, r a role, a an activity , v a view and c a context, then $Permission(org, r, a, v, c)$ means that organization org grants to role r the permission to perform activity a on view v in context c. Ex: $Permission(H, physician,$ $consult, medical\_record, working\_hours)$
$Prohibition$	$Org \times Role \times Activity \times View \times Context$	If org is an organization, r a role, a an activity , v a view and c a context, then $Prohibition(org, r, a, v, c)$ means that organization org prohibits role r from performing activity a on view v in context c. Ex: $Prohibition(H, nurse,$ $consult, medical\_record, night)$
$Is\_permitted$	$Subject \times Action \times Object$	If s is a subject, α an action, o an object, then $Is\_permitted(s, \alpha, o)$ means that s is concretely permitted to perform action α on object o. Ex: $Is\_permitted(John, "SELECT", med\_27)$
$Is\_prohibited$	$Subject \times Action \times Object$	If s is a subject, α an action, o an object, then $Is\_prohibited(s, \alpha, o)$ means that s is concretely prohibited to perform action α on object o. Ex: $Is\_prohibited(Mary, "DELETE", med\_27)$

Figure 2. Permissions and prohibitions specification in Or-BAC

org grants to role r the permission to perform the activity a on the view v, then the request by the subject s to perform the action α on the object o is accepted. Deriving concrete permissions from abstract permissions is modelled by the following rule:

- RG_1: $\forall org, \forall r, \forall a, \forall v, \forall s, \forall \alpha, \forall o,$
  $Permission(org, r, a, v, c) \wedge$
  $Empower(org, s, r) \wedge$
  $Consider(org, \alpha, a) \wedge$
  $Use(org, o, v) \wedge$
  $Define(org, s, o, \alpha, c)$
  $\rightarrow Is\_permitted(s, \alpha, o)$

Another similar rule (called RG_2) is used to derive concrete *prohibitions* from abstract prohibitions.

2.3 Constraints

Constraints that apply to an access control policy was first suggested in the RBAC model (more precisely, in the $RBAC_2$ sub-model [8]) and further analyzed in [1]. To

specify constraints in the Or-BAC model, we introduce a predicate $error()$. A constraint is then modelled as a rule whose conclusion is $error()$ (as suggested in [3, 9]).

For instance, we may specify that, in hospital H, a subject cannot be empowered in both roles $anesthetist$ and $surgeon$:

- C_1: $\forall s$,
  $Empower(H, s, anesthetist) \wedge Empower(H, s, surgeon)$
  $\rightarrow error()$

In the following, we shall consider the following constraint that apply to any organization:

- C_2: $\forall org, \forall s, \forall r$,
  $Empower(org, s, r) \wedge \neg Relevant\_role(org, r)$
  $\rightarrow error()$

Rule C_2 says that an organization org should not empower a subject s in role r if role r is not relevant in organization org.

There are other rules similar to C_2 but for activities (called C_3), views (called C_4), permissions (called C_5) and prohibitions (called C_6).

3 Hierarchy within an organization

In Or-BAC, it is possible to consider hierarchies of roles (as suggested in [8]) but also of views and activities. Every hierarchy respectively defines a partial order relation over the set of roles, views and activities. We present general inheritance rules of permissions and prohibitions associated with these different hierarchies.

3.1 Role hierarchy

Let us first address the case of inheritance between roles. In every organization, it is possible to associate a set of roles with a hierarchy. For this purpose, we introduce the predicate $sub\_role(org, r_1, r_2)$: in organization org, role r_1 is a sub-role of r_2. Notice that the role hierarchy depends on the organization. This means that the hierarchy may vary from one organization to another. Let us now model inheritance principles associated with this hierarchy.

Permission inheritance through the role hierarchy is modelled by the following rule:

- RH_1: $\forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c$,
  $sub\_role(org, r_1, r_2) \wedge Permission(org, r_2, a, v, c)$
  $\rightarrow Permission(org, r_1, a, v, c)$

This rule says that if role r_1 is a sub-role of role r_2 in organization org, then every permission assigned to role r_2 in organization org is also assigned to role r_1.

Regarding prohibition inheritance, things are more complex. It is necessary to recognize that the relationships between roles in the hierarchy may be semantically different. We actually identify two different relationships:

- Relationship of specialization/generalization. For instance, role $surgeon$ is a specialization of role *physician*. To model this first relationship we shall use the following predicate:

  $specialized\_role(org, r_1, r_2)$: in organization org, role r_1 is a specialized role of role r_2.

- Relationship of organizational hierarchy. For instance, role *department director* may be defined as hierarchically higher than role *team head*. This second relationship is modelled by the following predicate:

  $senior\_role(org, r_1, r_2)$: in organization org, role r_1 is a senior role of role r_2.

We consider that relationship $specialized\_role$ is included in sub_role:

- RH$_2$: $\forall org, \forall r_1, \forall r_2,$
  $specialized\_role(org, r_1, r_2) \rightarrow sub\_role(org, r_1, r_2)$

The consequence is that rule RH$_1$ applies to the specialization role hierarchy and thus permissions are inherited through this hierarchy. We also consider that prohibitions are inherited through the specialization role hierarchy:

- RH$_3$: $\forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c,$
  $specialized\_role(org, r_1, r_2) \wedge Prohibition(org, r_2, a, v, c)$
  $\rightarrow Prohibition(org, r_1, a, v, c)$

For instance, every prohibition of the role *physician* is inherited by the role *surgeon*. This is compatible with the intuition that a $surgeon$ is a special case of $physician$.

By contrast, the relationship $senior\_role$ is generally not included in sub_role. This means that we may have:

- $\exists org, \exists r_1, \exists r_2,$
  $senior\_role(org, r_1, r_2) \wedge \neg sub\_role(org, r_1, r_2)$

For instance, we can consider that role *hospital director* is hierarchically higher than *physician*. However, in some hospitals, role *hospital director* is a purely administrative role that is not assigned to a physician. In this case, there is no reason to conclude that *hospital director* is a sub role of physician.

Now let us assume that role r_1 is a senior role of r_2 and that r_1 is also a sub-role of r_2. In this case, the idea is to consider that r_1 is "more powerful" than r_2. This is compatible with rule RH$_1$ above that specifies that r_1 inherits the permissions assigned to r_2. However, if we assume that r_1 inherits the prohibitions assigned to r_2, this will not make r_1 more powerful than r_2. This is why it would be better to consider that, in case of organizational hierarchy, prohibitions are inherited "upward". This is modelled by the following rule:

- RH_4: $\forall org, \forall r_1, \forall r_2, \forall a, \forall v, \forall c,$
  $sub\_role(org, r_1, r_2) \wedge senior\_role(org, r_1, r_2) \wedge Prohibition(org, r_1, a, v, c) \wedge$
  $\rightarrow Prohibition(org, r_2, a, v, c)$

For instance, if we consider that *department director* is a senior role and also a sub role of *head team*, then *department director* inherits the permissions assigned to *team head* (from rule RH_1) and *team head* inherits the prohibitions assigned to *department director* (from rule RH_4).

To summarize, we have defined three different role hierarchies: sub_role, $specialized\_role$ (included in sub_role) and $senior\_role$ (generally not included in sub_role). If we are only interesting in the problem of inheritance of permissions and prohibitions, we can actually consider only two hierarchies: sub_role and $specialized\_role$. Regarding the $specialized\_role$ hierarchy, rules RH_1, RH_2 and RH_3 apply. Regarding the remainder of the sub_role hierarchy, rules RH_1 and RH_4 apply.

Notice also that every inheritance rule presented in this section may have exceptions. For instance, one may specify that, in hospital H, physicians are prohibited to consult the medical records of people who are not their patients. Thus, applying rule RH_3, a surgeon will inherit this prohibition. However, one can explicitly specify (as an exception) that, in hospital H, a surgeon is permitted to consult every medical records, even if they do not concern the surgeon's patient. How to manage exception is further discussed in section 5.2.

3.2 Activity hierarchy

We now suggest defining inheritance between activities. For this purpose, in every organization, the set of activities is associated with a hierarchy. This is modelled by the predicate $sub\_activity(org, a_1, a_2)$: in organization org, activity a_1 is a sub-activity of a_2. The interpretation of this hierarchy is that, in organization org, activity a_1 is a specialization of activity a_2. For instance, in hospital H, the activity of *managing* (for example medical records) may be specialized into the activities of *creating*, *consulting* and *updating*. Thus we have: *sub_activity(H,creating,managing)* and similarly for *consulting* and *updating*.

This hierarchy is associated with permission inheritance. This is modelled by the following rule:

- AH_1: $\forall org, \forall r, \forall a_1, \forall a_2, \forall v, \forall c,$
  $Permission(org, r, a_2, v, c) \wedge sub\_activity(org, a_1, a_2)$
  $\rightarrow Permission(org, r, a_1, v, c)$

For instance, let us assume that, in hospital H, physicians are permitted to manage medical records of their patients. Applying rule AH_1 we can derive that physicians are also permitted to create, consult and update medical records of their patients.

We consider that a similar rule (called AH_2) applies to inheritance of prohibitions. For instance, let us assume that, in hospital H, nurses are prohibited to manage medical

records. Applying rule AH_2 we can derive that nurses are also prohibited to create, consult and update medical records.

3.3 View hierarchy

Using a similar approach, the set of views is associated with a hierarchy that depends on the organization. This is modelled by the predicate $sub\_view(org, v_1, v_2)$: in organization org, view v_1 is a sub-view of v_2. Our interpretation is that, in org, view v_1 is a specialization of view v_2. This is actually close to class inheritance hierarchy used in object-oriented hierarchy (Isa hierarchy).

In the context of Or-BAC, view hierarchies are associated with the permission inheritance that is modelled by the following rules:

- VH_1: $\forall org, \forall r, \forall a, \forall v_1, \forall v_2, \forall c,$
  $Permission(org, r, a, v_2, c) \wedge sub\_view(org, v_1, v_2)$
  $\rightarrow Permission(org, r, a, v_1, c)$

A similar rule (called VH_2) applies to inheritance of prohibitions. For instance, in a hospital, we may consider that the view *surgeon_record* is a sub-view of the view *medical_record*. In this case, a role who is permitted or prohibited to perform a given activity on the view *medical_record* will be permitted or prohibited to perform the same activity on the view *surgeon_record*.

We can also define the concept of *derived* view as a special case of view specialization. Hence, we can define that a view v_1 is derived from view v_2 if there is a rule having the following form:

- $\forall org, \forall obj, \forall v_1, \forall v_2,$
  $(Use(org, obj, v_2) \wedge Condition) \rightarrow Use(org, obj, v_1)$
  where $Condition$ is a logical condition used to specialize view v_2 into view v_1.

4 Organization hierarchy

Previous section showed how to define hierarchies between roles, activities and views within a given organization. In this section, we study how to define hierarchies of organizations. For this purpose, we introduce the predicate $sub\_organization(org_1, org_2)$: organization org_1 is a sub-organization of organization org_2. We assume that this predicate defines a partial order relation on the set of organizations. For instance, if H is an hospital and $dept8$ is the casualty department of this hospital, then we have: $sub\_organization(dept8, H)$.

We may actually require that every sub-organization org_1 of a given organization org_2 is assigned to a role. This requirement is modelled by the following constraint:

- C_7: $\forall org_1, \forall org_2, \forall r,$
  $sub\_organization(org_1, org_2) \wedge \neg Empower(org_2, org_1, r)$
  $\rightarrow error()$

For instance, in the above example, constraint C_7 is satisfied if we have: $Empower(H, dept8, casualty\_dept)$.

Notice that some roles may be defined in a given organization but not in some of its sub-organizations. For instance if $dept7$ is the management department of the hospital H, then the role nurse may be not defined in $dept7$ (we have $\neg Relevant\_role(dept7, nurse)$) whereas it is defined in H (we have $Relevant\_role(H, nurse)$).

Conversely, if org_1 is a sub-organization of org_2, then some roles may be defined in org_1 whereas they are not defined in org_2.

Similar comments apply to views and activities: if org_1 is a sub-organization of org_2, then the views (resp. activities) defined in org_1 may be different from the views (resp. activities) defined in org_2.

4.1 Hierarchy inheritance

Let us assume that org_1 is a sub-organization of org_2. For those roles of org_2 that are relevant in org_1, we consider that the role hierarchy defined in org_2 also applies in org_1. This is modelled by the following rule:

- HH_1: $\forall org_1, \forall org_2, \forall r_1, \forall r_2,$
  $sub\_organization(org_2, org_1) \wedge sub\_role(org_1, r_1, r_2) \wedge$
  $relevant\_role(org_2, r_1) \wedge relevant\_role(org_2, r_2)$
  $\rightarrow sub\_role(org_2, r_1, r_2)$

Similar principles apply to inheritance of specialized role hierarchy and also of activity and view hierarchies through the organization hierarchy. Thus, we obtain three other rules (respectively called HH_2, HH_3 and HH_4) by replacing the sub_role predicate in rule HH_1 by the $specialized\_role$ predicate (resp. the $sub\_activity$ and sub_view predicates).

4.2 Permission and prohibition inheritance

We accept similar principles for inheritance of permissions and prohibitions through the organization hierarchy provided that the role, activity and view in the scope of the permission or prohibition are relevant in the sub-organization. This is modelled by the following rule:

- OH_1: $\forall org_1, \forall org_2, \forall r, \forall a, \forall v, \forall c,$
  $sub\_organization(org_2, org_1) \wedge Permission(org_1, r, a, v, c) \wedge$
  $relevant\_role(org_2, r) \wedge relevant\_activity(org_2, a) \wedge relevant\_view(org_2, v)$
  $\rightarrow Permission(org_2, r, a, v, c)$

A similar rule applies to inheritance of prohibition (rule called OH_2).

5 Specifying a security policy in Or-BAC

5.1 Policy theory

To summarize, a security policy that includes inheritance hierarchies is modelled as a logical theory corresponding to the following definition.
Definition 1: In the Or-BAC model, a security policy pol is modelled as a logical theory T_{pol} defined as follows:

- Sets of facts using predicates $Relevant\_role$, $Relevant\_activity$ and $Relevant\_view$
- Sets of facts using predicates $Empower$, Use, $Consider$, $Permission$ and $Prohibition$
- Rule GE and facts using predicate $G\_Empower$
- A set of rules for derived view definition (section 3.3)
- A set of facts using attribute binary predicates for describing attribute values of subjects, actions and objects
- A set of context definition rules, i.e. rules whose conclusion is the predicate $Define(org, s, \alpha, o, c)$
- Sets of facts (inheritance hierarchies) using predicates sub_role, $specialized\_role$, $sub\_activity$ and sub_view
- Rules RG_1 and RG_2 for deriving concrete permissions and prohibitions (section 2.1)
- Rules RH_1 to RH_4 (role inheritance rules), AH_1 and AH_2 (activity inheritance rules) and VH_1 and VH_2 (view inheritance rules)
- Rules HH_1 to HH_4 (hierarchy inheritance rules)
- Rules OH_1 and OH_2 (organization inheritance rules)
- A set of constraints, i.e. rules whose conclusion is the predicate $error()$.

Definition 2: The security pol violates a constraint if it is possible to derive $error()$ from T_{pol}: $T_{pol} \vdash error()$

5.2 Conflicts

Since the Or-BAC model provides means to specify both permissions and prohibitions, it is possible that some conflicts arise. This occurs when a given user is both permitted and prohibited to perform a given action on a given object. To model such a situation, we introduce a predicate called $conflict()$. The following rule specifies a situation of conflict:

- RC: $\forall s, \forall \alpha, \forall o,$
  $Is\_permitted(s, \alpha, o) \wedge Is\_prohibited(s, \alpha, o)$
  $\rightarrow conflict()$

Definition 3: There is a conflict in the security pol if it is possible to derive $conflict()$ from T_{pol}: $T_{pol} \vdash conflict()$

Notice that exceptions in the inheritance rules may lead to conflicts. For instance, in the example presented in section 3.1, a surgeon may be both prohibited to consult a given medical record of someone who is not his or her patient (prohibition inherited from role *physician*) and permitted to do so (explicit permission assigned to role *surgeon*).

In [6], we suggest managing such a conflict by assigning priority to permissions and prohibitions. In our previous example, prohibition inherited from *physician* should have lower priority than explicit permission assigned to *surgeon* and thus this surgeon should be finally permitted to consult the medical record. However, this is not the purpose of this paper to further discuss how to manage conflicts in the Or-BAC model (see [6] for a detailed presentation).

6 Application

In this section, we model a local area network, its security architecture and its connectivity to the Internet. We choose to reuse the example used in Firmato [2] so as to bring out how Or-BAC provides a natural statement of various entities and concepts used in the security architecture. Furthermore, we show that hierarchy notions of extended Or-Bac applied to the central entities, say organization, role, activity and view, avoid the use of artifices like ”open” and ”closed” groups suggested in [2].

6.1 Organization hierarchy

We want to model the access control policy of a corporate network used in an organization H. H has a two-firewall network configuration, as shown in Figure 3. As presented in [2], the external firewall guards the corporation’s Internet connection. Behind it is the DMZ, which contains the corporation’s externally visible servers. In our case these servers provide http/https (web), ftp, smtp (e-mail), and dns services. The corporation actually only uses two hosts to provide these services, one for dns and the other (called Multi_server) for all the other services. Behind the DMZ is the internal firewall which guards the corporation’s intranet. This firewall actually has three

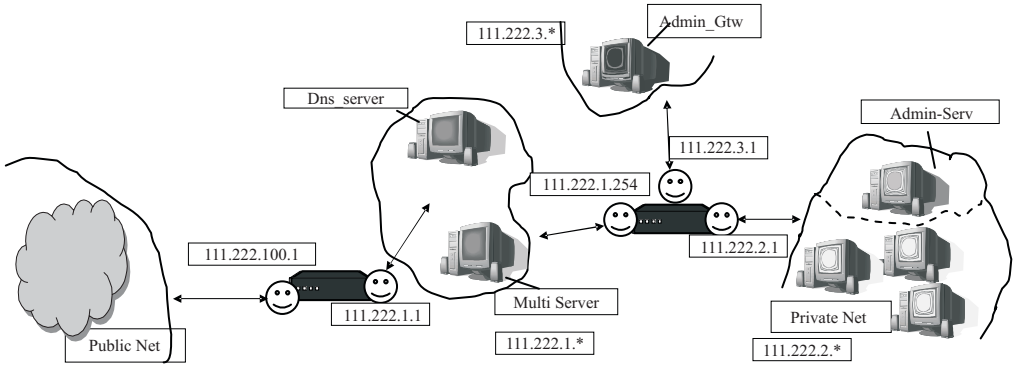


Figure 3. Application example

interfaces: one for the DMZ, one for the private network zone, and a separate interface connecting to the firewall administration host. Within the private network zone, there is one distinguished host, *Admin_serv*, which provides the administration for the servers in the DMZ.

In Or-BAC, we introduce several organizations to model this configuration. First, there is an organization H and to simplify, we shall actually identify H with its corporate network. H has two sub-organizations denoted H_fw_1 and H_fw_2 that respectively correspond to the internal and external firewalls. We may actually introduce other organizations, such as $H\_private\_net$ if one would like to specify the policy to be enforced within H private network. For instance, if H is an hospital, we might introduce roles such as $physician$, $nurse$, etc., to model this part of the policy. However, for the sake of simplicity, we shall not further refine this part. Notice that we could also use an organization called $internet$ if we had to specify an explicit policy to be enforced by the Internet.

6.2 Subject

In this example, subjects correspond to hosts identified by their IP address. So if h is an host, then predicate $address(h, a)$ means that the IP address of h is a. Roles are assigned to hosts as suggested in section 6.3 below. For this purpose, predicate $Empower$ enables us to assign a role to a given host. However, it would easier to cluster hosts into groups (also called $zone$ in Firmato) and use $G\_Empower$ to assign the same role to every host belonging to the same group. For instance, we can define the group $Private\_net$ as follows:

- $\forall h, Use(H, h, Private\_net) \leftarrow (Use(H, h, Host) \wedge address(h, a)$
  $\wedge a \in 111.222.2.* \wedge \neg Use(H, h, Firewall\_interface))$

6.3 Role

Hosts may be assigned to roles presented in figure 4. All these roles are relevant to H. Figure 4 specifies those roles that are respectively relevant to organizations H_fw_1

Role name	Hosts assigned to role	sub_role	Relevant to H_fw_1	Relevant to H_fw_2
$Public\_host$	Hosts in view *Public_Net*	-	X	
$Private\_host$	Hosts in view *Private_Net*	-		X
$Firewall$	Firewall interfaces	*External_firewall* *Internal_firewall*		X
External_firewall	External firewall interfaces	-	X	X
Internal_firewall	Internal firewall interfaces	-		X
DNS_server	DNS server	-	X	X
Ftp_server	Ftp server	$Multi\_server$	X	X
$Mail\_server$	Mail server	$Multi\_server$	X	X
Web_server	Web server	$Multi\_server$	X	X
$Multi\_server$	Multi-server	-	X	X
Adm_fw_host	Hosts in view *Admin_gtw*	-	X	X
Adm_serv_host	Hosts in view *Admin_serv*	-		X

Figure 4. Role description

and H_fw_2. Notice that role $Firewall$ is relevant to H_fw_2 (for administration purpose) but not to H_fw_1. For each role, figure 4 also presents the sub-roles of this role. In this example, the sub-role hierarchy actually corresponds to a specialization role hierarchy.

6.4 Activity

Activities correspond to various services available in corporate network H. We define a first activity all_tcp with different tcp activities (such as $smtp$, ssh and $https$) as sub-activities. Similarly, we define an activity all_icmp with different icmp activities (such as $ping$) as sub-activities. We also define two other activities. *admin_to_gtwy* has to sub-activities: ssh and $ping$. *gtwy_to_admin* has also two sub-activities: ssh and $https$. All these activities are relevant in organizations H, H_fw_1 and H_fw_2.

The main difference here with the approach suggested in [2] is that we use hierarchies of activities whereas Firmato defines elementary services and groups of services. In our approach, permissions and prohibitions all apply to a unique entity, the activity.

6.5 View

Views are used to structure objects on which network services apply. Thus, we define a view called $target$ having two attributes: $content$ that corresponds to messages transmitted when using the service and $dest$ that corresponds to the destination host of the service. The destination host is identified by its role.

Actually, the $content$ attribute is not used in the example because we shall only consider filtering rules on the destination host. However, it would be useful to filter messages depending on their content.

We can then define sub-views derived from view $target$ according to the role assigned to the destination host. For instance, we can define sub-view to_dns as follows:

- $\forall o, Use(H, o, to\_dns) \leftarrow (Use(H, o, target) \wedge dest(o, dns))$

This would lead to define as many views as there are roles. This would be quite fastidious. Instead, we suggest defining a function to_target from roles into views. Views created by function to_target are defined as follows:

- $\forall o, \forall r, Use(H, o, to\_target(r)) \leftarrow (Use(H, o, target) \wedge dest(o, r))$

We consider that a view $to\_target(r)$ is relevant in one of the organization of our example if r is a role relevant in this organization. We also consider that if role r_1 is a sub-role of role r_2, then view $to\_target(r_1)$ is a sub-view of view $to\_target(r_2)$.

6.6 Security policy

We can now specify different permissions that apply to organization H. These permissions correspond to the security policy presented in [2]. Figure 5 lists how these permissions are modelled in Or-BAC. For the sake of simplicity, we do not specify prohibitions and all permissions are supposed to apply in every context (corresponding to $default$ context that is always evaluated to $true$).

Compared to Firmato, one significant advantage of our approach is that it enables us to automatically derive permissions that respectively apply to H_fw_1 and H_fw_2 (using rule OH_1 for deriving permissions in sub-organizations of H). The results we obtain for H_fw_1 is presented in figure 6. To illustrate this derivation process let us consider the following permission:

- $Permission(H, adm\_fw\_host,$
  $admin\_to\_gtwy, to\_target(firewall), default)$

Since role adm_fw_host, activity $admin\_to\_gtwy$ and view $to\_target(firewall)$ are relevant in H_fw_2, we can apply rule OH_1 to derive:

- $Permission(H\_fw_2, adm\_fw\_host,$
  $admin\_to\_gtwy, to\_target(firewall), default)$

However, since view $to\_target(firewall)$ is not relevant in H_fw_1, we cannot derive a similar permission for H_fw_1. But, view $to\_target(external\_firewall)$ is a sub-view of $to\_target(firewall)$. Since $to\_target(external\_firewall)$ is a relevant view in H_fw_1, we can apply rules VH_1 and OH_1 to derive:

- $Permission(H\_fw_1, adm\_fw\_host,$
  $admin\_to\_gtwy, to\_target(external\_firewall), default)$

Notice that one permission, namely:

- $Permission(H, private\_host, all\_tcp, to\_target(public\_host), default)$

is not inherited by H_fw_1 nor H_fw_2. This is because role $private\_host$ is only relevant to H_fw_2 whereas view $target(public\_host)$ is only relevant to H_fw_1. So, no firewall alone can manage this permission. In this case, our proposal is to use this permission to configure both firewalls.

Our approach is actually based on fewer concepts than Firmato. In particular, we do not need to use the notion of "closed" groups. A closed group does not inherit from higher groups in the hierarchy. The example suggested in Firmato is the *firewall* group that should not inherit from *private_host*. We guess that the notion of closed group is complex to manage and actually useless. In our approach, we have simply to specify that *private-net* does not include *firewall-interface* (see the definition suggested for *private-net* in section 6.2). Our approach can also be used to handle more complex applications that include prohibitions, requirements on message contents or contextual rules.

$Permission(H, adm\_fw\_host, admin\_to\_gtwy, to\_target(firewall), default)$
$Permission(H, firewall, gtwy\_to\_admin, to\_target(adm\_fw\_host), default)$
$Permission(H, private\_host, all\_tcp, to\_target(public\_host), default)$
$Permission(H, adm\_server\_host, all\_tcp, to\_target(dns\_server), default)$
$Permission(H, adm\_server\_host, all\_tcp, to\_target(multi\_server), default)$
$Permission(H, public\_host, smtp, to\_target(mail\_server), default)$
$Permission(H, public\_host, dns, to\_target(dns\_server), default)$
$Permission(H, public\_host, ftp, to\_target(ftp\_server), default)$
$Permission(H, public\_host, https, to\_target(web\_server), default)$
$Permission(H, private\_host, smtp, to\_target(mail\_server), default)$
$Permission(H, private\_host, dns, to\_target(dns\_server), default)$
$Permission(H, private\_host, ftp, to\_target(ftp\_server), default)$
$Permission(H, private\_host, https, to\_target(web\_server), default)$
$Permission(H, dns\_server, dns, to\_target(public\_host), default)$
$Permission(H, ftp\_server, ftp, to\_target(public\_host), default)$
$Permission(H, dns\_server, dns, to\_target(private\_host), default)$
$Permission(H, ftp\_server, ftp, to\_target(private\_host), default)$

Figure 5. Permissions in organization H

$Permission(H\_fw_1, adm\_fw\_host, admin\_to\_gtwy, to\_target(external\_firewall), default)$
$Permission(H\_fw_1, external\_firewall, gtwy\_to\_admin, to\_target(adm\_fw\_host), default)$
$Permission(H\_fw_1, public\_host, smtp, to\_target(mail\_server), default)$
$Permission(H\_fw_1, public\_host, dns, to\_target(dns\_server), default)$
$Permission(H\_fw_1, public\_host, ftp, to\_target(ftp\_server), default)$
$Permission(H\_fw_1, public\_host, https, to\_target(web\_server), default)$
$Permission(H\_fw_1, dns\_server, dns, to\_target(public\_host), default)$
$Permission(H\_fw_1, ftp\_server, ftp, to\_target(public\_host), default)$

Figure 6. Permissions in organization H_fw_1

7 Conclusion

In this paper we show how to model inheritance hierarchies in the Or-BAC model. Previous works related to inheritance hierarchies only considered role hierarchies (as suggested in the RBAC model). By contrast, we define role, activity, view and organization hierarchies and analyze inheritance of both permissions and prohibitions through these hierarchies.

Regarding the role hierarchy, we show that it is useful to distinguish between two different hierarchies: the specialization/generalization role hierarchy and the senior/junior role hierarchy. Permissions are inherited "downward" in both hierarchies (the more specialized role inherits from the less specialized role and the senior role inherits from the junior role). However, we suggest that prohibitions are inherited downward in the specialization/generalization hierarchy (as for permissions) whereas they are inherited upward in the senior/junior role hierarchy (the junior inherits from the senior role). Previous proposals did not make such a distinction. For instance, [3] always considers that prohibitions are inherited upward through the role hierarchy. We guess that our proposal eliminate some ambiguities of previous approaches.

Regarding the activity hierarchy, we only consider specialization hierarchy. We may actually define other activity "decomposition" such as decomposing an activity a into $b; c$ denoting activity b followed by activity c. It is clear that if a given role r is permitted to perform activity a, then r should have also permission to perform activity b. However, defining permission associated with c is more complex; role r will be permitted to perform activity c only after having performed activity a. In Or-BAC, we guess that we can represent such a constraint using the notion of context. Modelling such activity decomposition in Or-BAC is an interesting problem that have several applications, in particular to specify security policies for workflow systems [4]. This represents further work that remains to be done.

We also only consider simple view hierarchy corresponding to specialization/generalization. We plan to analyze other relationships between views, in particular the aggregation relationship (also called $Part\_of$ relationship). It would be interesting

to have a general model that defines how permissions and prohibitions propagate from a given view to its different sub-parts. There are several applications to such a model, for instance UML modelling or XML security. However, some difficulties arise, in particular some activities that are relevant to a given view may not apply to some of its sub-parts. This is another problem that requires more investigation.

Finally, we define organization hierarchy and model inheritance of permissions and prohibitions in this hierarchy. We show how this hierarchy is useful to derive, from corporate security policies specification, policies of particular security components such as a firewall. We have implemented a translation module to concretely generate rules to automatically configure the NetFilter firewall. We also plan to apply a similar approach to generate policies of other components such as operating systems or database management systems and to use Or-BAC to model interoperability requirements between these various policies.

Acknowledgement

The work presented in this paper is supported by the RNRT project MP6 and the ACI DESIRS of the French ministry of Research. For this work, Alexandre Miège is funded by France Télécom R&D.

References

[1] G.-J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[2] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *20th IEEE Symposium on Security and Privacy*, pages 17–31, Oakland, California, May 1999.

[3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1), February 2003.

[4] Elisa Bertino, Elena Ferrari, and Vijay Atluri. Specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1), February 1999.

[5] J. Crampton. On permissions, inheritance and role hierarchies. In *10th ACM Conference on Computer and Communication Security*, Washington, November 2003.

[6] F. Cuppens and A. Miège. Conflict management in the or-bac model. Technical report, ENST Bretagne, December 2003.

[7] F. Cuppens and A. Miège. Modelling contexts in the Or-BAC model. In *19th Annual Computer Security Applications Conference*, Las Vegas, December 2003.

[8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):222–274, August 2001.

[9] S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[10] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Come, Italy, June 2003.

[11] J. D. Moffett. Control Principles and Role Hierarchies. In *3rd ACM Workshop on Role-Based Access Control*, October 1998.

[12] J. D. Moffett and E. C. Lupu. The use of role hierarchies in access control. In *4th ACM Workshop on Role-Based Access Control*, October 1999.

[13] Sejong Oh and Ravi Sandhu. A Model for Role Administration Using Organization Structure. In *Seventh ACM Symposium on Access Control Models and Technologies, (SACMAT'02)*, pages 155–162, Monterey, California, USA, june 3–4 2002.

[14] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

An Algebraic Approach to the Analysis of Constrained Workflow Systems

Jason Crampton
Information Security Group, Royal Holloway, University of London

Abstract

The enforcement of authorization constraints such as separation of duty in workflow systems is an important area of current research in computer security. We briefly summarize our model for constrained workflow systems and develop a systematic algebraic method for combining constraints and authorization information. We then show how the closure of a set of constraints and the use of linear extensions can be used to develop an algorithm for computing authorized users in a constrained workflow system. We show how this algorithm can be used as the basis for a reference monitor. We discuss the computational complexity of implementing such a reference monitor and briefly compare our methods with the best existing approach.

Keywords Workflow specification, entailment constraints, linear extensions, satisfiability

1 Introduction

A workflow is a representation of an organizational or business process and is typically specified as a set of tasks and a set of dependencies between the tasks. Dependencies may include authorization constraints such as separation of duty requirements, where two different users must execute two different tasks. There exist several schemes and models in the literature for specifying separation of duty constraints [1, 2, 3, 5, 10, 12] and cardinality constraints [2] in computerized workflow systems.

These schemes are often based on a particular computational model: examples include logic programs [2, 12], active databases [5] and petri nets [1]. We introduced a simple specification scheme for authorization constraints that is independent of an underlying computational model and showed that it could be used to articulate *inter alia* separation of duty constraints and cardinality constraints [6]. In this paper, we exploit the simplicity and uniformity of our scheme to analyze the satisfiability of constrained workflow systems. We also show how this analysis can be used as the basis for a reference monitor for constrained workflow management systems and compare

the computational complexity of our approach with that of Bertino *et al* [2], the most sophisticated existing approach in this area.

In the next section we review our work on modelling workflows and authorization constraints in workflows. Most importantly, we introduce entailment constraints, linear extensions of a workflow specification and methods for combining entailment constraints and authorization information. In Section 3 we introduce the concepts of *satisfiability* in a workflow and the *closure* of a set of entailment constraints. This leads naturally to the development of an algorithm for determining the set of users that are authorized to perform a task and who satisfy the entailment constraints that apply to that task. In Section 4 we briefly describe a reference monitor for workflow systems, using this algorithm as the basis for deciding whether an access request should be granted. Finally we discuss future work.

2 A model for constrained workflows

A *workflow specification* is a partially ordered set of tasks T; if $\mathsf{t} < \mathsf{t}'$ then t must be performed before t' in any instance of the workflow. Let U be a set of users. A *workflow authorization schema* is a pair (T, A), where $A \subseteq \mathsf{T} \times U$ and $(\mathsf{t}, u) \in A$ means that u is *authorized to perform* (or *execute*) t. (Generally, A will not encode task-user pairs directly; often such authorizations will be inferred from the assignment of tasks and users to common roles.)

Let $Rel(U)$ denote the set of all binary relations on U. (In other words, $Rel(U)$ is the powerset of $U \times U$.) Define

$$0' = \{(u, v) : u, v \in U, u \neq v\} \qquad 1' = \{(u, u) : u \in U\}$$
$$0 = \emptyset \qquad 1 = 1' \cup 0' = U \times U$$

An *entailment constraint* has the form $(D, (\mathsf{t}, \mathsf{t}'), \rho)$, where $D \subseteq U$, $\rho \in Rel(U)$ and $\mathsf{t} \not\geqslant \mathsf{t}'$. A *constrained workflow authorization schema* is a triple (T, A, C), where C is a set of entailment constraints.

Informally, if users u and u' perform t and t', respectively, and $u \in D$, then constraint $(D, (\mathsf{t}, \mathsf{t}'), \rho)$ is satisfied iff $(u, u') \in \rho$. (In other words, the constraint is not applied if $u \notin D$. We refer to D as the *domain* of the constraint.) Hence a separation of duty constraint can be expressed as $(U, (\mathsf{t}, \mathsf{t}'), 0')$ and a binding of duty constraint can be expressed as $(U, (\mathsf{t}, \mathsf{t}'), 1')$.

In fact, any binary relation between users can be used (including those that can be derived from contextual information). Hence it is possible to articulate constraints of the form "tasks t and t' must be performed by two different users in the same department". If we assume the existence of group-based or role-based authorization structures, then it is possible to induce an ordering (binary relation) on the set of users determined by the relative seniority of the roles to which each user is assigned. The relation $\ell \in Rel(U)$ will be used to denote an ordering on the set of users, which may

be derived, depending on context, from role information, organizational information or the user groups to which users belong. We anticipate that this sort of relation will prove particularly important, because it is natural to implement access control in workflow systems using role-based techniques.

We have previously shown that cardinality constraints can be expressed as entailment constraints and that role-based authorization constraints should either be expressed as constraints on the authorization information or as entailment constraints based on the relative seniority of users (encoded by the ℓ relation) [6]. In that paper we also provide an example of a constrained workflow authorization schema; lack of space prevents us reproducing the example here.

2.1 Linear extensions and execution schedules

Let $\langle X, \leqslant \rangle$ be a partially ordered set. A *linear extension* of X is a total ordering of the elements of X that respects the ordering of the elements in X. In other words, $\langle X, \preccurlyeq \rangle$ is a linear extension of $\langle X, \leqslant \rangle$ if for all $x_1, x_2 \in X$, either $x_1 \preccurlyeq x_2$ or $x_2 \preccurlyeq x_1$, and if $x_1 \leqslant x_2$ then $x_1 \preccurlyeq x_2$. We denote the set of linear extensions of X by $\mathcal{L}(X)$.

Linear extensions are important in the context of workflows because they "linearize" a partially ordered set of tasks.[1] In other words, a linear extension of T represents a possible sequence of execution of the tasks in a workflow. Figure 2 shows a simple example of a workflow specification and its three linear extensions.

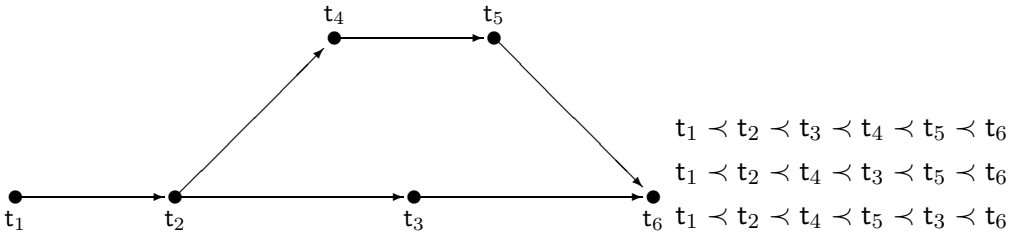


Figure 2: A simple workflow specification and its linear extensions

Definition 1 *Let* (T, A, C) *be a constrained workflow authorization schema. An* execution schedule *for* (T, A, C) *is a pair* (L, α)*, where* $L \in \mathcal{L}(\mathsf{T})$ *and* $\alpha : L \to U$ *assigns tasks to users, such that for all* $\mathsf{t} \in \mathsf{T}$*,* $(\mathsf{t}, \alpha(\mathsf{t})) \in A$*, and for all* $(D, (\mathsf{t}, \mathsf{t}'), \rho) \in C$*,* $\alpha(\mathsf{t}) \in D$ *implies* $(\alpha(\mathsf{t}), \alpha(\mathsf{t}')) \in \rho$*.*

[1] We note that in certain circumstances, it will be possible for certain tasks in a workflow to execute in parallel. Specifically, if t and t' are tasks with $\mathsf{t} \parallel \mathsf{t}'$ and neither t nor t' appears in any constraint, then they may be executed in parallel. Such situations are outside the scope of this paper.

In other words, an execution schedule respects the relative ordering of tasks in the workflow specification (since it is a linear extension of T), every task is performed by an appropriately authorized user and every entailment constraint is satisfied. A constrained workflow authorization schema is *satisfiable* if there exists an execution schedule for the schema (and *unsatisfiable* otherwise).

In general, the set of linear extensions in T can be generated in time $\mathcal{O}(|\mathcal{L}(\mathsf{T})|)$ [11] and computing $|\mathcal{L}(\mathsf{T})|$ is #P-complete [4]. However, if the width of the poset is small (as will be the case for a typical workflow specification), then the set of linear extensions can be computed quickly using dynamic programming techniques. We now discuss this is in more detail.

Proposition 2 *Suppose $\langle X, \leqslant \rangle$ is a poset and $x_1 \prec x_2 \prec \cdots \prec x_n$ is a linear extension of X. Then $\{x_1, \ldots, x_k\}$ is an order ideal in X, $1 \leqslant k \leqslant n$.*

Proof Suppose $\{x_1, \ldots, x_k\}$ is not an order ideal. Then there exists $y \in X$ such that $y \leqslant x_j$ for some j, $1 \leqslant j \leqslant k$, and $y \notin \{x_1, \ldots, x_k\}$. Therefore $y \leqslant x_j$ and $x_j \prec y$; hence $\{x_1, \ldots, x_k\}$ is not a linear extension and the result follows by contradiction. ■

Hence each linear extension is a directed path of maximal length in the graph of $\mathcal{I}(\mathsf{T})$, the lattice of order ideals of T.

Lemma 3 *Let $\langle X, \leqslant \rangle$ be a poset and let $\mathcal{I}(X)$ denote the set of order ideals in X. Then*

$$|\mathcal{I}(X)| \leqslant \left(\left\lceil \frac{|X|}{w} \right\rceil + 1 \right)^w,$$

where w is the width of X.

Proof By Dilworth's theorem [8], we can partition the poset $\langle X, \leqslant \rangle$ into w disjoint chains $C_1, \ldots, C_w$. Consider the poset $\langle X, \trianglelefteq \rangle$, where $x \trianglelefteq y$ iff $x, y \in C_i$ for some i and $x \leqslant y$ (in X). Then any order ideal in $\langle X, \leqslant \rangle$ is an order ideal in $\langle X, \trianglelefteq \rangle$. To see this, note that there is an isomorphism between the set of order ideals and the set of antichains, where an order ideal is mapped to the antichain comprising the maximal elements in the order ideal [7]. It is clear that any antichain in $\langle X, \leqslant \rangle$ must also be an antichain in $\langle X, \trianglelefteq \rangle$ by construction. In other words, the number of antichains in $\langle X, \leqslant \rangle$ (and hence the number of order ideals) is bounded by the number of antichains in $\langle X, \trianglelefteq \rangle$.

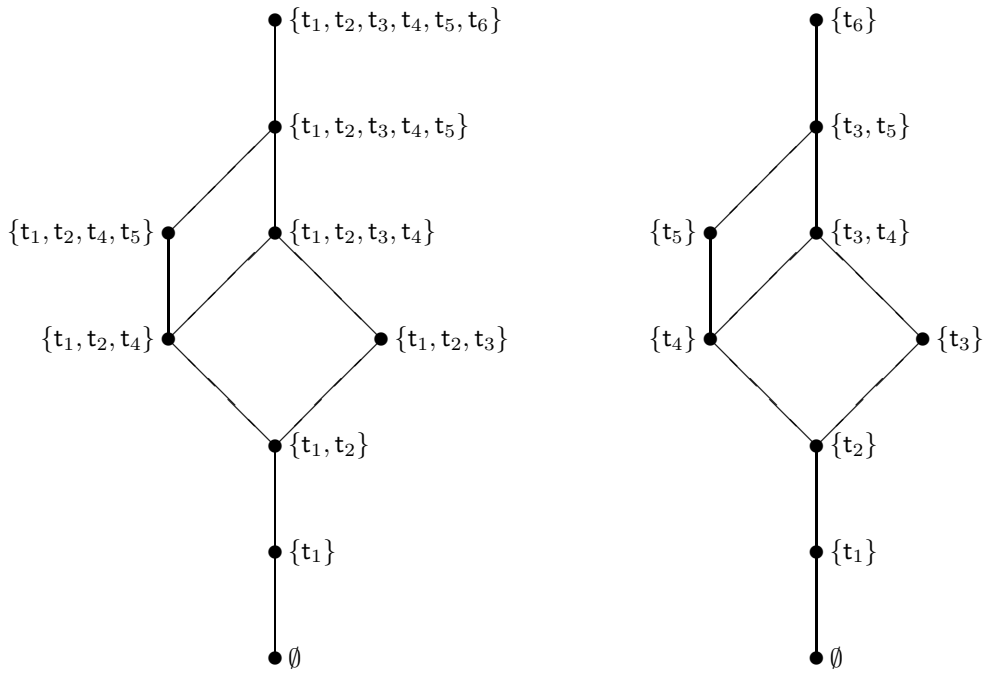
The number of antichains in $\langle X, \trianglelefteq \rangle$ is equal to $\prod_{i=1}^{w} (|C_i| + 1)$ (because we can choose at most one element from each chain in $\langle X, \trianglelefteq \rangle$). It is easy to show using elementary calculus that the product xy, subject to $x + y = k$, is maximized when $x = y = k/2$. Generalizing this result, we obtain

$$\prod_{i=1}^{w} (|C_i| + 1) \leqslant \prod_{i=1}^{w} \left(\left\lceil \frac{|X|}{w} \right\rceil + 1 \right).$$

The result follows. ■

Hence the number of order ideals in T is bounded by $\left(\left\lceil \frac{|\mathsf{T}|}{w} \right\rceil + 1\right)^w$, where w is the width of T, and the directed paths can be computed using a breadth-first search whose complexity is linear in the number of nodes of the graph. In other words, if w is small, the number of order ideals can be computed in time polynomial in the number of tasks in the workflow specification.

Figure 3 shows the lattice of order ideals and the lattice of antichains for the workflow specification depicted in Figure 2. (We have adopted the usual convention in Hasse diagrams that $x \leqslant y$ implies y is above x in the diagram.) In our example $w = 2$ and the number of order ideals is 9.



(a) Order ideals (b) Antichains

Figure 3: Lattices derived from the workflow specification in Figure 2

2.2 The algebra of entailment constraints

In this section we state without proof a number of simple results concerning entailment constraints. The reader is referred to our earlier work for further details [6]. We

conclude the section with a new result that enables us to omit authorization information from a workflow schema, thereby facilitating the analysis of workflow systems.

Proposition 4 (Merging domains) *Let* $(\mathsf{T}, A, \{(D_1, (\mathsf{t}, \mathsf{t}'), \rho), (D_2, (\mathsf{t}, \mathsf{t}'), \rho)\}$ *be a constrained workflow authorization schema. Then* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(D_1, (\mathsf{t}, \mathsf{t}'), \rho), (D_2, (\mathsf{t}, \mathsf{t}'), \rho)\}$ *iff* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{D_1 \cup D_2, (\mathsf{t}, \mathsf{t}'), \rho\})$.

Proposition 5 (Expanding the domain) *Let* $(\mathsf{T}, A, \{(D, (\mathsf{t}, \mathsf{t}'), \rho)\})$ *be a constrained workflow authorization schema and define* $\sigma = (U \setminus D) \times U$. *Then* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(D, (\mathsf{t}, \mathsf{t}'), \rho)\})$ *iff* (L, α) *is a workflow execution schedule for* $(\mathsf{T}, A, \{(U, (\mathsf{t}, \mathsf{t}'), \rho \cup \sigma)\})$.

Proposition 6 (Merging constraints) *Let* $W = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}, \mathsf{t}'), \rho_2)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is an execution schedule for* W *iff* (L, α) *is an execution schedule for* $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1 \cap \rho_2)\})$.

In other words, we can assume that the domain of every constraint is U (by Proposition 5), and that for each pair of tasks $(\mathsf{t}, \mathsf{t}')$ there is a single constraint (by Proposition 6).

Proposition 7 (Composing constraints) *Let* $W = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is an execution schedule for* W *iff* (L, α) *is an execution schedule for* $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2), ((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2)\})$, *where*

$$\rho_1\rho_2 = \{(u, w) : \exists v \in U, (u, v) \in \rho_1, (v, w) \in \rho_2\}.$$

It is important to note that if (L, α) is an execution schedule for $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2)\})$, then it is not necessarily an execution schedule for $(\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2)\})$. (Although we have $(\alpha(\mathsf{t}), \alpha(t'')) \in \rho_1\rho_2$, we can not necessarily infer that there exists an authorized user for t'.) In other words, we cannot delete the constraints from which a compound constraint is derived. Unfortunately, the composition of relations is neither commutative nor associative. However, for each linear extension of the workflow there is a unique order in which the relations are composed.

At the moment we only consider entailment constraints to be specifications of security policy requirements such as separation of duty. However, we can view the authorization information as a set of entailment constraints on the execution of tasks. In particular, let $\mathsf{T} = \{\mathsf{t}_1, \ldots, \mathsf{t}_n\}$ and for all $\mathsf{t}_i \not\geqslant \mathsf{t}_j$, define $a_{ij} = U(\mathsf{t}_i) \times U(\mathsf{t}_j)$, where $U(\mathsf{t}) = \{u \in U : (\mathsf{t}, u) \in A\}$. Then the entailment constraint $((\mathsf{t}_i, \mathsf{t}_j), a_{ij})$, is only satisfied if two appropriately authorized users perform tasks t_i and t_j. If there exists an entailment constraint of the form $((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij})$ then we form the new constraint $((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij} \cap a_{ij})$. More formally, we have the following result. The proof

of this result follows immediately from the definition of an execution schedule and is omitted.

Proposition 8 (Incorporating authorization information) *Let* $\mathsf{W} = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is an execution schedule for* W *iff* (L, α) *is an execution schedule for* $(\mathsf{T}, \mathsf{T} \times U, \{((\mathsf{t}, \mathsf{t}'), \rho \cap a)\})$*, where* $a = \{(u, u') : (\mathsf{t}, u), (\mathsf{t}', u') \in A\}$.

In other words, we can express all the information required to make an authorization decision in terms of entailment constraints. Hence it is sufficient from a theoretical point of view to consider workflow schemata of the form (T, C), although, from a practical perspective, it is clearly more natural to include authorization information.

3 Satisfiability in workflow systems

There are three questions that are of interest:

- Is a constrained workflow authorization schema satisfiable? Given a constrained workflow authorization schema (T, A, C), is it possible for some instance of the workflow to complete. In other words, is there an assignment of tasks to users $\alpha : \mathsf{T} \to U$ and a linear extension $L \in \mathcal{L}(\mathsf{T})$ such that (L, α) is an execution schedule.

- Is an instance of a workflow schema satisfiable? We write t to denote an instance of the task $\mathsf{t} \in \mathsf{T}$. In other words, given a constrained workflow authorization schema (T, A, C) and an order ideal $I \subseteq \mathsf{T}$, where $t_i \in I$ has been executed by u_i, is it possible to extend the ideal to a linear extension of T and to find an assignment of the remaining tasks to users such that the resulting linear extension and the assignment of tasks to users forms an execution schedule for (T, A, C).

- Given a satisfiable workflow authorization schema, is it possible to design a reference monitor so that every instance of that schema is satisfiable? In other words, is it possible to design a decision process that only permits a request from a user to execute a task if the remaining tasks can be completed. Clearly, an answer to the previous question will provide a blueprint for the design of such a reference monitor.

3.1 The closure of a set of constraints

Let $\mathsf{W} = (\mathsf{T}, C)$ be a constrained workflow authorization schema. We assume that $c_{ij} = ((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij})$ is defined whenever $\mathsf{t}_i \not\geqslant \mathsf{t}_j$ (setting $\rho_{ij} = 1$ where necessary) and

define

$$C^2 = C \cup \{((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2) : ((\mathsf{t}, \mathsf{t}'), \rho_1), ((\mathsf{t}', \mathsf{t}''), \rho_2) \in C\},$$
$$C^k = C^{k-1} \cup \{((\mathsf{t}, \mathsf{t}''), \rho_1\rho_2) : ((\mathsf{t}, \mathsf{t}'), \rho_1) \in C^{k-1}, ((\mathsf{t}', \mathsf{t}''), \rho_2) \in C\}.$$

Let $|\mathsf{T}| = n$. Then C^{n-1} denotes the set of all possible constraints that can be derived from the original set of constraints C using composition. Let $c_{ij}^k \in C^k$ denote the constraint $((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij}^k)$.[2] Define the *closure* of C, denoted C^*, to be the set of constraints $c_{ij}^* = ((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij}^*)$, where $\mathsf{t}_i \not\geqslant \mathsf{t}_j$ and $\rho_{ij}^* = \bigcap_{k=1}^{n-1} \rho_{ij}^k$, $1 \leqslant i, j \leqslant n$. If a pair of users satisfy a constraint $((\mathsf{t}, \mathsf{t}'), \rho) \in C^*$, then for every linear extension of T, there exists a sequence of users that can execute the sequence of tasks between t and t'.

Informally, we can regard C as a labelled directed graph in which the set of nodes is T and an edge $(\mathsf{t}_i, \mathsf{t}_j)$ labelled ρ_{ij} exists if $\mathsf{t}_i \not\geqslant \mathsf{t}_j$. The constraints in C^k are the paths of length k in this graph. In fact, we can realize C as a matrix (in which the ijth entry is ρ_{ij}) and "multiply" the matrix by itself $n-1$ times to derive $C^2, \ldots, C^{n-1}$.

Theorem 9 *Let (T, C) be a constrained workflow authorization schema and let $\mathsf{t} \in \mathsf{T}$ be a minimal element and $\mathsf{t}' \in \mathsf{T}$ be a maximal element. Then $((\mathsf{t}, \mathsf{t}'), \rho) \in C^*$ for some $\rho \subseteq 1$ and (T, C) is satisfiable if $\rho \neq \emptyset$.*

Proof sketch There exists a "path" of length $n-1$ between t and t' and hence there exists a constraint of the form $((\mathsf{t}, \mathsf{t}'), \rho)$. A simple induction (using Propositions 6 and 7 for the base case) shows that every constraint on that path is satisfied. Hence if ρ is non-empty, then there exists a pair of users that can perform the first task and last task and a sequence of users that satisfies all the constraints in between. In other words, there exists an execution schedule for (T, C). ■

Unfortunately, it is rather difficult to compute the closure of C in this way directly because the graph of C is not acyclic.[3] More specifically, we need to be able to distinguish between constraints that arise because of paths (in which each node is visited at most once) and those that arise because of walks (in which a node may be visited more than once). One possible way of doing this is to compute the length of the longest path between each pair of tasks and omit any constraints that arise due to a walk between a given pair of tasks which exceed this length. The computation of the longest path between a pair of nodes in a directed graph is NP-complete [9, Problem ND29].

Alternatively, given a workflow schema $\mathsf{W} = (\mathsf{T}, C)$, we can enumerate all possible linear extensions, thereby creating a family of workflow schemata $\mathcal{W}$ in which

[2] There may be more than one constraint that can be derived for tasks t_i and t_j. In this case we simply take the intersection of the relations for each of these constraints to derive a single constraint.

[3] For example, if we define the constraints $((\mathsf{t}_3, \mathsf{t}_4), 0')$ and $(\mathsf{t}_4, \mathsf{t}_3), 0')$ for the workflow in Figure 2, meaning that the same user cannot perform both t_3 and t_4, then we have a cycle of length 2 in the graph.

each specification is a totally ordered set of tasks. For each such schema we compute the closure of the set of constraints. Finally, we can create a single workflow schema $\mathsf{W}^* = (\mathsf{T}, C^*)$, where for all $((\mathsf{t}_i, \mathsf{t}_j), \rho_{ij} \in C^*$, ρ_{ij} is obtained by taking the intersection of the relations in every constraint of the form $((\mathsf{t}_i, \mathsf{t}_j), \rho)$ in $\mathcal{W}$.

3.2 An algorithm for computing execution schedules

Figure 4 illustrates an algorithm (written in pseudo-code) that computes $V(\mathsf{t}, \mathsf{t}')$ for each pair $(\mathsf{t}, \mathsf{t}')$, where $V(\mathsf{t}, \mathsf{t}')$ is the set of users that can execute t and t' (in that order) given the authorization information and the entailment constraints in the schema that apply to t and t'. The basic strategy is to initialize each $V(\mathsf{t})$ to the set of users that are authorized to perform T (line `02`) and then, for each linear extension, to apply all the possible constraints (including those derived from authorization information) (lines `07`–`08`). Essentially, the algorithm is applying Proposition 8 and computing a new relation for each entailment constraint. If one of these relations is empty, then the algorithm terminates prematurely (line `08`), since there does not exists a pair of authorized users that comply with the entailment constraints. Finally, for each task t we (re-)compute the set of users that can perform t (lines `10`–`11`).

```
01  for i = 1 to |T|
02   let V(i) = set of users authorized to perform task i
03  for each linear extension
04   for i = 1 to |T|
05    for j = 1 to |T|
06     if ((i, j), R) ∈ C
07      let V(i,j) = (V(i) × V(j)) ∩ R
08      if V(i,j) is empty then exit
09      else
10       let V(i) = set of users in first position of V(i,j)
11       let V(j) = set of users in second position of V(i,j)
```

Figure 4: An algorithm for determining whether an execution schedule exists

The overall time complexity of the algorithm is $\mathcal{O}(|T|^w |T|^2 |U|^4) = \mathcal{O}(|T|^{w+2} |U|^4)$, since the number of linear extensions is $\mathcal{O}(|T|^w)$ (see Section 2.1), the number of constraints is $\mathcal{O}(|T|^2)$ and the comparison in line `07` is $\mathcal{O}(|U|^4)$ in the worst case. Note that the computational complexity of the comparison in line `07` dominates the complexity of the computations required in lines `10` and `11`, which are $\mathcal{O}(|U|^2)$. Note also that if R is $0'$ or $1'$, then the computation in line `07` is a simple comparison of $V(i)$ and $V(j)$ and hence has time complexity $\mathcal{O}(|U|^2)$. In other words, if we restrict our attention to cardinality, separation of duty and binding of duty constraints, then the overall complexity reduces to $\mathcal{O}(|\mathsf{T}|^{w+2} |U|^2)$. (Recall that cardinality constraints can be modelled using separation of duty constraints [6].)

4 A reference monitor for constrained workflows

A *workflow system* is a pair $\mathcal{S} = (\mathcal{W}, \mathcal{M})$, where $\mathcal{W}$ is a set of workflow schemata and $\mathcal{M}$ is a reference monitor. A *reference monitor* is an abstract machine for deciding whether an access request from a user will be granted. A workflow *instance* is created (instantiated) when the first task in some linear extension of T is executed.

Let $\mathsf{W} = (\mathsf{T}, A, C)$ be a constrained workflow authorization schema. Then we denote an instance of this schema by W and an instance of task $\mathtt{t}$ by t. A workflow instance completes if every task in the workflow specification is performed by some user.

We will say that a workflow system is *complete* if every instance of every schema is guaranteed to complete. A workflow instance, in general, will not complete because the execution of certain tasks by certain users and the existence of entailment constraints in the schema may restrict the users that can perform subsequent tasks. Hence, a workflow system will be complete only if the reference monitor is able to identify and deny tasks that would prevent subsequent tasks from being executed because certain entailment constraints could not be satisfied. However, a reference monitor that guarantees a workflow system is complete is likely to be computationally expensive [2].

In the context of workflow systems, a user requests the permission to execute a task in a workflow instance. In other words, $\mathcal{M}$ is a function that takes a triple (t, i, u) and returns `allow` if the request is granted and `deny` otherwise. The triple (t, i, u) is interpreted as a request by user u to execute task $\mathtt{t} \in \mathsf{T}$ in W_i, the ith instance of $\mathsf{W} = (\mathsf{T}, A, C)$.

Let W be a workflow instance in which all the tasks in $\mathsf{T}' \subseteq \mathsf{T}$ have been executed, where T' is an order ideal in T. Then this workflow instance can be represented as a function $I : \mathsf{T}' \to U$, where user $I(\mathtt{t})$ performed task $\mathtt{t}$. The execution of a workflow instance is constrained by I and C. Given a workflow authorization schema (T, A, C), let $\mathsf{W}|I$ denote the workflow schema $(\mathsf{W}, A|I, C)$, where

$$A|I = \{(\mathtt{t}, I(\mathtt{t})) : \mathtt{t} \in \mathsf{T}'\} \cup \{(\mathtt{t}, u) \in A : \mathtt{t} \in \mathsf{T} \setminus \mathsf{T}'\}.$$

In other words, $\mathsf{W}|I$ is a constrained workflow authorization schema in which each task $\mathtt{t} \in \mathsf{T}'$ has a single authorized user $I(\mathtt{t})$; that is, the user that performed $\mathtt{t}$ in instance I.

In general, given a partially completed workflow instance I and a request by u to execute $\mathtt{t}$ in this instance there are three questions a reference monitor could consider:

Q1 Is u authorized to perform $\mathtt{t}$?

Q2 Are all constraints in which $\mathtt{t}$ is the consequent task satisfied?

Q3 Can the workflow complete if u performs t?

The reference monitor must certainly guarantee that the answers to the first two questions are yes. It is up to the designers of the reference monitor to decide whether the third question should always have an affirmative answer. Indeed, some research has been done on overriding (that is, not enforcing) constraints in the event that a workflow cannot complete because of previous task executions and the existence of constraints [12]. We say a reference monitor is *enforcement compliant* if it guarantees (for all requests) that the answers to the first two questions are yes and *completion compliant* if it guarantees that the answer to each of the three questions is yes.

Let $\mathsf{W} = (\mathsf{T}, A, C)$ be a constrained workflow authorization schema. In order to implement a reference monitor for this workflow, we compute C^* and use the algorithm in Figure 4 to establish that an execution schedule for the workflow exists and to compute a relation $V \subseteq A \subseteq \mathsf{T} \times U$, where $(\mathsf{t}, u) \in V$ implies that u is authorized to perform t and all entailment constraints can be satisfied. We write $V(\mathsf{t})$ to denote the set $\{u \in U : (\mathsf{t}, u) \in V\}$.

Let us first consider the case where $\mathsf{t} \in \mathsf{T}$ is a minimal element (and hence can be the first task executed in a workflow). In this case, $I = \emptyset$ and $\mathsf{W}|I = \mathsf{W}$.[4] Then a request to execute t may be granted if $(\mathsf{t}, u) \in V$. If the request were to be granted, then $I = \{(\mathsf{t}, u)\}$; a completion compliant reference monitor must recalculate V for the workflow $\mathsf{W}|\{(\mathsf{t}, u)\}$. If $V(\mathsf{t}') = \emptyset$ for some $\mathsf{t}' \in \mathsf{T}$ then the workflow schema, and hence the workflow instance, cannot be satisfied. Hence, in order to implement a completion compliant reference monitor, we simply run the algorithm in Figure 4 for the workflow $\mathsf{W}|\{(\mathsf{t}, u)\}$ *before* granting the request (t, i, u). If the request is granted, the next request must be evaluated for the workflow $\mathsf{W}|\{(\mathsf{t}, u)\}$.

In the general case, let I be an instance of $\mathsf{W} = (\mathsf{T}, A, C)$ and let $I \cup \{\mathsf{t}\}$ be an order ideal in T. Then a request by u to execute t in this instance of W is granted by a completion compliant reference monitor if there exists an execution schedule for $\mathsf{W}|I$ such that u executes t and there exists an execution schedule for $\mathsf{W}|(I \cup \{(\mathsf{t}, u)\})$. In other words, we simply run the algorithm in Figure 4 for the workflow $\mathsf{W}|(I \cup \{(\mathsf{t}, u)\})$ *before* granting the request (t, i, u). If the request is granted, the next request must be evaluated for the workflow $\mathsf{W}|(I \cup \{\mathsf{t}, u\})$.

In summary, a completion compliant reference monitor must calculate V before any instance of the workflow is created. The reference monitor must also re-calculate V before every request (to check that the request is completion compliant) and update the workflow after every successful request (to ensure that the fact that a particular user executed a particular task is considered in enforcing constraints that apply to subsequent tasks).

The only comprehensive treatment of completion compliant workflow systems in the literature [2] includes an algorithm for "user planning", which associates tasks with a user-role pair. The complexity of this algorithm is $\mathcal{O}((N_R \cdot N_U \cdot N_{act})^{|\mathsf{T}|})$, where N_R is the maximum number of roles associated with any task in the workflow, N_U is

[4] $I = \emptyset$ in the sense that there are no pairs $(\mathsf{t}, I(\mathsf{t}))$ defined.

the maximum number of users associated with any role in the workflow and N_{act} is the maximum number of activations associated with any task in the workflow. This algorithm is run before any workflow instance is created and, in the worst case, is also run when a request is received and when a task has been successfully executed. Our algorithm has time complexity $\mathcal{O}(|T|^{w+2} \cdot |N_U|^4)$. It has better time complexity than the user planner algorithm for several reasons:

- firstly, we only consider entailment constraints, which makes the analysis of constraints uniform and hence simpler;
- secondly, we only consider user-based constraints (having shown that role-based constraints can be enforced in other ways [6]);
- thirdly, we do not compute every possible sequence of tasks and users, instead computing the closure and determining if a workflow instance can complete (without explicitly calculating a sequence of tasks and users).

We note that the performance of the user planner algorithm can be improved by adopting certain heuristics, but the worst case complexity is still exponential in the number of tasks in the workflow specification.

5 Concluding remarks

The analysis of satisfiability in this paper suggests that there are distinct advantages to our approach to authorization constraints in workflow systems. We believe our approach has the following advantages over existing approaches:

- The ability to treat most, if not all, useful authorization constraints as special cases of entailment constraints means that the analysis of a set of authorization constraints for a workflow is greatly simplified.
- The ability to express authorization information in terms of entailment constraints means that satisfiability questions can be analyzed entirely in the context of the closure of a set of entailment constraints.
- The fact that our model for constraints is independent of any underlying computational model coupled with its simplicity means that it can be easily implemented in a variety of ways.

There are numerous opportunities for further research. Perhaps the most obvious of these is a prototype implementation, perhaps using an off-the-shelf relational database management system, to assess the usability and scalability of our approach.

In many workflow models, a task may be repeated several times within a workflow. When the number of occurrences of the task is fixed in each instance of the

workflow, this can be modelled using cardinality constraints. However, in other models, the number of occurrences is allowed to vary. This clearly makes the analysis of satisfiability and the design of a reference monitor for such systems more complex. However, we believe that extending our model to include an entailment constraints of the form $((\mathsf{t}, \mathsf{t}), \rho)$ may provide a suitable platform for investigating such workflow systems. The investigation of this topic will be one of our immediate priorities in future research.

Another avenue for further work is to consider substituting "proxy users" for actual users in the analysis of the workflow schema. The complexity of the algorithm is polynomial in the number of users and it is likely that the number of tasks will be considerably smaller than the number of users. A proxy user is simply identified with a subset of tasks (and hence is synonymous with a role). This means that proxy users can also be used to derive a role hierarchy for the workflow.

The number of proxy users is certainly bounded by $2^{|\mathsf{T}|}$, and we would expect the number of roles to be bounded by the number of tasks. However, we also need to consider "compound roles" consisting of tasks assigned to two or more roles. (For example, we might identify that $\{\mathsf{t}_1, \mathsf{t}_2\}$ naturally form one role and $\{\mathsf{t}_3, \mathsf{t}_4\}$ form another. We need to allow for the fact that a user may be assigned to both roles and so there must be a proxy user for the set $\{\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, \mathsf{t}_4\}$.) Hence, in general the number of proxy users will actually be bounded by the number of antichains in the role hierarchy. We would expect that the number of roles is less than $|\mathsf{T}|$. Hence we can run the algorithm in Figure 4 using proxy users rather than actual users, thereby reducing the time complexity of the algorithm to $\mathcal{O}(|T|^{W+w+2})$, where W denotes the width of the role hierarchy.

Acknowledgements We would like to thank Frank Ruskey for his helpful comments on generating linear extensions.

References

[1] V. Atluri and W. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64, 1996.

[2] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[3] R.A. Botha and J.H.P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[4] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.

[5] F. Casati, S. Castano, and M. Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, 2001. Technical Report HPL-2000-156, Hewlett Packard Laboratories.

[6] J. Crampton. On the satisfiability of authorization constraints in workflow systems. Technical Report RHUL–MA–2004–1, Department of Mathematics, Royal Holloway, University of London, 2004. `http://www.ma.rhul.ac.uk/techreports/`.

[7] J. Crampton and G. Loizou. The completion of a poset in a lattice of antichains. *International Mathematical Journal*, 1(3):223–238, 2001.

[8] R.P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–6, 1950.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, California, 1979.

[10] K. Knorr and H. Stormer. Modeling and analyzing separation of duties in workflow environments. In *Trusted Information: The New Decade Challenge, IFIP TC11 Sixteenth Annual Working Conference on Information Security*, pages 199–212, 2001.

[11] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.

[12] J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC – A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(4):455–486, 2003.

Session III

Security Protocols I

CPL: An Evidence-Based 5-Dimensional Logic for the Compositional Specification and Verification of Cryptographic Protocols
Part I: Language, Process Model, Satisfaction

Simon.Kramer@a3.epfl.ch

Abstract

We (1) define a logic[1], called *CPL* (for *Cryptographic Protocol Logic*), where truth is established on the grounds of evidence-based knowledge (as opposed to awareness-based belief), spanning the dimensions[2] of first-order, temporal, epistemic, deontic, and linear logic; (2) state a few of its key properties; and (3) illustrate how it can be used to compositionally specify and verify cryptographic protocols designed to establish trust in the security of communication (as opposed to security of storage) between protocol-compliant participants in a hostile environment. Our claim hereby is to give (1) the first formalisation of cryptographic discourse within the framework of multi-dimensional logic, (2) the most comprehensive, logically connected formal model of cryptographic protocols proposed so far, and (3) a rigourous clarification of the concepts constituting the common knowledge of the community of protocol designers.

Keywords Modal and linear logics, process calculi

1 Introduction

Motivation Protocol designers commonly specify a cryptographic protocol jointly by (1) a semi-formal description of its *behaviour* (local properties) in terms of *protocol narrations*, and by (2) an informal prescription of its intended *goals* (global properties) in *natural language*. Informal specifications present three major drawbacks: (1) they

[1] announced in [16]

[2] cf. [12] for a research monograph on multi-dimensional modal logic, characterised in [5] as

> ...a branch of modal logic dealing with special relational structures in which the states, rather than being abstract entities, have some inner structure. ...Furthermore, the accessibility relations between these states are (partly) determined by this inner structure of the states.

do not have a well-defined, and thus a well-understood *meaning*; (2) they do not allow for the verification of *internal correctness* (referring to an internal notion of truth), i.e., the virtue that the conjunction of local properties implies each global property, typically by means of a *proof system*; and (3) they do not allow for the verification of *external correctness* (referring to an external notion of truth requiring a formal protocol model), i.e., the virtue that a proposed implementation (protocol model) satisfies each global property, typically by means of *model checking*.

In *formal* specifications of cryptographic protocols, local and global properties are expressed either explicitly *as such* in terms of a logical (or property-based) language, or implicitly *as code*, resp. *as encodings* in a protocol modelling (or model-based) language. Examples of such encodings are equations between instantiations of protocol schemata, and predicates defined inductively on the traces those instantiations may exhibit [1]. However, such encodings present four major drawbacks: (1) they have to be found; worse, (2) they may not even exist; (3) they are neither directly comparable with other encodings in the same or in other protocol modelling languages, nor with properties expressed explicitly in terms of logical languages; and (4) they are difficult to understand because the intuition of the encoded property is implicit in the encoding.

Informal language and protocol modelling languages are patently inadequate for expressing and comparing cryptographic properties. It is our belief that only a logical language equipped with an appropriate notion of truth, i.e., a cryptographic *logic*, will produce the necessary adequacy therefore. A number of logics have been proposed in this aim so far, ranging from ad-hoc special-purpose cryptographic logics [7, the so-called BAN-logic] and [23, a unification of several BAN-logics], over classical first- and higher-order, modal, and linear logics used for the special purpose of cryptographic protocol analysis [9, 22, first-order], [21, higher-order], [14, temporal modalities], [15, 8, epistemic modalities], and [20, deontic modalities], resp. [6, linear], to combinations thereof, e.g., [4, epistemic, temporal and program modalities] and [11, epistemic post-conditions]. However in our opinion and w.r.t. our understanding of adequacy, each of these logics fails to be adequate due to limitations of *scope* (and *style*), i.e., the power to express (*intuitively*[3], *succinctly*, and *endogenously*[4]) arbitrary cryptographic goals, and/or *grain*, i.e., the power to discriminate sufficient detail in the analysis of cryptographic protocols. These limitations originate in *design decisions* of syntactical (language-defining *operators*) and/or semantic (meaning-defining *notion of truth*) nature.

[3] meaning that the conceptual dimensions of the goal are evident in distinctive forms in the formula that expresses it

[4] an endogenous (as opposed to *exogenous*) logical language is purely property-based (pure in the sense that the language is free from model-based forms, e.g., program fragments). Classical examples of endogenous and exogenous logical languages are LTL and CTL, resp. Hoare Logic and Dynamic Logic. The terms are due to Harel, Kozen, and Tyurin.

Goal Our goal is to supply a logic that allows one to (1) *express* (cf. Section 2.3) and *compare* (cf. Section 4.4) arbitrary cryptographic properties intuitively, succinctly, and in an endogenous fashion, and to (2) *verify* correctness of cryptographic goals on cryptographic protocols up to a fine (though for the moment still *formalistic*[5]) grain of detail. Our design decision thereby is to equip the logic with (1) seven novel basic operators and a selection of classical operators from *first-order*, *temporal*, *epistemic*, *deontic*, and *linear* logics (cf. Section 2.2); and, in a first step, with (2) a novel external notion of truth defined through satisfaction (cf. Section 4) in terms of *cryptographic processes* (cf. Section 3).

JUSTIFICATION A cryptographic protocol involves the concurrent interaction of participants that are physically separated by — and exchange messages across — an unreliable and insecure transmission medium. It is folklore that expressing properties of concurrent interaction requires temporal modalities. The physical separation by an unreliable and insecure transmission medium in turn demands the epistemic and deontic modalities. To see why, consider that the existence of such a separation and medium introduces an *uncertainty* among protocol participants about the *trustworthiness* of the execution of *communication acts* (sending and receiving) and the contents of exchanged *messages*, both w.r.t. *actuality* (an epistemic concern) and *legitimacy* (a deontic concern). It is exactly the role of a cryptographic protocol to re-establish this trustworthiness through the judicious use of *cryptographic evidence* (e.g., ciphers, signatures and hash values) bred in a *crypto(graphic) system* (e.g., shared-key or public-key cryptography) and from *cryptographic germs* (e.g., keys and[6] nonces). However, any use of keys (as opposed to hash values and nonces) requires that the knowledge of those keys be shared a priori. This sharing of key knowledge is established by cryptographic protocols called *key establishment protocols* [19, Chapter 12], which are executed before any cryptographic protocol that may then subsequently use those keys. Thus certain cryptographic protocols must be considered interrelated by a notion of *sequential composition*, which in turn induces a notion of *linearity* between the properties associated with those protocols. For example, the secrecy of a key k is a linearly-necessary condition for the secrecy of any datum d encrypted under that key in the following sense: only if k is secret in the protocol that distributes it as well as in the protocol p that uses it to encrypt d, d is secret in p.

We give priority to the definition of an external notion of truth because we opine that such a notion is practically more relevant, especially when defined through satisfaction in terms of practically executable processes.

[5] as opposed to *complexity-* and *information-theoretic*

[6] themselves generated from *cryptographic seeds* (or seed values)

2 Language

The language $\mathcal{F}_{\mathcal{M}}$ of CPL is parametric in the language of its individuals, i.e., protocol messages $\mathcal{M}$, which may mention protocol participants, keys, nonces, and — as a novelty — also *message types*. It is chiefly relational, and functional in exactly the language of messages $\mathcal{M}$ it may be instantiated with. The temporal fragment of $\mathcal{F}_{\mathcal{M}}$ coincides with the syntax of CTL* with past.

2.1 Individuals

Our individuals shall be *participants* $a, b, c \in \boldsymbol{\mathcal{P}}$, *shared keys* $k \in \boldsymbol{\mathcal{C}}$, *public keys* $k^+ \in \boldsymbol{\mathcal{C}}^+$, *private keys* $k^- \in \boldsymbol{\mathcal{C}}^-$, *nonces* $x \in \boldsymbol{\mathcal{X}}$, *messages* $M \in \mathcal{M}$, and *message types* $\sigma \in \mathcal{T}$. Further, $v \in \mathcal{V}$ shall denote variables for individuals, $F \in \mathcal{M}^{\mathrm{f}}$ so-called *message forms*, and $\theta \in \mathcal{T}^{\mathrm{f}}$ so-called *message type forms*. $n \in \boldsymbol{\mathcal{N}} := \boldsymbol{\mathcal{P}} \cup \boldsymbol{\mathcal{C}} \cup \boldsymbol{\mathcal{C}}^+ \cup \boldsymbol{\mathcal{C}}^- \cup \boldsymbol{\mathcal{X}}$ shall be referred to as a *name* or *constant*, and shared and private keys as *confidential keys* $\boldsymbol{\mathcal{C}}^{\times}$, i.e., keys that must remain secret. Message forms will be used in process terms where they may instantiate to (transmittable) messages via substitution of names in the variables that occur in them.

Definition 1 (Messages) Messages shall be names, tuples of messages, hashed messages, shared-key encrypted messages, public-key encrypted messages, or signed messages:

$$M, M' \quad ::= \quad n \mid (M, M') \mid \lceil M \rceil \mid [M]_k \mid \|M\|_{k^+} \mid |M|_{k^-}$$

Message forms shall be messages with variables or the so-called *abstract message* $\square$ (a meta-variable for messages).

Note that the formalistic approach leads us to (1) making abstraction from the exact representation of messages, e.g., bit strings; and assuming (2.1) *perfect hashing*, i.e., collision resistance (hash functions are injective) and strong pre-image resistance (given $\lceil M \rceil$, it is infeasible to compute M), and (2.2) *perfect encryption* (given $[M]_k$ or $\|M\|_{k^+}$ but neither k nor k^+, it is infeasible to compute M). Further, observe that our hashing operation is unkeyed and thus corresponds to a modification detection code (MDC). A keyed hashing operation, or message authentication code (MAC), can be emulated with an MDC and subsequent encryption.

Definition 2 (Message types) Message types shall be of the form described by Tables 1 and 2. Message type forms shall be message types with variables.

Observe that (1) for each kind of message there is a corresponding type (e.g., $\mathtt{H}[\sigma]$ for hashes, $\mathtt{SC}_k[\sigma]$ for symmetric and $\mathtt{AC}_{k^+}[\sigma]$ for asymmetric ciphers, $\mathtt{S}_{k^-}[\sigma]$ for signatures, and $\mathtt{T}[\sigma, \sigma']$ for tuples), (2) encryption and signature types are parametric, and (3) the union and difference of two message types is again a message type. In short, message types are structure-describing *dependent types* closed under union and

$$\begin{array}{rl}
\sigma, \sigma' & ::= \mathtt{P} \mid \varsigma \mid \mathtt{H}[\sigma] \\
& \mid \mathtt{SC}_k[\sigma] \mid \mathtt{AC}_{k^+}[\sigma] \mid \mathtt{S}_{k^-}[\sigma] \\
& \mid \mathtt{T}[\sigma, \sigma'] \mid \sigma \cup \sigma' \mid \sigma - \sigma' \mid \mathtt{S} \\
\varsigma, \varsigma' & ::= \mathtt{C} \mid \mathtt{C}^+ \mid \mathtt{C}^- \mid \mathtt{X}
\end{array}$$

Table 1: Pre-defined message types

$$\begin{array}{c}
\sigma \cap \sigma' := \sigma - (\sigma - \sigma') \\
\mathtt{C}^* := \mathtt{C}^{\mathtt{x}} \cup \mathtt{C}^+ \qquad \mathtt{C}^{\mathtt{x}} := \mathtt{C} \cup \mathtt{C}^- \\
\mathtt{N} := \mathtt{P} \cup \mathtt{C}^* \cup \mathtt{X} \cup \mathtt{S} \\
\mathtt{SC}[\mathtt{M}] := \bigcup_{k \in \mathcal{C}} \mathtt{SC}_k[\mathtt{M}] \\
\mathtt{AC}[\mathtt{M}] := \bigcup_{k^+ \in \mathcal{C}^+} \mathtt{AC}_{k^+}[\mathtt{M}] \\
\mathtt{S}[\mathtt{M}] := \bigcup_{k^- \in \mathcal{C}^-} \mathtt{S}_{k^-}[\mathtt{M}] \\
\mathtt{M} := \mathtt{N} \cup \mathtt{H}[\mathtt{M}] \cup \mathtt{T}[\mathtt{M}, \mathtt{M}] \cup \mathtt{SC}[\mathtt{M}] \cup \mathtt{AC}[\mathtt{M}] \cup \mathtt{S}[\mathtt{M}]
\end{array}$$

Table 2: Macro-defined message types

difference. We further define the (meta) message type S (for *s*ort) of message types, and macro-define the type of the intersection of two message types, the message type of names, resp. the message type of (general) messages.

2.2 Formulae

Definition 3 (Formulae) The set of formulae $\mathcal{F}_{\mathcal{M}}$ shall contain precisely those propositions that are closed (1) basic state predicates ψ, or compound (2.1) standard state predicates φ, (2.2) linear state predicates λ, or (2.3) path predicates ϕ. These predicates are formed with the operators of Table 3. Note that (1) a and b designate either a participant or a variable, (2) n designates either a key, a nonce, or a variable, (3) ι denotes the number of sessions during which a key is considered valid (with $\iota = 1$ for a session-key and $\iota = \infty$ for a long-term key), and (4) quantification is *typed*, where v may not occur in θ.

Note that predicates can be transformed into propositions either via binding of free variables, i.e., *generalisation* (universal quantification) or *abstraction* (existential quantification), or through substitution of constants into free variables, i.e., *individuation*.

$$\begin{array}{rcl}
\lambda, \lambda' & ::= & \varphi \mid \neg\lambda \mid \lambda \multimap \lambda' \mid \lambda \otimes \lambda' \mid \lambda \oplus \lambda' \\
\varphi, \varphi' & ::= & \psi \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \forall(v:\theta)(\varphi) \mid \mathsf{K}_a(\varphi) \mid \mathsf{P}(\varphi) \mid \triangle\phi \\
\psi, \psi' & ::= & \chi \mid F \preccurlyeq F' \mid F : \theta \mid \theta \sqsubseteq \theta' \\
\chi, \chi' & ::= & a\,\mathsf{g}_\iota\,n \mid a\,\mathsf{k}\,F \mid \mathsf{s}^!(a, F, b) \mid a\,\mathsf{r}^!\,F \\
\phi, \phi' & ::= & \varphi \mid \neg\phi \mid \phi \wedge \phi' \mid \forall(v:\theta)(\phi) \mid \mathsf{K}_a(\phi) \mid \mathsf{P}(\phi) \mid \phi\,\mathsf{B}\,\phi' \\
& \mid & {}^{\text{-}}\mathsf{X}(\phi) \mid \mathsf{X}(\phi) \mid \phi\,\mathsf{W}\,\phi'
\end{array}$$

Table 3: Pre-defined predicates

Our operators are pronounced as: $\neg$ "not", $\multimap$ "linearly implies", $\otimes$ "linear and", $\oplus$ "linear or", $\wedge$ "and", $\forall(v:\theta)$ "for all v of type θ", K_a "a knows that", P "it is permitted that", $\triangle$ "for all futures", g "generated", k "knows", $\mathsf{s}^!$[7] "precisely sent off to", $\mathsf{r}^!$ "precisely received", $\preccurlyeq$ "is a subterm of", : "has type", $\sqsubseteq$ "is a subtype of", B "back to", $\bar{}\mathsf{X}$ "previous", X "next", and W "waiting for".

Linear formulas describe properties of *multiple* (concurrent) protocols (cf. Table 5) whereas "[...] temporal formulas describe properties of [...] a single (concurrent) program." (Goldblatt in [13, Section 7.3]).

K expresses knowledge *de dicto*[8] (or *propositional* knowledge), i.e., *justified true belief*. To us, 'justified' shall mean *corroborated with adequate* cryptographic *evidence*, where the meaning of 'adequate' is determined by the degree of *skepticism* we decide to adopt toward that evidence. For the moment, that degree shall be bounded by the assumptions of *perfect cryptography*, i.e., perfect hashing plus perfect encryption plus the impossibility to guess cryptographic germs. Knowledge de dicto is established based on so-called bodies of *first-order knowledge*, which is why we also call it *higher-order knowledge*. It conveys the knowledge that certain statements about cryptographic communication are true. k expresses knowledge *de re*[9], which is established based on pieces of cryptographic information (first-order knowledge). It conveys *possession* and *understanding of the purpose* of a piece of cryptographic information *up to cryptographically irreducible parts*.

Further, we macro-define the predicates formed by the auxiliary operators of Table 4 by assuming the remaining operators of (1) propositional logic $\bot$, $\vee$, $\top$, $\rightarrow$, $\leftrightarrow$, and $\exists(v:\theta)$; and (2.1) branching-time logic $\bigtriangledown$ "there is a future s.t.", and (2.2) linear time logic $\boxminus$ "so far", $\boxplus$ "henceforth", $\diamondplus$ "eventually", U "until", $\diamondminus$ "once", and S "since" macro-defined on basic and compound standard state and compound path predicates (cf. Definition 3).

These auxiliary operators are pronounced as: k_θ "knows down to grain θ", $\mathsf{k}^!_\theta$ "knows precisely down to grain θ", $\mathsf{k}^!$ "totally knows", $\mathsf{k}^F_{\leq}$ "knows less or equally much w.r.t. F than", $\mathsf{sent}^!$ "is a sent message", s "sent off to", r "received", $:^!$ "has precisely the type", F "it is forbidden that", $\mathsf{key_s}$ "is a session key", $\mathsf{key_{lt}}$ "is a long-term key", $\mathsf{key}^!$ "is a corrupted (session or long-term) key", $\mathsf{key}_{\{a,b\}}$ "is a shared-key among a and b", and $\mathsf{key_o}$ "is an operational confidential key". (Observe the alternative definitions of $\mathsf{k}^!$ and $\mathsf{k}^F_{\leq}$.)

2.3 Some cryptographic goals and characterisations of adversaries and participants

As an illustration of how CPL can be used to express cryptographic properties, we propose in [17] formalisations of principal cryptographic goals, such as forms of secrecy

[7] the $^!$ mark can be viewed as a sort of *connotation operator*

[8] knowledge *from saying*

[9] knowledge *from the thing*, i.e., a piece of cryptographic information

$$
\begin{array}{rcl}
a\,\mathsf{g}\,n & := & a\,\mathsf{g}_1\,n \vee a\,\mathsf{g}_\infty\,n \\
a\,\mathsf{k}_\theta\,F & := & \mathsf{K}_a(F:\theta) \\
a\,\mathsf{k}^!_\theta\,F & := & a\,\mathsf{k}_\theta\,F \\
 & \wedge & \forall(v:\mathtt{S})(v \sqsubset \theta \rightarrow \neg\, a\,\mathsf{k}_\theta\,F) \\
a\,\mathsf{k}^!\,F & ::= & \exists(v:\mathtt{S})(\mathsf{K}_a(F :^! v)) \\
 & | & \forall(v:\mathtt{M}-\mathtt{C}^{\text{-}})(v \preccurlyeq F \rightarrow a\,\mathsf{k}\,v) \\
 & | & \forall(v:\mathtt{M}-\mathtt{C}^{\text{-}})(v \preccurlyeq F \rightarrow \mathsf{K}_a(v \preccurlyeq F)) \\
a\,\mathsf{k}^F_{\leq}\,b & ::= & \forall(v:\mathtt{S})(a\,\mathsf{k}_v\,F \rightarrow b\,\mathsf{k}_v\,F) \\
 & | & \forall(v:\mathtt{M}-\mathtt{C}^{\text{-}})(v \preccurlyeq F \rightarrow (a\,\mathsf{k}\,v \rightarrow b\,\mathsf{k}\,v)) \\
a\,\mathsf{k}^F_{=}\,b & := & a\,\mathsf{k}^F_{\leq}\,b \wedge b\,\mathsf{k}^F_{\leq}\,a \\
a\,\mathsf{k}^F_{\neq}\,b & := & \neg\, a\,\mathsf{k}^F_{=}\,b \\
a\,\mathsf{k}^F_{<}\,b & := & a\,\mathsf{k}^F_{\leq}\,b \wedge a\,\mathsf{k}^F_{\neq}\,b \\
\mathsf{sent}^!(F) & := & \exists(v,v':\mathtt{P})(\mathsf{s}^!(v,F,v')) \\
\mathsf{s}(a,F,b) & := & \exists(v:\mathtt{M})(F \preccurlyeq v \wedge \mathsf{s}^!(a,v,b)) \\
a\,\mathsf{r}\,F & := & \exists(v:\mathtt{M})(F \preccurlyeq v \wedge a\,\mathsf{r}^!\,v) \\
F = F' & := & F \preccurlyeq F' \wedge F' \preccurlyeq F
\end{array}
$$

$$
\begin{array}{rcl}
F \prec F' & := & F \preccurlyeq F' \wedge F \neq F' \\
F :^! \theta & := & \forall(v:\mathtt{S})((F:v \wedge v \sqsubseteq \theta) \rightarrow v = \theta) \\
\theta = \theta' & := & \theta \sqsubseteq \theta' \wedge \theta' \sqsubseteq \theta \\
\theta \sqsubset \theta' & := & \theta \sqsubseteq \theta' \wedge \theta' \neq \theta \\
\mathsf{K}(\varphi) & := & \forall(v:\mathtt{P})(\mathsf{K}_v(\varphi)) \\
\mathsf{K}(\phi) & := & \forall(v:\mathtt{P})(\mathsf{K}_v(\phi)) \\
\mathsf{F}(\varphi) & := & \neg\mathsf{P}(\varphi) \\
\mathsf{F}(\phi) & := & \neg\mathsf{P}(\phi) \\
{}^{\text{-}}\mathsf{X}^{i+1}(\phi) & := & {}^{\text{-}}\mathsf{X}^i({}^{\text{-}}\mathsf{X}(\phi)) \\
{}^{\text{-}}\mathsf{X}^0(\phi) & := & \phi \\
\mathsf{key_s}(k) & := & \exists(v:\mathtt{P})(v\,\mathsf{g}_1\,k) \\
\mathsf{key_{lt}}(k) & := & \exists(v:\mathtt{P})(v\,\mathsf{g}_\infty\,k) \\
\mathsf{key^!_s}(k) & := & \mathsf{key_s}(k) \wedge \mathtt{Eve}\,\mathsf{k}\,k \\
\mathsf{key^!_{lt}}(k) & := & \mathsf{key_{lt}}(k) \wedge \mathtt{Eve}\,\mathsf{k}\,k \\
\mathsf{key}_{\{a,b\}}(k) & := & \mathsf{K}_a(b\,\mathsf{g}\,k) \vee \mathsf{K}_b(a\,\mathsf{g}\,k) \\
\mathsf{key_o}(k,F) & := & \exists(v:\mathtt{M})([v]_k \preccurlyeq F \vee |v|_k \preccurlyeq F)
\end{array}
$$

Table 4: Some macro-defined predicates

(e.g., $\neg\triangledown\oplus(\mathtt{Eve}\,\mathsf{k}\,M)$) and forms of authenticity (e.g., $\mathsf{K}_a(\mathsf{s}^!(b,M,a))$), as well as characterisations of passive and active adversaries, and prudent[10] participants in terms of their capabilities to *attack*, resp. in terms of their competence to *secure* cryptographic communication. Note that (1) *authorisation* goals (expressing legitimacy concerns) can be expressed directly in CPL as the concept of authorisation is hardwired in it via the operator P, and (2) protocol-compliant participants are *non-repudiating* by definition. Our formalisations express secrecy and authenticity w.r.t. the *content*[11] as well as w.r.t. the *sender and recipient*[12] of a message, and to different degrees of *generality* (arbitrary or specific message) and *strength* (weak/strong variants).

[10] in the spirit of [3]

[11] message content authenticity is also called *message authentication*

[12] with respect to a sender or recipient, secrecy is also called *anonymity* (of the corresponding entity) and authenticity *entity authentication*

3 Process model

We model cryptographic protocols as processes (as opposed to functions) and protocol execution as non-deterministic process reduction defined by a *reduction calculus*. Processes are composable and participant-name-based asynchronously communicating distributed entities of concurrent non-recursive[13] threads allowing for dynamic name generation and secure and insecure input/output of cryptographic messages. Moreover, they are parametric in the language of *messages* they may transmit, and in the language of *conditionals* they may admit as guards on their execution. Process execution engenders (1) the activity of protocol participants and the (Dolev-Yao [10]) adversary (Eve), i.e., the generation of legitimate resp. illegitimate protocol events; and (2) the evolution (involving computation) of their respective knowledge de re and de dicto. As it proceeds, it produces a *history of protocol events* and a *sequence of knowledge states* recorded in suitable data structures and amenable to run-time model-checking.

The existence of primitives for insecure *and* secure input/output allows us to write specifications of cryptographic goals in the style of process algebra, i.e., by means of process equivalences. It is one of our goals to characterise these equivalences logically, i.e., in terms of CPL-formulae.

Data structures We need the following data structures, i.e., abstractions for storage: (1) a set for the body of first-order knowledge $\mathcal{K}_{\mathtt{Eve}}$ of the adversary (Eve); (2) for each participant b, a set for the body of their first-order private and public knowledge $\mathcal{K}_b^-$ resp. $\mathcal{K}_b^+$ (cf. [17, Table 11]), and session counters $\mathrm{s}_b(p, r)$ for each protocol p in which they may take part and role r they may take on; and (3) a list $\mathcal{H}$ for the history of protocol events.

Control structures Our control structures, i.e., abstractions for cryptographic communication, are: abstractions for (1) the (participant-centered) execution of threads, (2) the (protocol-centered) execution of processes, and (3) the evaluation of cryptographic messages. Our abstraction for message evaluation is an algebra of cryptographic messages with a certain set of external[14] operations:

Definition 4 (Message algebra) Let $\mathcal{O}$ denote a set of operations on messages $M \in \mathcal{M}$ with (1) fst, snd, decr, decr^+, and vsign inverse operations for tuples, symmetric and asymmetric ciphers, resp. signed messages; (3) $\mathcal{C}_i$ composition operations on message operations; and (2) $\mathcal{P}_i$ projection operations on message operation parameters.

Then $\mathfrak{M}$ shall denote our algebra of crypto messages:

$$\mathfrak{M} := \langle \mathcal{M}, \mathcal{O} \rangle$$

[13] cryptographic protocols do not have loops

[14] as opposed to internal operations that are equationally *specified*, as for example in the so-called Applied Pi-Calculus [2]

Our abstractions for protocol execution are process and thread constructors together with an associated notion of execution defined in terms of the reduction calculus C^3 (*Calculus of Cryptographic Communication*):

Definition 5 (Process reduction calculus) Let

- $\boldsymbol{P}, \boldsymbol{Q} \in \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}}$ denote *processes* (cf. [17]) that transmit messages $M \in \mathcal{M}$ and admit conditionals $\lambda \in \mathcal{F}_\mathcal{M}$ as guards on their execution
- $\dot{\longrightarrow}_{\cdot\models} \subseteq (\mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}}) \times \boldsymbol{\mathcal{E}} \times (\mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}})$ denote a reduction relation (cf. [17]). $\boldsymbol{\mathcal{S}}$ denotes the set of *memory states* (or data structure states) $\mathfrak{m}$, and $\boldsymbol{\mathcal{E}}$ the set of *protocol events*.

Then $\mathfrak{C}^3$ shall denote a calculus of cryptographic communication instantiated with a crypto system combining shared-key and public-key cryptography with hashing:

$$\mathfrak{C}^3 := \langle \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}}, \dot{\longrightarrow}_{\cdot\models} \rangle$$

4 Satisfaction

Our definition of satisfaction is *anchored* (or *rooted*) and defined in terms of models of cryptographic protocols (process terms) and corresponding global memories (data structures). It is novel in the sense that it proceeds via *nested induction* across three layers of (logical) syntax. Nesting is resolved through *exhaustion of operator combinations*.

Due to this combinatorics of operators, our definition may appear unnecessarily complicated at first sight. However given the intrinsic complexity of cryptographic reality, we claim that our definition is only just *necessarily complex* and even *particularly insightful*. Our definition is insightful in the sense that it extricates the *structure* of cryptographic discourse by revealing/inducing it in combinations of what we believe to be its *elementary parts*, i.e., the operators of our logic, and in the *algebraic laws* that govern it.

These laws are induced in the sense that they hold by definition of the satisfaction relation[15]; they are algebraic in the sense that they relate structures, i.e., combinations of operators, that are logically equivalent.

4.1 Definition

We define satisfaction inductively and in a relational (as opposed to functional) style on the structure of formulae.

[15] thus they are automatically sound axioms in any proof system for — or *deduction relation* on — CPL formulae

Definition 6 (Satisfaction) Let $\models_{\mathfrak{C}^3} \subseteq \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \mathcal{F}_\mathcal{M}$ denote satisfaction of a formula $\lambda \in \mathcal{F}_\mathcal{M}$ in a process $\boldsymbol{P} \in \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}}$, the anchor/root of a an implicit (execution tree) model for λ. Then satisfaction shall be defined as:

$$\boldsymbol{P} \models_{\mathfrak{C}^3} \lambda \quad \text{:iff} \quad (\boldsymbol{P}, [\![\boldsymbol{P}]\!]) \cdot\vDash_{\mathfrak{C}^3} \lambda$$

where $[\![\cdot]\!] : \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \to \boldsymbol{\mathcal{S}}$ maps a process term to its initial (memory) state, and $\cdot\vDash_{\mathfrak{C}^3}$ (cf. Tables 5 and 6) defines satisfaction on protocol-memory pairs referring to $\vDash_{\mathfrak{C}^3}$ denoting satisfaction on paths of such pairs (cf. Table 8).

$$(\boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \neg\lambda \quad \text{:iff} \quad \text{not } (\boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda$$

$$(\boldsymbol{P} \circ \boldsymbol{Q}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \multimap \lambda' \quad \text{:iff}$$
$$\text{if } (\boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \text{ and } (\boldsymbol{Q}, \mathfrak{m}') \cdot\vDash_{\mathfrak{C}^3} \lambda \text{ then } (\boldsymbol{Q}, \mathfrak{m}') \cdot\vDash_{\mathfrak{C}^3} \lambda',$$
$$\text{for all } \mathfrak{m}' \text{ s.t. } (\boldsymbol{P}, \mathfrak{m}) \longrightarrow^* (\texttt{SKIP}, \mathfrak{m}')$$

$$(\boldsymbol{P} \times \boldsymbol{Q}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \otimes \lambda' \quad \text{:iff}$$
$$(\boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \text{ and } (\boldsymbol{Q}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda'$$

$$(\boldsymbol{P} + \boldsymbol{Q}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \oplus \lambda' \quad \text{:iff}$$
$$(\boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda \text{ or } (\boldsymbol{Q}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda'$$

$$(\boldsymbol{Q} \blacktriangleleft \lambda \triangleright \boldsymbol{P}, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \lambda' \quad \text{:iff}$$
$$((\boldsymbol{P} \times \boldsymbol{P}) + (\boldsymbol{P} \times \boldsymbol{Q}), \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} (\lambda \otimes \lambda') \oplus (\neg\lambda \otimes \lambda')$$

Table 5: Linear state satisfaction

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \psi \quad \text{:iff} \quad (p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \in \mathrm{V}(\psi)$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \neg\varphi \quad \text{:iff} \quad \text{not } (p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \varphi$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \varphi \wedge \varphi' \quad \text{:iff}$$
$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \varphi \text{ and } (p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \varphi'$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \forall(v : \sigma)(\varphi) \quad \text{:iff}$$
$$\text{for all } c \in [\![\sigma]\!], (p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \boxed{c/v \cdot \varphi}$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \mathsf{K}_c(\varphi) \quad \text{:iff} \quad \mathrm{K}_\mathrm{s}(c, \varphi)$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \mathsf{P}(\varphi) \quad \text{:iff} \quad \mathrm{P}_\mathrm{s}(\varphi)$$

$$(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \cdot\vDash_{\mathfrak{C}^3} \triangle\phi \quad \text{:iff}$$
$$\text{for all } \mathfrak{p} \in \mathrm{paths}((p\langle P\rangle{:}\{C\}_l, \mathfrak{m})), \mathfrak{p}@0 \vDash_{\mathfrak{C}^3} \phi$$

Table 6: Modal state satisfaction

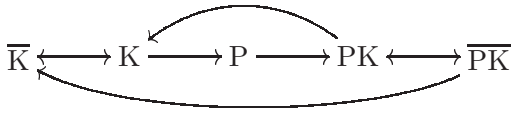
In Table 5, (1) note that $\circ$, $\times$, $+$, $\blacktriangleleft\ \lambda\ \triangleright$ denote serial, parallel, alternative, resp. conditional composition of protocols; and (2) observe how the interpretation of the

Table 7: Defining relations for K and P for satisfaction on states (s) and paths (p)

conclusion λ' in the linear implication $\lambda \multimap \lambda'$ takes place only after the left protocol in the sequential composition has terminated, or — in the jargon of linear logics — has been consumed as a resource (for execution).

In Table 6, observe that satisfaction of basic formulae ψ is defined in terms of a *valuation function* V (cf. [17]), and satisfaction of epistemic and deontic formulae is defined in terms of the relations $\mathrm{K_s}$ resp. $\mathrm{P_s}$ (cf. Table 7). Note that $\mathrm{paths}((p\langle P\rangle{:}\{C\}_l, \mathfrak{m}))$ is defined as the set of paths $\mathfrak{p}$ such that the first state of $\mathfrak{p}$ is $(p\langle P\rangle{:}\{C\}_l, \mathfrak{m})$ and the last state of $\mathfrak{p}$ is $(p\langle P'\rangle{:}\{C\}_l, \mathfrak{m}')$ where $(p\langle P\rangle{:}\{C\}_l, \mathfrak{m}) \longrightarrow^* (p\langle P'\rangle{:}\{C\}_l, \mathfrak{m}') \not\longrightarrow$.

Table 7 shows the combinations of what we believe to be the elementary parts of cryptographic discourse about security of communication and the laws that govern it. The table represents our main contribution in this paper. Observe the nested induction across three layers of syntax in the definition of the operators K and P. The definition is structured on states as well as on paths as follows:



Of course, this definition needs to be revised and refined so that consensus about the meaning of those elementary parts can be established in the community of protocol designers and analysts. We hope that once this consensus has been established, the definition will constitute a *standard* for the body of common knowledge of that community.

In $\mathrm{K_s}$ observe or note that

1. $\mathrm{synth}_{\langle \mathcal{M}, \equiv \rangle}$ (cf. [17]) denotes a function of message synthesis. It is an enhanced version of Paulson's [21] in the sense that it is refinable via an equivalence $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ defined by the set of *algebraic laws*[16] that may hold between the cryptographic primitives in $\mathcal{M}$. Further, $\preccurlyeq$ (cf. [17]) denotes a sub-term relation, and $\vdash$ (cf. [17]) denotes a type-inference relation.

2. satisfaction of the formulae $\mathsf{K}_c(\mathsf{s}^!(a, M, b))$ (capturing authenticity of datum M w.r.t. its origin) and $\mathsf{K}_c(a\,\mathsf{r}^!\,M)$ (capturing acknowledged reception of datum M) is sound, i.e., conveys the desired meaning, iff the operational confidential key v has been — and will remain — secret. Otherwise c's knowledge actually is — resp. will eventually be — just true or false *belief*, depending on the absence resp. presence of active impersonation. Thus, violation of the secrecy of operational confidential keys entails *non-monotonicity* of knowledge, and its observance is a necessary condition (or *healthiness constraint* of our definition of satisfaction) for any goal relying on those keys. An open question is whether there is a class of goals which it is (also) a sufficient condition for.

[16] such as the so-called *homomorphic property* of RSA, which would be expressed as $|(M, M')|_{k^-} \equiv (|M|_{k^-}, |M'|_{k^-})$

3. $\boxdot$ is a syntactic expansion function of standard state and path formulae, e.g., $\mathsf{K}_{c'}(c\,\mathsf{k}\,(x,x') \wedge c\,\mathsf{k}\,k) \rightsquigarrow \mathsf{K}_{c'}(c\,\mathsf{k}\,(x,x')) \wedge \mathsf{K}_{c'}(c\,\mathsf{k}\,k) \rightsquigarrow \mathsf{K}_{c'}(c\,\mathsf{k}\,x \wedge c\,\mathsf{k}\,x') \wedge \mathsf{K}_{c'}(c\,\mathsf{r}\,k \vee \mathsf{s}(c,k,c')) \rightsquigarrow \ldots$ represents the beginning of the expansion chain of $\boxed{\mathsf{K}_{c'}(c\,\mathsf{k}\,(x,x') \wedge c\,\mathsf{k}\,k)}$. Observe that such an expansion results in the elimination of the operator for propositional knowledge or in a formula containing occurrences of the formula $\mathsf{K}_c(\varphi)$ for $\varphi ::= \mathsf{K}_{c'}(\mathtt{Eve}\,\mathsf{k}\,M) \mid \mathsf{K}_{\mathtt{Eve}}(\mathsf{s}^!(a,M,b)) \mid \mathsf{K}_{\mathtt{Eve}}(a\,\mathsf{r}^!\,M) \mid \mathsf{K}_{c'}(M \preccurlyeq M') \mid \mathsf{K}_{c'}(M:\sigma)$. Satisfaction of such a formula is defined in terms of those pieces of c''s first-order knowledge that c must know to be able to know that φ holds. We omit further details. However, it is now clear that K is *not truth-functional*, i.e., it is not defined exclusively in terms of the logical *form*, but rather also in terms of the information *content* of the proposition it operates on. In this respect, K resembles *strict*[17] (as opposed to *material*) implication, which is not truth-functional either.

4. (branching) future of protocol execution is epistemologically inaccessible to participants because they are not in control of that execution in reality (although they may partially be in the model, e.g., in our model a participant may always generate a new germ or send off some message to some participant).

In $\mathrm{K_p}$ observe or note that

1. linear future is epistemologically inaccessible to participants for the same reason as is branching future.

2. the function δ calculates the difference of back (X) and forth (X) steps in a possible prefix of such steps in $\mathsf{X}(\phi)$ The function $\rfloor$ effectively chops that prefix off.

In Table 8, note that $\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi$ is pronounced "path $\mathfrak{p}$ satisfies the formula ϕ (in the state) at position i", and that $\mathrm{state}(\mathfrak{p}, i,)$ returns the state at position i in $\mathfrak{p}$.

This concludes our definition of satisfaction.

4.2 Discussion

In the terminology of modal logics,

1. our calculus of cryptographic communication $\mathfrak{C}^3 := \langle \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}}, \dot{\longrightarrow}_{\cdot\vDash} \rangle$ with *universe* $\mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}}$, *possible worlds* $\mathfrak{w} \in \mathcal{P}_{\mathfrak{M},\mathcal{F}_\mathcal{M}} \times \boldsymbol{\mathcal{S}}$, and (temporal) *accessibility relation* $\dot{\longrightarrow}_{\cdot\vDash}$ constitutes a *frame* and

2. the structure $\mathfrak{M}_{\mathrm{CPL}} := \langle \mathfrak{C}^3, \mathrm{V} \rangle$ with frame $\mathfrak{C}^3$ and valuation V constitutes a *model* for the language $\mathcal{F}_\mathcal{M}$.

[17] Strict implication requires that the antecedent imply the consequent in accordance with some notion of *necessity*, e.g., inclusion of the information content of the consequent in the information content of the antecedent.

$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \varphi \quad$:iff $\quad \mathrm{state}(\mathfrak{p}, i) \cdot\vDash_{\mathfrak{C}^3} \varphi$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \neg\phi \quad$:iff $\quad$ not $\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi \wedge \phi' \quad$:iff $\quad \mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi$ and $\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi'$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \forall(v:\sigma)(\phi) \quad$:iff $\quad$ for all $c \in [\![\sigma]\!]$, $\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \boxed{c/v \cdot \phi}$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \mathsf{K}_a(\phi) \quad$:iff $\quad \mathrm{K}_{\mathrm{p}}(a, \phi)$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \mathsf{P}(\phi) \quad$:iff $\quad \mathrm{P}_{\mathrm{p}}(\phi)$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi \,\mathsf{B}\, \phi' \quad$:iff $\quad$ there is a k s.t. $0 \leq k \leq i$ and $\mathfrak{p}@k \vDash_{\mathfrak{C}^3} \phi'$, and for all j, if $k < j \leq i$ then $\mathfrak{p}@j \vDash_{\mathfrak{C}^3} \phi$; or for all k, if $0 \leq k \leq i$ then $\mathfrak{p}@k \vDash_{\mathfrak{C}^3} \phi$.
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \overset{\cdot}{\mathsf{X}}(\phi) \quad$:iff $\quad \begin{cases} \mathfrak{p}@(i-1) \vDash_{\mathfrak{C}^3} \phi & \text{if } i-1 \geq 0 \\ \text{true} & \text{otherwise} \end{cases}$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \mathsf{X}(\phi) \quad$:iff $\quad \begin{cases} \mathfrak{p}@(i+1) \vDash_{\mathfrak{C}^3} \phi & \text{if } i+1 \leq \mathrm{len}(\mathfrak{p}) \\ \text{false} & \text{otherwise} \end{cases}$
$\mathfrak{p}@i \vDash_{\mathfrak{C}^3} \phi \,\mathsf{W}\, \phi' \quad$:iff $\quad$ there is a k s.t. $i \leq k$ and $\mathfrak{p}@k \vDash_{\mathfrak{C}^3} \phi'$, and for all j, if $i \leq j < k$ then $\mathfrak{p}@j \vDash_{\mathfrak{C}^3} \phi$; or for all k, if $i \leq k$ then $\mathfrak{p}@k \vDash_{\mathfrak{C}^3} \phi$.

Table 8: Path satisfaction

3. λ is a *local truth* in $\mathfrak{M}_{\mathrm{CPL}}$ at $\mathfrak{w}$:iff $(\mathfrak{M}_{\mathrm{CPL}}, \mathfrak{w}) \cdot\vDash_{\mathfrak{C}^3} \lambda$.

4. λ is a *semantic consequence* of a set Λ of formulae in $\mathfrak{M}_{\mathrm{CPL}}$, written $\Lambda \Rightarrow_{\mathfrak{C}^3} \lambda$, :iff for all $\mathfrak{w} \in \mathcal{P}_{\mathfrak{M},\mathcal{F}_{\mathcal{M}}} \times \boldsymbol{\mathcal{S}}$, if $(\mathfrak{M}_{\mathrm{CPL}}, \mathfrak{w}) \cdot\vDash_{\mathfrak{C}^3} \bigwedge_{\lambda_i \in \Lambda} \lambda_i$ then $(\mathfrak{M}_{\mathrm{CPL}}, \mathfrak{w}) \cdot\vDash_{\mathfrak{C}^3} \lambda$.

5. λ is a *global truth* in $\mathfrak{M}_{\mathrm{CPL}}$, written $\mathfrak{M}_{\mathrm{CPL}} \cdot\vDash_{\mathfrak{C}^3} \lambda$, :iff for all $\mathfrak{w} \in \mathcal{P}_{\mathfrak{M},\mathcal{F}_{\mathcal{M}}} \times \boldsymbol{\mathcal{S}}$, $(\mathfrak{M}_{\mathrm{CPL}}, \mathfrak{w}) \cdot\vDash_{\mathfrak{C}^3} \lambda$.

6. λ is a *tautology* of CPL, written $\cdot\vDash_{\mathfrak{C}^3} \lambda$, :iff for all $\mathfrak{M}_{\mathrm{CPL}}$, $\mathfrak{M}_{\mathrm{CPL}} \cdot\vDash_{\mathfrak{C}^3} \lambda$.

7. λ is an *antinomy* of CPL, written $\cdot\nvDash_{\mathfrak{C}^3} \lambda$, :iff $\neg\lambda$ is a tautology of CPL.

8. $\mathcal{CPL}_{\mathfrak{C}^3} := \{\, \lambda \mid \cdot\vDash_{\mathfrak{C}^3} \lambda \,\}$

Observe that the frame $\mathfrak{C}^3$ neither mentions an epistemic nor a deontic accessibility relation, which is non-standard for a frame of a logic with standard epistemic and deontic modalities. However, $\mathcal{CPL}_{\mathfrak{C}^3}$ contains tautologies that characterise such a frame (cf. next section). It follows that the temporal accessibility relation actually *induces* both an epistemic and a deontic accessibility relation. We leave it for further work to recover these implicit accessibility relations from the temporal one, and to give an alternative presentation of $\mathcal{CPL}_{\mathfrak{C}^3}$ as a *product* (rather than as a *fusion*, as there is intrinsic interaction between our modalities) of the corresponding single-dimensional logics.

4.3 Characteristic tautologies and antinomies

By definition, the set of all tautologies of a language constitutes the logic of that language. It is a standard meta-theoretical task to try to isolate a subset S of that set that characterises the logic in the sense that every tautology of the logic is a semantic consequence of S. One says that S *generates* the logic via semantic consequence. In effect, $\mathcal{CPL}_{\mathfrak{C}^3}$ contains at least the epistemic system **S5**, i.e., LO, ES, PI, and NI, and the deontic system **DAlt**, i.e., DK, DCY, and CL (cf. Table 9). This means that our implicit accessibility relations correspond in fact to an epistemic *equivalence*, and a deontic *functional* and *serial*[18] relation.

$\cdot\models_{\mathfrak{C}^3} \mathsf{K}_a(\varphi \to \varphi') \to (\mathsf{K}_a(\varphi) \to \mathsf{K}_a(\varphi'))$	LO
$\cdot\models_{\mathfrak{C}^3} \mathsf{K}_a(\varphi) \to \varphi$	ES
$\cdot\models_{\mathfrak{C}^3} \mathsf{K}_a(\varphi) \to \mathsf{K}_a(\mathsf{K}_a(\varphi))$	PI
$\cdot\models_{\mathfrak{C}^3} \neg\mathsf{K}_a(\varphi) \to \mathsf{K}_a(\neg\mathsf{K}_a(\varphi))$	NI
$\cdot\not\models_{\mathfrak{C}^3} \varphi \to \mathsf{K}_a(\varphi)$	GEI
$\cdot\models_{\mathfrak{C}^3} \chi \to \mathsf{K}_a(\chi)$	LEC
$\cdot\models_{\mathfrak{C}^3} \neg\chi \to \mathsf{K}_a(\neg\chi)$	LEC
$\cdot\not\models_{\mathfrak{C}^3} \neg\mathsf{K}_a(\varphi) \to \mathsf{K}_a(\neg\varphi)$	ECM
$\cdot\models_{\mathfrak{C}^3} \mathsf{K}_a(\neg\varphi) \to \mathsf{K}_a(\varphi)$	ECY
$\cdot\models_{\mathfrak{C}^3} (\mathsf{K}_a(\varphi) \wedge \mathsf{K}_a(\varphi \to \varphi')) \to \mathsf{K}_a(\varphi')$	PR
$\cdot\models_{\mathfrak{C}^3} \mathsf{K}_a(\varphi) \to \triangle\boxplus\mathsf{K}_a(\varphi)$	PM
$\cdot\models_{\mathfrak{C}^3} \mathsf{O}(\varphi \to \varphi') \to (\mathsf{O}(\varphi) \to \mathsf{O}(\varphi'))$	DK
$\cdot\models_{\mathfrak{C}^3} \mathsf{O}(\varphi) \to \mathsf{P}(\varphi)$	DCY
$\cdot\models_{\mathfrak{C}^3} \mathsf{P}(\varphi) \to \mathsf{O}(\varphi)$	CL
$\cdot\models_{\mathfrak{C}^3} \mathsf{P}(\varphi) \to \mathsf{K}_a(\mathsf{P}(\varphi))$	PLEK
$\cdot\models_{\mathfrak{C}^3} \mathsf{F}(\varphi) \to \mathsf{K}_a(\mathsf{F}(\varphi))$	PLEK
$\cdot\models_{\mathfrak{C}^3} \mathsf{P}(\varphi) \to \triangle\boxplus\mathsf{P}(\varphi)$	SL
$\cdot\models_{\mathfrak{C}^3} \mathsf{F}(\varphi) \to \triangle\boxplus\mathsf{F}(\varphi)$	SL

Table 9: Some tautologies and antinomies

The abbreviations in Table 9 are pronounced as: LO "Logical Omniscience", ES "Epistemic Soundness", PI "Positive Introspection", NI "Negative Introspection", GEI "Global Epistemic Incompleteness", LEC "Local Epistemic Completeness", ECM "Epistemic Constructivism", ECY "Epistemic Consistency", PR "Perfect Reasoning", PM "Perfect Memory", DCY "Deontic Consistency" (note that $\mathsf{O}(\varphi) := \mathsf{F}(\neg\varphi)$ and

[18] a binary relation $\mathcal{R}$ is *serial* on a set S :iff for all $s \in S$, there is $t \in S$ s.t. $s\mathcal{R}t$.

is pronounced "it is obligatory that"), CL "Constringent Law", PLEK "Perfect Legal Knowledge", and SL "Static Law".

$\mathcal{CPL}_{\mathfrak{C}^3}$ also satisfies the following closure laws (PLOK "Perfect Logical Knowledge", LL "Logic is Law"):

$$\text{if } \varphi \in \mathcal{CPL}_{\mathfrak{C}^3} \text{ then } \mathsf{K}_a(\varphi) \in \mathcal{CPL}_{\mathfrak{C}^3} \qquad \text{PLOK}$$

$$\text{if } \varphi \in \mathcal{CPL}_{\mathfrak{C}^3} \text{ then } \mathsf{O}(\varphi) \in \mathcal{CPL}_{\mathfrak{C}^3} \qquad \text{LL}$$

A note on common knowledge: it is commonly defined as the largest fixpoint of propositional knowledge of a community of entities. Unfolding it leads to an infinitely deeply nested chain of operators K, e.g., $\ldots \mathsf{K}_b(\mathsf{K}_a(\mathsf{K}_b(\mathsf{K}_a(\varphi))))$ for the common knowledge of φ among a community of two entities a and b. But asserting that b knows that a knows that φ requires that b received at least one message from a providing evidence to b that a knows that φ (cf. Table 7). Hence, asserting that there is common knowledge in a community of already two entities requires the transmission of an infinite number of messages. This proves informally — but constructively — that establishing common knowledge across an insecure transmission medium is infeasible. In particular, public knowledge is not common knowledge !

4.4 Comparing cryptographic goals

In Section 4.2, CPL was presented as a *body of truth*, namely $\mathcal{CPL}_{\mathfrak{C}^3}$, based on the key concept of *validity*. Alternatively, it can be presented as a *system of inference* $\mathfrak{CPL}_{\mathfrak{C}^3} := \langle \mathcal{F}_{\mathcal{M}}, \Rightarrow_{\mathfrak{C}^3} \rangle$ based on the key concept of semantic *consequence*. We suggest the latter, which is a partial order relation, and its derivative, semantic *equivalence* $\Leftrightarrow_{\mathfrak{C}^3} := \{ (\lambda, \lambda') \mid \lambda \Rightarrow_{\mathfrak{C}^3} \lambda' \text{ and } \lambda' \Rightarrow_{\mathfrak{C}^3} \lambda \}$ as the basic means of comparing cryptographic goals. We leave it for further work to compare the goals stated in [17].

5 Conclusions

This paper is about the mathematical and philosophical foundations of the security of communication and its crystallisation into a standard logical theory.

Technical claims Our technical claims are to present (1) the first formalisation of cryptographic discourse about the security of communication within the framework of multi-dimensional logic, (2) the most comprehensive formal, logically connected model of cryptographic protocols proposed so far, and (3) a novel technique for defining evidence-based satisfaction on that model. As a result, we (1) are able to express and compare cryptographic properties intuitively and succinctly, (2) are confident to be able to verify correctness of cryptographic goals on target protocols down to a fine

grain of detail (cf. [17]), and (3) provide new insight into the subtleties of interaction between quantifiers and modal operators (pointed out by Gabbay et al. in [12, Page 143]), e.g., $\mathsf{K}_a(\forall(v:\sigma)(\varphi)) \Leftrightarrow_{\mathfrak{C}^3} \forall(v:\sigma)(a\,\mathsf{k}\,v \wedge \mathsf{K}_a(\varphi))$.

Philosophical claims Our philosophical claims are to present new insight in the *concerns*, the *dimensions*, and the *structure* of the universe of cryptographic discourse (cf. Table 10). As a result, we obtain a rigourous clarification of the concepts constituting common knowledge of the community of protocol designers and analysts.

In Table 10, by modulation of the interpretation structure we mean the variation of the compositionality of the process term $\boldsymbol{P}$ in a world $(\boldsymbol{P}, \mathfrak{m})$. By modulation of the scope of the truth statement, we mean the variation of the truth value of a formula φ or ϕ due to the formula occurring in the scope of certain operators. By modulation of the grain of the truth statement, we mean the variation of the degree (or probability) of certitude about the truth value of a formula due to cryptographic evidence justifying that value but having been generated by cryptographic functions that are breakable under certain complexity-theoretic assumptions.

<table>
<tr><th colspan="2">Model of reality</th><th colspan="2">Universe of discourse</th></tr>
<tr><th>Modelling concept</th><th>Modelling concern</th><th>Linguistic dimension</th><th>Semantic efffect</th></tr>
<tr><td>Cryptographic process</td><td>Compositionality</td><td>Execution Space</td><td>Modulation of the interpretation structure</td></tr>
<tr><td rowspan="3">Event</td><td>Sequentiality Concurrency</td><td>Time</td><td rowspan="4">Modulation of the scope of the truth statement</td></tr>
<tr><td>Actuality</td><td>Knowledge</td></tr>
<tr><td>Legitimacy</td><td>Norms</td></tr>
<tr><td>Message</td><td>Existence Universality</td><td>Quantification</td></tr>
<tr><td>Cryptographic function</td><td>Complexity</td><td>Probability</td><td>Modulation of the grain of the truth statement</td></tr>
</table>

Table 10: Security of communication

Further work We envisage work in the development of the following applied, theoretical, and meta-theoretical aspects of our logic: in the applied field, we are working on a fully-fledged *engineering methodology* and on representative specification and verification *case studies*. In the theoretical domain, it is conceivable to extend CPL with (1) weak real time to allow for the use of *time stamps* as cryptographic evidence,

(2) numeric keys to allow for *key derivation*, and (3) *probability* for propositional knowledge and *complexity* for cryptographic functions (in the spirit of [18]) to account for imperfect cryptography. In the meta-theoretical domain, the framework of standard mathematical logic provides us with a road map for work in (1) *model theory*, e.g., the logical characterisation of relations (such as bisimulations) on process models, (2) *alternative semantics* for our relational/possible-worlds semantics, e.g., axiomatic (proof system or relation of deduction) and game-theoretic (Ehrenfeucht-Fraïssé games) semantics, (3) the study of *decidability* and *complexity* issues, (4) *correspondence theory*, i.e., the isolation of a cryptographic fragment of first-order logic via the so-called standard translation, and (5) *intuitionism* and *process types* (Curry-Howard isomorphism).

References

[1] M. Abadi. Security protocols and their properties. In *Foundations of Secure Computation*, volume 175 of *NATO Science Series: Computer & Systems Sciences*. IOS Press, 2000.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2001.

[3] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[4] P. Bieber. A logic of communication in hostile environment. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1990.

[5] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[6] M. Bozzano and G. Delzanno. Automated protocol verification in linear logic. In *Proceedings of the ACM SIGPLAN international conference on Principles and practice of declarative programming*, 2002.

[7] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), 1990.

[8] E. Clarke, S. Jha, and W. Marrero. A machine checkable logic of knowledge for specifying security properties of electronic commerce protocols. In *Proceedings of the LICS-Affiliated Workshop on Model Checking & Security Protocols*, 1998.

[9] E. Cohen. First-order verification of cryptographic protocols. *Journal of Computer Security*, 11(2), 2003.

[10] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12), 1983.

[11] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4), 2003.

[12] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications*, volume 148 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2003.

[13] R. Goldblatt. Mathematical modal logic: A view of its evolution. *Journal of Applied Logic*, 1(5-6), 2003.

[14] J. W. Gray and J. D. McLean. Using Temporal Logic to Specify and Verify Cryptographic Protocols (progress report). In *Proceedings of the IEEE Computer Security Foundations Workshop*, 1995.
[15] J. Halpern and K. O'Neill. Secrecy in multi-agent systems. In *Proceedings of the IEEE Computer Security Foundations Workshop*, 2002.
[16] S. Kramer. A language and a notion of truth for cryptographic properties. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, 2003. Short Presentation.
[17] S. Kramer. CPL: An Evidence-Based 5-Dimensional Logic for the Compositional Specification and Verification of Cryptographic Protocols. Part I: Language, Process Model, Satisfaction. Technical Report IC/2004/14, École Polytechnique Fédérale de Lausanne, Switzerland, 2004.
[18] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the ACM Conference on Computer and Communication Security*, 1998.
[19] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Series on Discrete Mathematics and its Applications. CRC Press, 1996.
[20] J.-J. C. Meyer and R. J. Wieringa, editors. *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.
[21] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1), 1998.
[22] P. Selinger. Models for an adversary-centric protocol logic. In *Proceedings of the Workshop on Logical Aspects of Cryptographic Protocol Verification*, 2001.
[23] P. F. Syverson and P. C. van Oorschot. A unified cryptographic protocol logic. NRL CHAC 5540-227, Naval Research Laboratory, Center for High Assurance Computer Systems, Washington D.C., USA, 1996.

A Note about proofs

Our proofs of the mentioned tautologies, antinomies, and closure laws are essentially based on the following lemmata: (1) $\varphi \Rightarrow_{\mathfrak{C}^3} \varphi'$ iff $\cdot \vDash_{\mathfrak{C}^3} \varphi \rightarrow \varphi'$, (2) if $\overline{\mathrm{K}}$ then not K (proving ECY *by inspection*!), and (3) first-order knowledge increases monotonically (by inspection of process reduction). We are planning to present detailed proofs validated by a theorem-prover.

Non-monotonic Properties for Proving Correctness in a Framework of Compositional Logic ⋆

Koji Hasebe Mitsuhiro Okada

Department of Philosophy, Keio University
2-15-45 Mita, Minato-ku, Tokyo 108-8345, Japan
{hasebe,mitsu}@abelard.flet.keio.ac.jp

Abstract

Following up our previous work [9], we distinguish the monotonic properties and the non-monotonic ones in our inference system based on the framework of compositional logic, and give the way to include some non-monotonic properties. As an example, we present a correctness proof of Challenge Response protocol, and explain how such properties can be used in more powerful derivations. We also give a semantics based on the notion of trace, and present a soundness proof of our inference system including non-monotonic properties.

1 Introduction

Compositional logic (originally introduced by Durgin-Mitchell-Pavlovic [6] and Datta-Derek-Mitchell-Pavlovic [3, 4]) is an inference system based on Floyd-Hoare style logical framework for proving protocol correctness. By means of this framework, a protocol is considered as a program, and a statement "from a principal P's viewpoint, a general property φ holds at the end of his/her protocol action α" can be represented as a formula of the form $[\alpha]\varphi$ (or of the form $\theta[\alpha]\varphi$ in [3, 4]). One of the most advantageous points of this framework is its compositional approach for reasoning about a compound protocol: in order to prove a property about a compound protocol we can reuse already established properties about its components.

In our previous work [9], we presented a way for making more explicit the compositionality property of this framework by introducing a notion of *primitive actions in*

⋆This work was partly supported by Grants-in-Aid for Scientific Research of MEXT, Center of Excellence of MEXT on Humanity Sciences (Keio University), the Japan-US collaborative research program of JSPS-NSF, Oogata-kenkyuu-jyosei grant (Keio University) and Global Security Center grant (Keio University). The first author was also supported by Fellowship for Japanese Young Scientists from Japan Society for the Promotion of Science.

a role (i.e., sending, receiving or generating actions). While in [6, 3, 4] an assumption about a principal's honesty is represented as implication of the form $Honest(Q) \supset \varphi$ (which means "if principal Q is honest, then φ holds"), in our framework such assumption (called *honesty assumption*) is represented by the predicates of the form $Honest(\vec{\alpha}^Q)$ (where $\vec{\alpha}^Q$ is a sequence of primitive actions in a role performed by Q) in the left hand side of a sequent style assertion. In a proving process of a property, these honesty assumptions are composed by combining the usual contraction rule of the sequent calculus and the following *weakening rule* (analogous to the weakening rule of the traditional logic).

$$\frac{Honest_Q(\vec{\alpha};\vec{\alpha}'), \Gamma \vdash [\vec{\beta};\vec{\beta}']\varphi}{Honest_Q(\vec{\alpha};\alpha'';\vec{\alpha}'), \Gamma \vdash [\vec{\beta};\beta'';\vec{\beta}']\varphi}\ \textbf{Weakening}$$

(This means that, "from P's view, if a property φ is derived from Γ with Q's honesty on $\vec{\alpha};\vec{\alpha}'$ after P's performance of the sequence of actions $\vec{\beta};\vec{\beta}'$, then φ is also derived from Γ with Q's honesty on $\vec{\alpha};\alpha'';\vec{\alpha}'$ after P's performance of $\vec{\beta};\beta'';\vec{\beta}'$, for any addition α'' and β'' in the roles".) In [9], we showed that this type of inferences is used for proving a property about a compound protocol, directly composing proofs of its components.

When we can freely apply the weakening rule to φ, we call φ a *monotonic* property[1]. Freshness, sending-fact, receiving-fact are examples of monotonic properties. In [9], we took as an example a set of monotonic properties, and demonstrated that such an inference system has enough power to prove *(non-injective) agreement property* (in the sense of Woo-Lam [10]) of some protocols, even if we do not use logical negation, nested implications, or any temporal operators as introduced in [3, 4][2].

However, if we want to prove a property stronger than the agreement property, we need some non-monotonic properties. In this paper, we give the way to include some non-monotonic properties in our framework of compositional logic. As an example, we aim at proving *matching conversations* of *Challenge Response Protocol* [5] which was also shown in [3, 4]. This property is stronger than the agreement property, because we need to prove additional properties about the ordering of actions performed by the different principals. To prove this property, we introduce a non-monotonic property "$firstly\_sends$. We show a proof of this property in this extended system only by adding a few restrictions on the weakening rules previously shown and on the inference rules on a principal's honesty (called *honesty inferences*). In particular, we

[1] This notion of monotonic property is essentially the same as *persistent* property in the sense of [6, 3, 4], except that the notion of monotonicity is related not only to weakening for protocol actions (described in the square bracket "[ ]") but also to weakening for honesty assumptions.

[2] The reason why we do not need logical negation nor nested implications (nor, disjunctions in the right hand side of a sequent) is that we restrict the honesty inferences. By this restriction, in our framework each sequent is expressed by a Horn-clause, however it is trade-off against some kinds of inferences on honesty. (See also Section 5.)

do not use logical negation, nested implication and temporal operators, which are used in the original proof of [3, 4].

In this paper, we use the following notations (cf. Appendix A of [9]). The letters A, B, C,... (P, Q, R,..., resp.) are constants (variables , resp.) of principal's names. The capital letters $K, K', \ldots, K_1, K_2, \ldots$ and $N, N', \ldots, N_1, N_2, \ldots$ are constants of keys and of nonces, respectively, while the small letters $k, k', \ldots, k_1, k_2, \ldots$ and $n, n', \ldots, n_1, n_2, \ldots$ are variables of the same sorts as above. The letters $m, m', \ldots, m_1, m_2, \ldots$ are used to denote messages, and $\{m\}_K$ is the encryption of m with key K, and $\langle m_1, \ldots, m_n \rangle$ is the concatenation of messages $m_1, \ldots, m_n$. We also introduce $m \sqsubseteq m'$ to represent the subterm relation as a meta symbol.

The rest of this paper is organized as follows. In Chapter 2, we shall review the inference system introduced in [9]. In Chapter 3, we shall show how to include non-monotonic properties in the system. Moreover, as an example, we prove the matching conversations of CR protocol which cannot be proved only by the monotonic properties. In Chapter 4, we shall give a semantics based on the notion of trace, and sketch out a soundness proof of the extended system. In Chapter 5, we shall present our conclusions and some further issues.

2 Inference system

2.1 The Language

Predicates of our inference system are as follows: $P\ generates\ n$, $P\ receives\ m$ [3], $P\ sends\ m$, $PK(P,k)$, $P \overset{K}{\leftrightarrow} Q$, $fresh(n)$ and $t = t'$. While the first three predicates are called *action predicates (performed by P)*, the rest of them are called *non-action predicates*. The letters $\alpha, \beta, \gamma, \delta, \ldots, \alpha', \alpha'', \ldots, \alpha_1, \alpha_2, \ldots$ are used to denote action predicates (also $\alpha^P, \beta^P, \gamma^P, \delta^P, \ldots$ to denote action predicates performed by P) and $\theta, \theta', \ldots, \theta_1, \theta_2, \ldots$ are non-action predicates. All those predicates except for equality are chosen from the BAN logic predicates [1]. Equality is used for explicit treatment of substitutions. As we have mentioned in Section 1, all those predicates except for $sends$ have *monotonic* properties (i.e., properties independent of the weakening rules for principal's actions and for honesty assumptions)[4].

As logical connectives, we introduce only usual conjunction (denoted by ",") and non-commutative conjunction (denoted by ";"). Our intention is to use non-commutat-

[3] We distinguish two kinds of "$receives$": the simple receiving and the receiving with decryptions. $P\ receives\ m(\{m'\}_K^*)$ means that "P receives a term m and decrypts the indicated subterm $\{m'\}_K^*$ of m. For a more formal description, instead of using $*$, we could introduce a new predicate $decrypts$ and describe it by $(P\ receives\ m) \wedge (P\ decrypts\ \{m'\}_K)$.

[4] As we shall see in the explanation of the Matching rule of the honesty inferences below, predicate "$sends$" is monotonic w.r.t. the weakening for concrete actions, however it is non-monotonic w.r.t. the weakening for honesty assumptions. In other words, this predicate is non-monotonic in the sense of our terminology, however it is "persistent" in the sense of [6, 3, 4].

ive conjunction to represent a sequence of principals' actions, implicitly treated in [9]. While in [3, 4] some temporal operators are used to reason about the ordering of actions, we get rid of any temporal operators: the orderings are directly derived from the axioms, using inference rules presented in Appendix. We introduce the vector notation such as $\vec{\alpha}$ to denote a sequence (i.e., non-commutative conjunct) of action predicates. We also introduce some notions related to a sequence of action. We say $\alpha_i \in \vec{\beta}$ $(= \beta_1; \cdots; \beta_n)$ if $\alpha_i = \beta_j$ for some $j = 1, \ldots, n$. For a sequence $\vec{\alpha} = \alpha_1; \ldots; \alpha_n$ and for $\alpha_i, \alpha_j \in \vec{\alpha}$, we denote $\alpha_i \leq_{\vec{\alpha}} \alpha_j$ if $i \leq j$. For $\vec{\alpha}$ and $\vec{\beta}$ $(= \beta_1; \ldots; \beta_n)$, if $\alpha_i \in \vec{\beta}$ for all $\alpha_i \in \vec{\alpha}$ and if $\forall \alpha_i, \alpha_j \in \vec{\alpha}.(\alpha_i \leq_{\vec{\alpha}} \alpha_j \Rightarrow \alpha_i \leq_{\vec{\beta}} \alpha_j)$, we say $\vec{\beta}$ is an *extension* of $\vec{\alpha}$ and denote it by $\vec{\alpha} \subseteq \vec{\beta}$.

Our inference system uses a sequent calculus style assertion. The basic form of assertion is as follows (where Q_i may be Q_j in the list of $P, \ldots, Q$).

$$Honest(\vec{\alpha}^P), \ldots, Honest(\vec{\beta}^Q), \Delta \vdash [\vec{\gamma}]_A \, \varphi$$

Here each of $\vec{\alpha}^P, \ldots, \vec{\beta}^Q$ is a sequence of action predicates performed by $P, \ldots, Q$, respectively, which represents a part of his/her role, and $\vec{\gamma}$ is a sequence of concrete actions performed by A.[5] Each of the letters $\varphi, \varphi', \ldots, \varphi_1, \varphi_2, \ldots$ is a sequence (i.e., non-commutative conjunct) of action predicates, or a single non-action predicate. Δ is of the form $\varphi_1, \ldots, \varphi_n$. Each predicate of the form $Honest(\vec{\alpha}^P)$ represents "a principal P honestly follows a part of role $\vec{\alpha}$". We call it P's *honesty assumption*. To formalize such assumptions on honesty, in [6, 3, 4], they introduce the predicate $Honest(P)$ which means "principal P is honest". On the other hand, our intention is to make more explicit the compositionality of honesty assumptions: we separate each honest principal's role into his/her primitive actions, and construct a composed proof by using some basic natural logical rules. (The details of the composing process were presented in [9].)

If $\vec{\alpha}^P$ consists of a sequence of primitive actions $\alpha_1^P; \cdots; \alpha_n^P$, we can consider the predicate $Honest(\vec{\alpha}^P)$ as an abbreviation of $Honest(\alpha_1^P); \cdots; Honest(\alpha_n^P)$, which is a conjunct of non-commutative conjunction.

Therefore, the intuitive meaning of the sequent previously introduced is "if principals $P, \ldots, Q$ honestly follow the parts of their roles $\vec{\alpha}^P, \ldots, \vec{\beta}^Q$, respectively, and if some properties Δ hold, then after A performs a sequence of concrete actions $\vec{\gamma}$, φ holds from A's viewpoint". (Here $\vec{\gamma}$ may be empty. In such case we often use φ, instead of $[\,]\varphi$.)

Finally, we introduce the postfix notation $[\vec{P}, \vec{n}, \vec{k}]$ in order to denote the lists of principal names $\vec{P}$ (list of variables $P_1, \ldots, P_m$), and the lists of variables of nonces and session keys $\vec{n}, \vec{k}$ (as variables). Substitutions are represented in terms of this notation.

[5] Note that for describing a sequence of action, while compositional logic of [6, 3, 4] uses the *cord calculus*, we describe it by the predicates previously shown.

2.2 Axioms and inference rules

Our inference system consists of the following four classes **(I)**-**(IV)** of axioms and of inference rules. (The complete list is also presented in Appendix.)

(I) Logical inferences with equality: As logical inferences, we use some structural rules (I-1) (weakening, contraction, exchange rules of the left hand side, and cut rule) and equality inference rules (I-2), and substitution rules (I-3). These are chosen from the traditional first order logic with equality. We also introduce some inference rules for non-commutative conjunction (I-4).

(II) Action properties axioms: These are composed of the *axioms about actions* and the *axioms for relationship between properties* in the sense of [6]. Our proposed axioms are listed in (II-1) and (II-2), respectively. (Here, axioms including non-monotonic property $firstly\_sends$, which is introduced in Section 3, are marked with the symbol "†".) However, our framework does not depend on any specific set of axioms in this class.

(III) Honesty inferences:

For deriving Q's other actions from P' viewpoint, P may assume Q's honesty and may use his/her own knowledge about Q's role in the protocol. For example, if P knows that Q has sent the message m in a current run, and assumes that Q is honest, then P can derive Q's previous action, because Q should not have sent the message m if he/she has not already performed all the previous actions of his/her role. For formalizing such inferences, compositional logic in [6, 3, 4] uses a special inference (called *honesty rule*) for deriving a conclusion of the form $Honest(Q) \supset \varphi$. On the other hand, in our system, inferences on honesty are formalized by the three kinds of inference rules: *Substitution* (III-1), *Matching* (III-2) and *Deriving another action* (III-3). These are called *honesty inferences*. For example, the following inference rule is the Matching rule.

$$\frac{\Delta \vdash [\vec{\alpha}]_P\ \vec{\beta}; (Q\ sends\ m); \vec{\gamma}}{\Delta, Honest(Q\ sends\ m', m) \vdash [\vec{\alpha}]_P\ \vec{\beta}; (Q\ sends\ m'); \vec{\gamma}}\ \textbf{Hon(Match)}$$

(Here $\vec{\beta}$ and $\vec{\gamma}$ are non-commutative conjuncts of some action predicates, respectively, (where each of them may be empty), and $m \sqsubseteq m'$.)

The intended meaning of this inference rule is that "if P assumes that Q is honest and follows the sending action "$Q\ sends\ m'$", and if P knows that Q sends a message m containing m'", then we can conclude that "P knows that Q has sent m'". This inference holds whenever the additional condition is satisfied such that "Q's honesty assumption does not include any other sending action of a message which includes m as a subterm". This means that the formula $Q\ sends\ m'$ appearing in the lower sequent is non-monotonic. Thus, to keep our system monotonic, we restrict all applications of honesty inferences and of weakening rule for honesty assumptions (explained in the

next item (IV)) so as to preserve this condition. More formally, we extend the language by introducing a new predicate $Honest(\alpha, m)$ (here $Honest(\alpha)$ previously defined can be regarded as a special case such that m is empty), and all applications of the honesty inferences and the weakening rule for honesty assumptions are restricted by the following condition (denoted by ($\sharp$)).

($\sharp$) Both predicates $Honest(Q\ sends\ m', m)$ and $Honest(Q\ sends\ m'')$ (with $m \sqsubseteq m''$) do not appear in the left hand side of the lower sequent.

(IV) Weakening rules for actions and for honesty assumptions: We now introduce the *weakening rules for honesty assumptions* and the *weakening rule for concrete actions*. All the applications of these weakening rules are restricted so as to satisfy the ($\sharp$) condition of Matching rule of honesty inferences.

If we introduce a non-monotonic predicate, as we shall explain in the next section, some additional condition should be necessary. In other words, our choice of predicates is one of the simplest formalism with respect to the weakening rules.

Finally we point out a limitation of our system. For a protocol including duplications of the same actions, we cannot distinguish one from another in our logic, because our logic does not explicitly deal with position during the run of a protocol. In this paper we consider only protocols which does not include any duplication.

3 Introducing a non-monotonic property in correctness proofs

In this section, we give the way to include some non-monotonic properties in our inference system presented in the previous section. In Section 3.1, as an example, we introduce a non-monotonic predicate "$firstly\_sends$" which is used to reason about some ordering of actions, and explain some additional restrictions on the honesty inferences and the weakening rules. In Section 3.2, we show a proof of *matching conversations* of *Challenge Response Protocol* [5]. This property was already proved in [3, 4], however logical negation, nested implications and temporal operators are not used in our proof.

3.1 Inferences for non-monotonic properties

In order to prove our aimed property, whereas the predicate $Fresh$ is used in [3, 4], we introduce a new predicate $firstly\_sends$ (also denoted by $fsends$ for readability). $P\ fsends\ (m, n)$ means "P sends a message m containing n as a subterm, and P does not send any other message m' containing n before the sending of m". Clearly, this predicate is non-monotonic, because if $\vdash [\vec{\alpha}; \alpha''; \vec{\alpha}']\ P\ fsends\ (m, n)$ holds, where α'' is P's sending of m, and if we insert another P's sending of m' with $n \sqsubseteq m'$ before α'', then this predicate becomes false under this weakened assumption (in square

bracket "[ ]"). As this observation tells us, if we introduce a non-monotonic predicate in our framework, we must restrict all applications of the weakening rules (both for honesty assumptions and for actions) and all the honesty inferences ((III-1)-(III-3)) by the following additional conditions (denoted by $(\sharp\sharp)$).[6]

> $(\sharp\sharp)$ For each sequence $\vec{\alpha}^P$, $\vec{\beta}^Q$, $\vec{\gamma}$ and for each $\vec{\delta}$ (which appears in Δ) in the lower sequent $Honest(\vec{\alpha}^P), \ldots, Honest(\vec{\beta}^Q), \Delta \vdash [\vec{\gamma}]\varphi$, it is no extension of a sequence of the form $(P\ sends\ m'); (P\ fsends\ (m,n))$ (with $n \sqsubseteq m, m'$) for any P.

Some non-monotonic properties are useful for reasoning about ordering of actions. Actually in the case of $fsends$, the order between different principals' actions can be derived by the additional inference rule and axiom as follows.

Firstly Sends:

$$\frac{Honest(\vec{\alpha}^P), \ldots, Honest(\vec{\beta}^Q), \Delta \vdash [\vec{\gamma}]\vec{\delta}; (R\ sends\ m); \vec{\delta}'}{Honest(\vec{\alpha}'^P), \ldots, Honest(\vec{\beta}'^Q), \Delta \vdash [\vec{\gamma}']\vec{\delta}; (R\ fsends\ (m,n)); \vec{\delta}'}$$

(Here R may be in the list of $P, \ldots, Q$, and $n \sqsubseteq m$, and each of $\vec{\alpha}'^P, \vec{\beta}'^Q$ and $\vec{\gamma}'$ is obtained from $\vec{\alpha}^P, \vec{\beta}^Q$ and $\vec{\gamma}$ by replacing all occurrences of $(R\ sends\ m)$ with $(R\ fsends\ (m,n))$.

Ordering of Actions:

$(P\ generates\ n), (P\ fsends\ (m,n)), \alpha \vdash (P\ fsends\ (m,n)); \alpha$
(where α is an action predicate of message m' with $n \sqsubseteq m'$.)

Firstly Sends is used to derive a $fsends$ predicate, and **Ordering of Actions** is used to derive an order of actions performed by different principals. (This is essentially the same as **AF3** presented in Table 5 of p.23 of [4].) For the same reason as the case of weakening rules and honesty inferences, we should restrict the application of **Firstly Sends** by the same condition $(\sharp\sharp)$.

3.2 An example of correctness proof

Table 1 is the full proof of matching conversations of CR protocol from initiator A's viewpoint. A proof of the same conclusion is presented in Table 10 of p.49 in [4]. CR protocol $\Pi[P, Q, n_1, n_2]$ described in an informal description is as follows.

[6] Here we point out another way for keeping the weakening rule meaningful: introducing the logical negation and separating the weakening rules into two rules as follows.

$$\frac{\Gamma \vdash [\vec{\alpha}; \vec{\alpha}']P\ fsends\ (m,n)}{\Gamma \vdash [\vec{\alpha}; \alpha''; \vec{\alpha}']\varphi}\ \textbf{W(Act)}$$

Here if α'' is P's sending of message m' such that $n \sqsubseteq m'$, and $\vec{\alpha}'$ includes another P's sending of message m, then φ is $\neg P\ fsends\ (m,n)$, and if not, then φ is $P\ fsends\ (m,n)$. This way is essentially the same as [3, 4]. (The *Freshness Loss Axiom* in Table 4 of p.22 in [4] is the corresponding axiom.)

PA	$fresh(N_1) \vdash [\vec{\alpha}]A \; sends \; \langle a, b, N_1 \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(1)
(1),**NV1**	$fresh(N_1) \vdash [\vec{\alpha}]A \; sends \; \langle a, b, N_1 \rangle; B \; sends \; m;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(2)
(2),**Hon(M)**	$H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}]A \; sends \; \langle a, b, N_1 \rangle;$ $B \; sends \; \langle q, p, n_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(3)
(3),**Hon(S)**	$H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}]p = A, q = B, n_1 = N_1,$ $n_2 = N_2$	(4)
(3),(4),**Eq**	$H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}]A \; sends \; \langle a, b, N_1 \rangle;$ $B \; sends \; \langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(5)
(4),**Hon(R)**,**Eq**	$H(\vec{\beta}); H(\vec{\gamma}), H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}]B \; receives$ $\langle a, b, N_1 \rangle; B \; sends \; \langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle$	(6)
(1),**FS**	$fresh(N_1) \vdash [\vec{\alpha}']A \; fsends \; \langle a, b, N_1 \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(7)
(6),**FS**	$H(\vec{\beta}); H(\vec{\gamma}), H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}']B \; receives$ $\langle a, b, N_1 \rangle; B \; fsends \; (\langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle, N_2)$	(8)
(7),(8),**OA**	$H(\vec{\beta}); H(\vec{\gamma}), H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}']A \; fsends$ $\langle a, b, N_1 \rangle; B \; receives \; \langle a, b, N_1 \rangle$	(9)
(5),(8),**OA**	$H(\vec{\beta}); H(\vec{\gamma}), H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}']B \; fsends$ $(\langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle, N_2);$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(10)
(8),(9),(10), **;**	$H(\vec{\beta}); H(\vec{\gamma}), H(\vec{\gamma}), H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}']A \; fsends$ $\langle a, b, N_1 \rangle; B \; receives \; \langle a, b, N_1 \rangle;$ $B \; fsends \; \langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(11)
Hon(W),**Cont** (11), **;**	$H(\vec{\beta}); H(\vec{\gamma}'), fresh(N_1) \vdash [\vec{\alpha}']A \; fsends \; \langle a, b, N_1 \rangle;$ $B \; receives \; \langle a, b, N_1 \rangle;$ $B \; fsends \; \langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle;$ $A \; receives \; \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$	(12)

Table 1: A's view at end of run following the initiator's role of Challenge-Response Protocol

1. $P \to Q$. $\langle p, q, n_1 \rangle$
2. $Q \to P$. $\langle q, p, n_2, \{n_2, n_1, p\}_{K_Q^{-1}} \rangle$
3. $P \to Q$. $\langle p, q, \{n_1, n_2, q\}_{K_P^{-1}} \rangle$

In this table, for readability we use some abbreviations as follows. $\vec{\alpha}$ is a sequence $A\ sends\ \langle a, b, N_1 \rangle$; $A\ receives\ \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$; $A\ sends\ \langle a, b, \{N_1, N_2, b\}_{K_A^{-1}} \rangle$. $\vec{\alpha}'$ is the sequence obtained from $\vec{\alpha}$ by replacing the first action $A\ sends\ \langle a, b, N_1 \rangle$ with $A\ fsends\ (\langle a, b, N_1 \rangle, N_1)$. Each symbols $H(\vec{\beta})$, $H(\vec{\gamma})$ and $H(\vec{\gamma}')$ are abbreviations of $Honest(Q\ receives\ \langle p, q, n_1 \rangle)$, $Honest(Q\ sends\ \langle q, p, n_2, \{n_2, n_1, p\}_{K_Q^{-1}} \rangle)$ and $Honest(Q\ fsends\ (\langle q, p, n_2, \{n_2, n_1, p\}_{K_Q^{-1}} \rangle, \{n_2, n_1, p\}_{K_Q^{-1}}))$, respectively. We also omit the predicates concerning information about keys: in this protocol, K_A and K_B are the public keys for A and B, respectively, and K_A^{-1} and K_B^{-1} are the secret part of these keys, respectively. Moreover, some predicates not related to the derived predicates at each line are omitted. (i.e., we implicitly use the contraction rules of non-commutative conjunction.) Finally, m on Line (2) is a message such that $\{N_2, N_1, a\}_{K_B^{-1}} \sqsubseteq m$.

First we would like to focus attention on the use of non-monotonic predicate *fsends*. On Line (6), the conclusion is the agreement property from A's viewpoint. Note that we do not use *fsends* to prove the agreement property. Therefore, if we want to prove the agreement property of this protocol, as we have also shown in our previous paper [9], we do not need to introduce any non-monotonic predicate.

The predicate $fsends$ is used to derive the orderings between A's action and B's one. Particularly, the order $B\ fsends\ \langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle$; $A\ receives\ \langle b, a, N_2, \{N_2, N_1, a\}^*_{K_B^{-1}} \rangle$ on Line (10) is derived by **Firstly Sends** and **Ordering of Actions**, and it cannot be derived without such non-monotonic notion. (The same property is derived on Line (10) in the example of [4] by means of the **AF3** axiom.) In this proof, $fsends$ is introduced on Line (7) by using the cut rule and the weakening rule for actions as follows: first, by applying the cut rule to (1) and **FSends** we obtain the sequent $fresh(N_1) \vdash [\,]A\ sends\ \langle a, b, N_1 \rangle; A\ receives\ \langle b, a, \ldots \rangle$, and then by applying some weakening rules for actions we obtain (7). Here we point out that at the second step each application of the weakening rule is restricted by ($\sharp\sharp$) condition that "no predicate of the form $A\ sends\ m$ (with $N_1 \sqsubseteq m$) does not appear before $A\ fsends\ (\langle a, b, N_1 \rangle, N_1)$ in the action operator (i.e., in the square bracket [ ])". However, in this case, we can obtain the sequent (7) by any order of applications of weakening rules for the components of this sequence. (Line (8) is similar to (7).)

By introducing the non-monotonic predicate, all weakening inferences for concrete actions and for honesty assumptions below the application of **Firstly Sends** on Line (7) and (8) are restricted by ($\sharp\sharp$) conditions that "A's sending of $\langle a, b, N_1 \rangle$ (introduced at Line (7)) and B's sending of $\langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}} \rangle$ (introduced at Line (8)) cannot be inserted before the corresponding actions". The information about orderings are necessary for proving our aimed property. Therefore, if we want to prove

a weaker property such as the agreement property, as we have shown in our previous paper [9], we don't have to introduce such non-monotonic predicates. Note that in our system we do not use any temporal operators for deriving properties about the orders of principals' actions: our formalization only use the non-commutative conjunction.
Remark. In our proof, Line (3) is derived by honesty inference **Hon(Match)**. The intended meaning of this inference is "if B sends a message including $\{N_2, N_1, a\}_{K_B^{-1}}$, then B should send $\langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}}\rangle$ under the assumption that B honestly follows $\vec{\gamma}$". we restrict applications of the weakening rules to satisfy the ($\sharp$) condition that "Q's sending action of a message including $\{N_2, N_1, a\}_{K_B^{-1}}$ does not appear in the honesty assumptions". Here we point out that we formalize this kind of inference by some restrictions instead of using logical negation. On the other hand, in the proof in [4], the same conclusion is derived by the honesty assumption that "if B does not freshly generate N_2 as a fresh value, then B should send $\langle b, a, N_2, \{N_2, N_1, a\}_{K_B^{-1}}\rangle$ under the assumption B is honest". This implication is obtained by using $\neg Fresh$.

4 Trace Semantics and Soundness of the System

In this section we give a semantics for our inference system. First we give a definition of the semantics (in Section 4.1), and next we give a sketch of soundness proof of our system (in Section 4.2).

4.1 Trace Semantics

Our semantics is based on the notion of *trace*, which is a sequence of *states*. A state is a multiset of *primitive states* of the form "principal P has information m", and denoted by $P(m), Q(m), \ldots$. We also introduce a special kind of primitive state "message m sent by P is currently transmitted through the network", and denoted by $Net(m, P)$. The notion of state is defined by the same way in *Multiset Rewriting System* [2], however we use it as semantic notion.

For preparing the definition of the semantics, here we introduce some notions and notations. $s_0, s_1, \ldots$ are used to denote states and $\mathtt{s}, \mathtt{s}', \ldots$ to denote sequences of states, namely traces. We also introduce some notions related to traces. The notions of membership relation (denoted by $s_i \in \mathtt{s}$), order relation on $\mathtt{s}$ (denoted by $\leq_{\mathtt{s}}$), and extension (denoted by $\mathtt{s} \subseteq \mathtt{s}'$) are defined by the same ways as those of sequences of actions. (See Section 2.1.) When s_i is the i-th element of $\mathtt{s}$, the number i is called the *position* of s_i in $\mathtt{s}$. We denote the number of occurrence of facts $P(m)$ in a state s_i by $\parallel s_i \parallel_{P(m)}$ (e.g. if $s_i = \{P(m), P(m), Q(m)\}$, then $\parallel s_i \parallel_{P(m)} = 2$). $Key(P, s_i)$ is used to denote the set of key possessed by principal P at position s_i. For messages m, m' and a set of keys $\{k_1, \ldots, k_l\}$, "m is *accessible* in m' with keys $\{k_1, \ldots, k_l\}$" (denoted by $m \in_{\{k_1,\ldots,k_l\}} m'$) is the reflexive-transitive closure satisfying the following conditions: (i) $m_i \in_{\{k_1,\ldots,k_l\}} \langle m_1, \ldots, m_n\rangle$ for some $i = 1, \ldots, n$,

(ii) $m \in_{\{k_1,\ldots,k_l\}} \{m\}_{k_j}$ for some $j = 1, \ldots, l$.

In this section, we assume that all states except for states of network are monotonously increasing for any trace. That is, we consider only traces where, once an information is possessed by a principal, it does not disappear in his/her memory.

By means of the notion of traces, truth conditions for predicates of our syntax is defined as follows. We denote the basic semantic relation "φ is true at state s_i in $\mathtt{s}$" by "$\models_{\langle \mathtt{s},i\rangle} \varphi$".

Truth condition for predicates:

- $\models_{\langle \mathtt{s},i\rangle} PK(P,k)$ iff $P(k'), KeyPair(k,k') \in s_i$ and $\forall X \neq P.(X(k') \notin s_i)$.
- $\models_{\langle \mathtt{s},i\rangle} P \overset{k}{\leftrightarrow} Q$ iff $P(k), Q(k) \in s_i$ and $\forall X \neq P, Q.(X(k) \notin s_i)$.
- $\models_{\langle \mathtt{s},i\rangle} t = t'$ iff $s_i[t/x] = s_i[t'/x]$. (for any terms t and t')
- $\models_{\langle \mathtt{s},i\rangle} P \; sends \; m$ iff $P(m) \in s_{i-1}$, $Net(m,P) \notin s_{i-1}$ and $Net(m,P) \in s_i$.
- $\models_{\langle \mathtt{s},i\rangle} P \; receives \; m(\{m_1\}^*_{k_1}, \ldots, \{m_n\}^*_{k_n})$ iff $\exists X.\ (Net(m,X) \in s_{i-1}$ and $Net(m,X) \notin s_i)$ and $\| s_{i-1} \|_{P(m)} +1 = \| s_i \|_{P(m)}$, and $\{m_j\}_{k_j} \in_{Key(P,s_i)} m$ and $\| s_{i-1} \|_{P(m_j)} +1 = \| s_i \|_{P(m_j)}$ for each $j = 1, \ldots, n$.
- $\models_{\langle \mathtt{s},i\rangle} P \; generates \; m$ iff $P(m) \notin s_{i-1}$ and $P(m) \in s_i$.
- $\models_{\langle \mathtt{s},i\rangle} fresh(m)$ iff $\exists X.(X(n) \notin s_{i-1}$ and $X(n) \in s_i)$ and $n \sqsubseteq m$.
- $\models_{\langle \mathtt{s},i\rangle} P \; fsends \; (m,n)$ iff $\models_{\langle \mathtt{s},i\rangle} P \; sends \; m$ and $n \sqsubseteq m$ and $\forall j < i. \forall m' \sqsupseteq m.(\not\models_{\langle \mathtt{s},j\rangle} A \; sends \; m')$.
- $\models_{\langle \mathtt{s},i\rangle} \alpha_1; \cdots; \alpha_n$ iff $\models_{\langle \mathtt{s},i_1\rangle} \alpha_1$ and $\cdots$ and $\models_{\langle \mathtt{s},i_n\rangle} \alpha_n$, and $i_1 \leq \cdots \leq i_n \leq s_i$.

Next, the definition "φ is true for trace $\mathtt{s}$" (denoted by $\models_{\mathtt{s}} \varphi$) is as follows.

- $\models_{\mathtt{s}} \beta$ iff $\forall s_i \in \mathtt{s}.(\models_{\langle \mathtt{s},i\rangle} \beta)$ (where $\beta = PK(P,k)$ or $P \overset{k}{\leftrightarrow} Q$, or $t = t'$.)
- $\models_{\mathtt{s}} fresh(m)$ iff $\exists s_i \in \mathtt{s}.(\models_{\langle \mathtt{s},i\rangle} fresh(m))$.
- $\models_{\mathtt{s}} \alpha_1; \cdots; \alpha_n$ iff $\exists s_i \in \mathtt{s}.(\models_{\langle \mathtt{s},i\rangle} \alpha_1; \cdots; \alpha_n)$ (where each α_i is an action predicate.)

We define that $\models_{\mathtt{s}} \Gamma$ iff "$\models_{\mathtt{s}} \vec{\alpha}$ and ... and $\models_{\mathtt{s}} \vec{\beta}$, and $\models_{\mathtt{s}} \theta_i$ for each $i = 1, \ldots, n$" (where $\Gamma = \vec{\alpha}, \ldots, \vec{\beta}, \theta_1, \ldots, \theta_n$). By the above definition, it is clear that for any φ except for $fsends$ (i.e., for the case that φ is monotonic), if $\models_{\mathtt{s}} \varphi$ then $\models_{\mathtt{s}'} \varphi$ for any $\mathtt{s} \subseteq \mathtt{s}'$.

In terms of the above definitions, we define that the basic form of assertion is true under $\mathtt{s}$, namely,

$$Honest(\alpha_1^P); \cdots; Honest(\alpha_n^P), \ldots Honest(\alpha_1^Q); \cdots; Honest(\alpha_k^Q), \ldots, \Gamma \models_{\mathtt{s}} [\vec{\alpha}]\varphi$$

holds if and only if the following is satisfied (where φ is a state predicate or a sequence of action predicates).

If C1 $\forall i \leq n.\forall i' < i.(\models_{\mathtt{s}} \alpha_i^P \Rightarrow \models_{\mathtt{s}} \alpha_{i'}^P)$, and
$\forall j \leq k.\forall j' < j.(\models_{\mathtt{s}} \alpha_j^Q \Rightarrow \models_{\mathtt{s}} \alpha_{j'}^Q)$,
C2 $\exists \mathtt{s}'.(\mathtt{s} \subseteq \mathtt{s}' \wedge \forall i \leq n.(\models_{\mathtt{s}'} \alpha_i) \wedge \forall j \leq k.(\models_{\mathtt{s}'} \alpha_j))$,
C3 $\models_{\mathtt{s}} \Gamma$,
C4 $\models_{\mathtt{s}} \vec{\alpha}$,
then $\models_{\mathtt{s}} \varphi$.

Here for each honest predicate $Honest(\alpha_i^X)$, if it is of the form $Honest(\alpha_i^X, m_i^X)$ for $X = P, Q$ and for $i = 1, \ldots, n$ or $1, \ldots, k$ (i.e., m_i^X is not empty), then the following condition is also satisfied:

C5 $\forall m'.((m' \sqsupseteq m_i^X) \wedge (m' \neq m'') \wedge (\alpha_i^X = X\ sends\ m'') \Rightarrow \forall \mathtt{s}' \sqsupseteq \mathtt{s}.(\not\models_{\mathtt{s}'} X\ sends\ m'))$.

The reason why we need this additional condition is as follows: first recall that $Honest(\alpha_i^X, m_i^X)$ (where m_i^X is not empty term) means that "X honestly follows the sending action α_i^X (say, $X\ sends\ m''$) and he/she does not follow any other sending actions of the message m' including m_i^X". Therefore, to satisfy this restriction, we assume $X\ sends\ m''$ is false for any extension $\mathtt{s}'$ of $\mathtt{s}$.

If the above form of assertion is true for any trace $\mathtt{s}$, then this assertion is called *valid* and we omit the subscription $\mathtt{s}$.

Remind that in this paper we consider only protocols which do not include any duplication of primitive actions. We assume that all traces considered here are also restricted by the same condition. (Formally, for any state $\mathtt{s}$ and for any action predicate α, if $\models_{\langle \mathtt{s}, i \rangle} \alpha$ and $\models_{\langle \mathtt{s}, j \rangle} \alpha$ then $i = j$.)

4.2 Soundness of the System

In this subsection we show a sketch of a soundness proof of our system. In our previous paper [9], we presented a soundness proof for the system including only monotonic predicates. Then, here we only consider some of cases related to the non-monotonic predicate $fsends$.

Nonce verification 2: (where $\{m_1\}_K \sqsubseteq m_2, m_3$, and $\{m_1\}_K \not\sqsubseteq m_5$, and $n \sqsubseteq m_1, m_4, m_5$.)

$$(PK(K,Q)), (P\ fsends\ (m_2, n)), (P\ generates\ n), (P\ receives\ m_5)$$
$$\vdash (P\ fsends\ (m_2, n)); (Q\ receives\ m_3(\{m_1\}_K^*));$$
$$(Q\ sends\ m_4); (P\ receives\ m_5)$$

Assume that all the predicates appearing in the left hand side are valid. That is, for any $\mathtt{s}$ $(= s_1, \ldots, s_n)$, (i) $\forall j < n.\forall X \neq Q.\ ((Q(K^{-1}) \in s_j) \wedge (X(K^{-1}) \notin s_j))$, (ii) $\exists s_{i_1}.((\models_{\langle \mathtt{s}, i_1 \rangle} P\ sends\ m_2) \wedge (\forall j' < i_1.(\not\models_{\langle \mathtt{s}, j' \rangle} P\ sends\ m' \text{ with } n \sqsubseteq m')))$, (iii) $\exists i_2 < i_1.(\models_{\langle \mathtt{s}, i_2 \rangle} P\ generates\ n)$, (iv) $\exists i_3 > i_1.(\models_{\langle \mathtt{s}, i_3 \rangle} P\ receives\ m_5$ with $n \sqsubseteq m_5$, $\{m_1\}_K \not\sqsubseteq m_5)$. From (ii), (iii) and (iv), $\exists X \neq P.\exists i_4 < i_3.(\models_{\langle \mathtt{s}, i_4 \rangle} X\ sends\ m_5)$ holds, and then $\exists Y \neq P.\exists i_5 < i_4.((Y(n) \in s_5) \wedge (\forall Z \neq P, Y.(Z(n) \notin i_5)))$.

Then by (i) and (ii), $\exists l.((i_1 < l < i_5) \wedge (\models_{\langle \mathtt{s},l\rangle} Q\ receives\ m_3(\{m_1\}^*_K)))$ and $\exists l'.((l < l') \wedge (\models_{\langle \mathtt{s},l'\rangle} Q\ sends\ m_5)$ with $n \sqsubseteq m_4)$. This is the truth condition for $(Q\ receives\ m_3(\{m_1\}^*_K)); (Q\ sends\ m_4)$, and therefore, the sequence of actions in the right hand side of the sequent is true.

Firstly Sends:
(Without loss of generality, here we only consider a special case such that only P's honesty assumptions appear in the left hand side of each sequent and omit any context for readability.)

$$\frac{Honest(\alpha^P_1; \ldots; \alpha^P_n) \vdash [\vec{\gamma}]\vec{\delta}; (P\ sends\ m); \vec{\delta'}}{Honest(\alpha'^P_1; \ldots; \alpha'^P_n) \vdash [\vec{\gamma}]\vec{\delta}; (P\ fsends\ (m,n)); \vec{\delta'}}$$

(Here $n \sqsubseteq m$, and for each α'^P_i, $\alpha'^P_i = P\ fsends\ (m,n)$ if $\alpha^P_i = P\ sends\ m$, otherwise $\alpha'^P_i = \alpha^P_i$.)

Here it is clear that the soundness holds when $\alpha'^P_i = \alpha^P_i$ for all $i < n$ (i.e., $\vec{\alpha}^P$ does not contain $P\ sends\ m$). Then, from now we shall consider only the case that $\alpha^P_i = P\ sends\ m$ and $\alpha'^P_i = P\ fsends\ (m,n)$ for some $i < n$. (In this case, by the condition $(\sharp\sharp)$, $\vec{\delta}$ does not include $\delta_j = P\ sends\ m'$ with $n \sqsubseteq m'$ for all $j < i$.)

First, consider a trace $\mathtt{s}$ satisfying the conditions C1 and C2 (previously shown in the definition of truth condition for the sequent) for $Honest(\alpha'^P_1; \cdots; \alpha'^P_n)$. By the definition of the truth condition for $fsends$, $\mathtt{s}$ also satisfies the conditions C1 and C2 for $Honest(\alpha^P_1; \cdots; \alpha^P_n)$". Here we assume that the upper sequent is valid, then $\models_{\mathtt{s}} \vec{\delta}; P\ sends\ m; \vec{\delta'}$ holds. Moreover, if we assume that $\mathtt{s}$ satisfies C1 and C2 for $Honest(\alpha'^P_1; \cdots; \alpha'^P_n)$, then by C1, the following holds that "if $\models_{\mathtt{s}} \alpha'_i\ (= P\ fsends\ (m', n))$, then $\forall j < i.(\not\models P\ sends\ m')$ with $n \sqsubseteq m'$ and $m' \neq m$". Therefore, $\forall j < i.\ \not\models_{\langle \mathtt{s},j\rangle} P\ sends\ m''$ for any m'' with $m \sqsubseteq m''$. This is the truth condition for $\models_{\mathtt{s}} \vec{\delta}; P\ sends\ m$. Therefore the right hand side of the lower sequent is valid.

Weakening rules:

$$\frac{\Gamma, Honest(\vec{\alpha}^P; \vec{\alpha'}^P) \vdash [\vec{\beta}]\varphi}{\Gamma, Honest(\vec{\alpha}^P; \alpha''^P; \vec{\alpha'}^P) \vdash [\vec{\beta}]\varphi}\ \textbf{W(Hon)} \qquad \frac{\Gamma \vdash [\vec{\alpha};^P \vec{\alpha'}^P]\varphi}{\Gamma \vdash [\vec{\alpha}^P; \alpha''^P; \vec{\alpha'}^P]\varphi}\ \textbf{W(Act)}$$

(Here we only consider the case that α''^P is $P\ sends\ m$. It is similar way to prove the case that α''^P is $P\ fsends\ (m,n)$.)

(1) Weakening (Honesty):

Here we assume that the lower sequent satisfies the $(\sharp\sharp)$ condition. That is, $\vec{\alpha}'^P$ does not include any action of the form $P\ fsends\ (m', n)$ with $n \sqsubseteq m'$. It is sufficient to show that for any trace $\mathtt{s}$, "if $\mathtt{s}$ satisfies the conditions C1 and C2 for $Honest(\vec{\alpha}^P; \alpha''^P; \vec{\alpha'}^P)$, then $\mathtt{s}$ also satisfies the condition C1 and C2 for $Honest(\vec{\alpha}^P; \vec{\alpha}^P)$", however it immediately follows from the definition of the truth condition of $fsends$ and the $(\sharp\sharp)$ condition.

(2) Weakening (Actions):

By ($\sharp\sharp$) condition, we can assume that $\vec{\alpha}'^P$ does not include $P\, fsends\ (m', n)$ with $n \sqsubseteq m, m'$. It is sufficient to show that for any trace $\mathtt{s}$, $\models_{\mathtt{s}} \vec{\alpha}^P; (P\ sends\ m); \vec{\alpha}'^P$ then $\models_{\mathtt{s}} \vec{\alpha}^P; \vec{\alpha}'^P$, however, this immediately follows from the definition of $\models_{\mathtt{s}} \vec{\alpha}^P; (P\ sends\ m); \vec{\alpha}'^P$.

Remark. As for the weakening rule for honesty assumptions, if the lower sequent violate the ($\sharp\sharp$) condition (i.e., if $\vec{\alpha}'^P$ includes action predicate of the form $P\, fsends\ (m', n)$ with $n \sqsubseteq m'$), then there exists a trace such that it does not satisfy the conditions for $Honest(\vec{\alpha}^P; \alpha''^P; \vec{\alpha}'^P)$ whereas it satisfies $Honest(\vec{\alpha}^P; \vec{\alpha}'^P)$. The same is true of the other weakening rule.

5 Conclusions and future work

By the distinction between monotonic and non-monotonic predicates, we gave the way to include non-monotonic properties, and showed that they can be used in more powerful derivation to prove correctness properties of a protocol. As an example, we proved the matching conversations of CR protocol, where the ordering of actions performed by different principals are derived by **Firstly Sends** and **Ordering of Actions**. Other examples are **Nonce Verification** 2 and 3 presented in (II-2) of Appendix. These are formalizations of the notion of *Outgoing test* in the *Authentication tests* based *Strand space method* (cf. Guttman-Fábrega [7]). This notion can be formalized only by using non-monotonic property $fsends$ or similar one, which can represent the same notion of *uniquely originates* (in [7]).

In our extended system, we did not use logical negation, nested implications and any temporal operators to prove our aimed property, which were used in [3, 4]. (In other words, in our system each sequent is the form of Horn-clause.) This simplification is realized by the restriction on the honesty inferences. However, this restriction is a trade-off. For example, the following kind of inferences cannot be expressed in our system: assume that a principal (say P) is honest following a role $\vec{\alpha} = \alpha_1; \alpha_2; \alpha_3$ of a protocol. Then from this assumption, we can conclude that "$Honest(P) \wedge (P$ performs $\beta) \supset (\beta = \alpha_1) \vee (\beta = \alpha_2) \vee (\beta = \alpha_3)$" (i.e., "if P is honest and he/she performs an primitive action β then it is α_1 or $alpha_2$ or α_3"), because honest principal does not perform any other actions than the actions defined by his/her role. One of our next aims is to investigate the formalization of such inferences and clarify what kinds of properties (useful for a correctness proof) become provable in such extended system.

We also gave a semantics based on the notion of trace and show a sketch of soundness proof. This direction should make a contribution to our further target, namely, automated generation of correctness proofs or correct protocols for example.

Acknowledgments

We would like to express our sincere thanks to Drs. Andre Scedrov and Iliano Cervesato for their invaluable comments and discussions. We also would like to express our sincere thanks to Mr. Lam Ta-Minh for his helpful comments. Finally, helpful comments from the anonymous reviewers results in improvements to this paper.

References

[1] M. Burrows, M. Abadi and R. Needham. A Logic of Authentication. *Technical Report 39,* Digital System Research Center, 1989.

[2] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, vol.12, no.1, pp.677-622, 2004.

[3] A. Datta, A. Derek, J. C. Mitchell and D. Pavlovic. Secure Protocol Composition. *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics, ENTCS Vol. 83*, 2004.

[4] A. Datta, A. Derek, J. C. Mitchell and D. Pavlovic. A Derivation System for Security Protocols and its Logical Formalization. *Journal of Computer Security, Special Issue of Selected Papers from CSFW-16*, 2004.

[5] W. Diffie, P. C. van Oorschot and M. J. Wiener. Authentication and authenticated key exchange *Designs, Codes and Cryptography*, vol.2, pp.107-125, 1992.

[6] N. Durgin, J. Mitchell and D. Pavlovic. A Compositional Logic for Protocol Correctness. *Journal of Computer Security (Special Issue of Selected Papers from CSFW-14)*, 11(4), pp.677-721, 2003.

[7] J. D. Guttman and F. J. T. Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, vol. 283(2), pp.333-380, 2002.

[8] K. Hasebe and M. Okada. A Logical Verification Method for Security Protocols Based on Linear Logic and BAN Logic. M. Okada, B. Pierce, A. Scedrov, H. Tokuda and A. Yonezawa (eds.), *Software Security — Theories and Systems*, Lecture Notes in Computer Science, Hot Topics, vol.2609, Springer-Verlag, pp.417-440, 2003.

[9] K. Hasebe and M. Okada. Inferences on Honesty in Compositional Logic for Security Analysis. to appear in *Proceedings of the International Symposium on Software Security (ISSS2003)*, Springer LNCS, 2004.

[10] T. Y. C. Woo and S. S. Lam. Verifying authentication protocols: Methodology and example. *Proceedings of the International Conference on Network Protocols*, 1993.

Appendix: axioms and inference rules of the system

(I) Logical inference rules

(1) Structural rules: weakening, contraction and exchange rules in the left hand side, and cut rule (**Cut**) (2) Inference rules for equality (**Eq**) (a typical rule which we often use is presented below), (3) Substitution rule (**Subst**), (4) Inference rules for non-commutative conjunction (**;**): Concatenation, Weakening and Contraction.

$$\frac{\Gamma \vdash [\vec{\alpha}]\varphi \quad \varphi, \Delta \vdash [\vec{\alpha}]\varphi'}{\Gamma, \Delta \vdash [\vec{\alpha}]\psi}\ \textbf{Cut} \qquad \frac{\Gamma \vdash [\vec{\alpha}]x = t \quad \Delta \vdash [\vec{\alpha}]\varphi[x]}{\Gamma, \Delta \vdash [\vec{\alpha}]\varphi[t/x]}\ \textbf{Eq;}$$

$$\frac{\Gamma \vdash [\vec{\alpha}]\vec{\beta}; \beta'' \quad \Delta \vdash [\vec{\alpha}']\beta''; \vec{\beta}'}{\Gamma, \Delta \vdash [\vec{\alpha}; \vec{\alpha}']\vec{\beta}; \beta''; \vec{\beta}'}\ \textbf{Concat-} \qquad \frac{\Gamma[x] \vdash [\vec{\alpha}[x]]\varphi[x]}{\Gamma[t/x] \vdash [\vec{\alpha}[t/x]]\varphi[t/x]}\ \textbf{Subst;}$$

$$\frac{\vec{\beta}; \vec{\beta}', \Gamma \vdash [\vec{\alpha}]\varphi}{\vec{\beta}; \beta''\vec{\beta}', \Gamma \vdash [\vec{\alpha}]\varphi}\ \textbf{Weak-} \qquad \frac{\Gamma \vdash [\vec{\alpha}]\vec{\beta}; \beta''; \vec{\beta}'}{\Gamma \vdash [\vec{\alpha}]\vec{\beta}; \vec{\beta}'}\ \textbf{Cont-;}$$

(II-1) Axioms about primitive actions

$$\vdash [\alpha_1; \cdots; \alpha_n]\alpha_1; \cdots; \alpha_n$$

(II-2) Axioms for relationships between properties

(Here axioms including non-monotonic property are marked by †.)

Freshness 1:
$P\ generates\ n \vdash fresh(n)$

Freshness 2: (where $m \sqsubseteq m'$.)
$fresh(m) \vdash fresh(m')$

Nonce Verification 1: (where $\{m\}_{K^{-1}} \sqsubseteq m', m''$.)
$(PK(K, Q)), (fresh(m)), (P\ receives\ m'(\{m\}^*_{K^{-1}}))$
$\vdash (Q\ sends\ m''); (P\ receives\ m'(\{m\}^*_{K^{-1}}))$

Nonce verification 2†: (where $\{m_1\}_K \sqsubseteq m_2, m_3$ and $\{m_1\}_K \not\sqsubseteq m_5$ and $n \sqsubseteq m_1, m_4, m_5$.)
$(PK(K, Q)), (P\ fsends\ (m_2, n)), (P\ generates\ n), (P\ receives\ m_5)$
$\vdash (P\ fsends\ (m_2, n)); (Q\ receives\ m_3(\{m_1\}^*_K));$
$(Q\ sends\ m_4); (P\ receives\ m_5)$

Nonce verification 3†: (additionally to the condition for **Nonce Verification 2**, $\{m_1\}'_K \not\sqsubseteq m_5$ is also satisfied.)
$(PK(K, Q)), (P\ fsends\ (m_2, n)), (P\ generates\ n),$
$(P\ receives\ m_5), (Q\ sends\ \{m_4\}'_K), (PK(K', A)) \vdash m_5 = \{m_4\}'_K$

We also admit axioms obtained from Nonce verification 1-3 by replacing $PK(K, Q)$ with $P \overset{K}{\leftrightarrow} Q$, respectively.

Shared secret: (where $K' \sqsubseteq m_1, m_2$.)

$(P\ sends\ \{m_1\}_{K_1}), (P\ sends\ \{m_2\}_{K_2}), (P\ generates\ K'), (P \overset{K_1}{\leftrightarrow} Q), (P \overset{K_2}{\leftrightarrow} R) \vdash (Q \overset{K'}{\leftrightarrow} R)$

Firstly Sends†:

This rule satisfies ($\sharp\sharp$) condition (cf. Section 3.1).

$$\frac{Honest(\vec{\alpha}^P), \ldots, Honest(\vec{\beta}^Q), \Delta \vdash [\vec{\gamma}]\vec{\delta}; (R\ sends\ m); \vec{\delta}'}{Honest(\vec{\alpha}'^P), \ldots, Honest(\vec{\beta}'^Q), \Delta \vdash [\vec{\gamma}']\vec{\delta}; (R\ fsends\ (m,n)); \vec{\delta}'}$$

Ordering of Actions†: (where α is an action predicate of message m' with $n \sqsubseteq m'$.)

$(P\ generates\ n), (P\ fsends\ (m,n)), \alpha \vdash P\ fsends\ (m,n); \alpha$

(III) Honesty inferences

For (1) Substitution and for (3) Deriving another action, we also admit the inference rules obtained by replacing "$receives$" with "$generates$" or "$sends$", respectively. These rules satisfy the ($\sharp$) (cf. Section 2.2 (III)) and ($\sharp\sharp$) conditions.

(1) Substitution: (where $m \sqsubseteq m'$.)

$$\frac{\Gamma \vdash [\vec{\alpha}](Q\ receives\ m[t/x])}{\Gamma, Honest(Q\ receives\ m) \vdash [\vec{\alpha}]x = t}\ \textbf{H(S)}$$

(2) Matching:

$$\frac{\Gamma \vdash [\vec{\alpha}]\vec{\beta}; (Q\ sends\ m); \vec{\gamma}}{\Gamma, Honest(Q\ sends\ m', m) \vdash [\vec{\alpha}]\vec{\beta}; (Q\ sends\ m'); \vec{\gamma}}\ \textbf{H(M)}$$

(3) Deriving another action in a role: **H(R)**

$$\frac{\Gamma \vdash [\vec{\alpha}]\vec{\beta}; (Q\ sends\ m); \vec{\gamma}}{\Gamma, Honest_Q(Q\ receives\ m'; Q\ sends\ m) \vdash [\vec{\alpha}]\vec{\beta}; (Q\ receives\ m'); (Q\ sends\ m); \vec{\gamma}}$$

(IV) Weakening rules for actions and honesty assumptions

Weakening rule for honesty assumptions (left below) satisfies ($\sharp$) and ($\sharp\sharp$) conditions, and weakening rule for actions (right below) satisfies ($\sharp\sharp$) condition.

$$\frac{\Gamma, Honest(\vec{\alpha}^P; \vec{\alpha}'^P) \vdash [\vec{\beta}]\varphi}{\Gamma, Honest(\vec{\alpha}^P; \alpha''^P; \vec{\alpha}'^P) \vdash [\vec{\beta}]\varphi}\ \textbf{W(Hon)} \qquad \frac{\Gamma \vdash [\vec{\alpha}^P; \vec{\alpha}'^P]\varphi}{\Gamma \vdash [\vec{\alpha}^P; \alpha''^P; \vec{\alpha}'^P]\varphi}\ \textbf{W(Act)}$$

Session IV

Security Protocols II

A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer

Dennis Hofheinz[†] and Jörn Müller-Quade[†]

Abstract

This work develops a composable notion of security in a synchronous communication network to analyze cryptographic primitives and protocols in a reliable network with guaranteed delivery. In such a synchronous model the abort of protocols must be handled explicitly. It is shown that a version of *global bit commitment* which allows to identify parties that did not give proper input cannot be securely realized with the primitives *oblivious transfer* and *broadcast*. This proves that the primitives oblivious transfer and broadcast are not complete in our synchronous model of security.

In the synchronous model presented ideal functionalities as well as parties can be equipped with a "shell" which can delay communication until the adversary allows delivery or the number of rounds since the shell received the message exceeds a specified threshold. This additionally allows asynchronous specification of ideal functionalities and allows to model a network where messages are not necessarily delivered in the right order. If these latency times are chosen to be infinite the network is no more reliable and becomes completely asynchronous. In the full version [HMQ04] of this paper, it is shown that a large class of protocols which are secure in the asynchronous settings [Can01, CLOS02] can be transformed into secure realizations in the new model by choosing infinite latency times.

Keywords: Security protocols, oblivious transfer, protocol composition.

1 Introduction

In this contribution it is proven that in a communication network in which message delivery is guaranteed and participating parties are periodically activated, oblivious transfer together with a broadcast primitive are not complete for secure multi-party computations.

To show this separation between security in reliable networks and security in completely asynchronous networks a new synchronous model is developed. In addition to the properties of the synchronous models of [Can00] the new model al-

[**]IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth, Fakultät für Informatik, Universität Karlsruhe, Am Fasanengarten 5, 76 131 Karlsruhe, Germany

lows very general composition of protocols along the line of the asynchronous settings [Can01, PW01, BPW04]. The new model is a synchronous variation of [Can01] (for a relation, cf. Section 2.5). It differs from the synchronous variant sketched in [Can01], which was not suitable for our purpose as it does not guarantee activation of ideal functionalities in each round.

It might have been possible to formulate our main result in the frameworks of [PW01, BPW04, PW00]. Yet since machine modeling and scheduling there differs substantially from that in [Can01, CLOS02], it would be difficult to compare our result to established realizability results in the latter settings: Our goal is to point out the importance of reliability assumptions on a network for deducing hierarchies of primitives, and to relate our result to results derived for the asynchronous modelings of [Can01, CLOS02].

Our new model shares with the aforementioned notions of security the concept of *simulatability*. Intuitively, this means that a given protocol is compared to an idealization of the protocol task in question and considered secure if no difference can be detected by any protocol environment, or, an arbitrary user. There is already a (positive and constructive) general realizability result for protocol tasks in a synchronous variant of the setting [Can01], cf. [Can01, Theorem 9 of full version]. Another realizability result was established in [CLOS02], again for a slight (asynchronous) variation of the setting of [Can01]. Specifically, [CLOS02] present a protocol construction with which general reactive ideal functionalities (i. e., idealizations of protocol tasks) can be securely realized, given only a *common reference string*. (A common reference string, ideally drawn from a fixed distribution, can be considered an idealization of a public set-up information.)

However, the setting of [Can01] (and the mentioned variations) does not allow to formulate "timeouts" or functionalities which guarantee certain response times. In consequence, even secure protocols in that sense may "get stuck" or "hang" in face of corrupted parties, *even* if all protocol messages of the uncorrupted parties get delivered immediately.[1] Thus, we believe it is reasonable to investigate—in a simulatability-based setting—security properties of functionalities which *do* guarantee service. In Section 2, we therefore present a *synchronous* modeling of multi-party computation which allows for universal composition, and a result allowing to carry over realizability results established in the settings of [Can01, CLOS02] into our setting.

In the new model tools are provided to catch reliable or even asynchronous networks in our setting. In particular, we show that a protocol realizing a certain ideal functionality in the settings [Can01, CLOS02] realizes a similar functionality in our setting, *yet one in which it is made explicit that no response can be guaranteed*.

However, properties like guaranteed output and explicit abort can be especially important for real world applications—e. g., an electronic election or an electronic auction should not "hang", but should be robust to attacks like the one presented here.

[1] For the framework of [PW01, BPW04], this problem was addressed in [BPSW02].

If output is to be guaranteed, then aborting protocols must be handled explicitly. A synchronous, i. e., a completely reliable network allows to distinguish different kinds of abort—the most interesting of which is the abort with cheater identification. A commitment of one party to all parties to the same bit (a *global bit commitment*) becomes more challenging in a synchronous network as the ideal functionality aborts only if the committer refused to commit. Hence this protocol allows for cheater identification. To make this strong contrast to the asynchronous setting explicit we prove that it is not possible to securely implement a global bit commitment in our synchronous model given the cryptographic primitives of oblivious transfer and broadcast.

2 The modeling

2.1 Real and Hybrid Model

The real model is an abstraction of a *malicious* protocol environment as one would expect it in reality. Thus, a real-model adversary may read all messages sent between parties, or corrupt parties and then control their behavior. The hybrid model is a real model in which parties are additionally offered blackbox access to idealizations of (sub)protocols, henceforth called *ideal functionalities*.

All parties, adversaries and ideal functionalities are modelled as *interactive Turing machines (ITMs)*, just as in [Can01]. An ITM has read-only tapes for incoming communication and local input, write-once tapes for outgoing communication and local output, a work tape, a one-bit activation tape, a read-only random tape and read-only tapes containing machine identity and security parameter, respectively. Unless explicitly noted, any ITM mentioned in this work is assumed to be *polynomially bounded* in the sense that no matter with which tape contents activated, it terminates this activation within $p(k)$ steps (i. e., transitions) for a fixed, ITM-specific polynomial p and the value k on the security parameter tape. To reflect polynomial total length of a protocol run, the ITMs $\mathcal{Z}$ and $\mathcal{A}$ described below are assumed to *halt* after a polynomial number of activations. An ITM which has halted terminates instantly—without switching at all—on all future activations.

Aside from parties P_i and an adversary $\mathcal{A}$, an *environment machine* $\mathcal{Z}$ (modelled as an ITM[2]) takes part in a protocol run. $\mathcal{Z}$ represents an arbitrary procotol environment in which the investigated protocol is run as a subprotocol. In particular, $\mathcal{Z}$ supplies parties with input, reads their output and may even communicate with the adversary. In the simulatability-based definition of security given below, $\mathcal{Z}$ takes a crucial role.

To protect the polynomially bounded adversary from being activated "too often" by the environment, we introduce the following special capability of the adversary: $\mathcal{A}$ may enter a special class of states to signal that further messages from $\mathcal{Z}$ are not

[2]In [Can01], $\mathcal{Z}$ is the only *non-uniform* ITM, i. e., $\mathcal{Z}$ gets as initial input the value of an arbitrary function of the security parameter. We adopt this, but stress that all results below hold also for *uniform* $\mathcal{Z}$, cf. also the discussion in [HMQS03].

to be delivered to $\mathcal{A}$ and thus, such messages do not cause activation of $\mathcal{A}$. When $\mathcal{A}$ enters such a state, we say that $\mathcal{A}$ *blocks*. This convention resembles the mechanism of *length functions* used in [BPSW02] for similar purposes. Without such a convention and polynomially bounded adversaries which may not depend on the distinguishing environment, the environment may simply "kill" the adversary by activating it sufficiently often right at the start of the protocol. This is especially crucial for simulators (see below).

The real model can be seen as a (trivial) special case of the hybrid model, and hence it suffices to give a description of a protocol run in the $\{\mathcal{F}_i\}$-hybrid model for a *finite* set $\{\mathcal{F}_i\}$ of ideal functionalities. First some terminology: *Delivering* a message means *moving* it from the outgoing communication tape of the sending ITM to the incoming communication tape of the receiving ITM. Here we assume *authentication*: the sender identity is automatically added to the message at delivery. After an ITM terminates its activation, its incoming communication tape is automatically cleared to ensure future message processing.

All ITMs may, when active, of course access their own tapes; furthermore, $\mathcal{Z}$ may read the local output tapes of the P_i and write onto their local input tapes in a write-only manner. $\mathcal{A}$ may read all outgoing communication tapes of the P_i and may also *corrupt* one or more parties. Upon corruption of P_i, $\mathcal{A}$ instantly gets a message containing P_i's complete past history (including states, head positions and tapes). $\mathcal{A}$ may from then on write arbitrary messages on its outgoing communication tape in the name of P_i, and all messages addressed to P_i are delivered to $\mathcal{A}$. Moreover, a message stating that P_i was corrupted is automatically delivered both to $\mathcal{Z}$ and to all $\mathcal{F}_i$.[3] Very briefly, the message transfer rules are: $\mathcal{Z}$ may talk to $\mathcal{A}$, $\mathcal{A}$ may talk to $\mathcal{Z}$, to the parties and the $\mathcal{F}_i$, the $\mathcal{F}_i$ may talk to $\mathcal{A}$ and to the parties, and the parties may talk to each other, to the $\mathcal{F}_i$ and to $\mathcal{A}$. A detailed description of a protocol run in the $\{\mathcal{F}_i\}$-hybrid model follows.

1. **Attack Phase:** Basically, this is a message-driven interaction between $\mathcal{Z}$, $\mathcal{A}$ and the $\mathcal{F}_i$, only $\mathcal{Z}$ and the $\mathcal{F}_i$ may not interact directly. First, $\mathcal{Z}$ is activated with local input "`round-start`". After $\mathcal{Z}$ has terminated its activation, all messages $\mathcal{Z}$ possibly wrote to $\mathcal{A}$ are delivered or, if $\mathcal{A}$ *blocked* messages from $\mathcal{Z}$, simply erased. If there was no such message, or if $\mathcal{Z}$ has halted, we consider the complete protocol run ended and the first cell on $\mathcal{Z}$'s local output tape is interpreted as $\mathcal{Z}$'s (binary) output. Otherwise, $\mathcal{A}$ is activated next or, if $\mathcal{A}$ blocked, $\mathcal{Z}$ is activated again. Once $\mathcal{A}$ has terminated its activation and written at least one message to $\mathcal{Z}$, all such messages are delivered and $\mathcal{Z}$ is activated again. However, if $\mathcal{A}$ wrote *no* message to $\mathcal{Z}$, the first $\mathcal{F}_i$ (in order of ITM identities) to which $\mathcal{A}$ wrote at least one message is activated with all messages addressed to it from $\mathcal{A}$ delivered. This includes messages sent from $\mathcal{A}$ to $\mathcal{F}_i$ in the name of a

[3] This is in analogy to [Can01, CLOS02] and ensures that $\mathcal{Z}$ can be used to compare two protocols using knowledge about the corruptions. Furthermore, it allows formulating strong functionalities $\mathcal{F}_i$.

corrupted party. Once this $\mathcal{F}_i$ terminates its activation, all messages it possibly wrote to $\mathcal{A}$ or the parties are delivered and $\mathcal{A}$ gets activated again. If $\mathcal{A}$ wrote messages neither to $\mathcal{Z}$ nor to an $\mathcal{F}_i$, all messages $\mathcal{A}$ wrote to the P_i are delivered and we proceed to the next phase.

2. **Party Computation:** All messages from any party P_i to another party P_j are delivered. Then, all non-corrupted P_i are activated in parallel. When all P_i have terminated their activations, messages they have written to the $\mathcal{F}_i$ or to $\mathcal{A}$ are delivered.

3. **Ideal Functionality Computation:** All $\mathcal{F}_i$ are activated in parallel with local input "`computation`". After the $\mathcal{F}_i$ have terminated their activations, messages the $\mathcal{F}_i$ have written to $\mathcal{A}$ or to the parties are delivered. Then, we start over with the attack phase.

Note that $\mathcal{A}$ cannot access communication of uncorrupted parties with ideal functionalities. However, our scheduling models a "rushing" adversary that may let corrupted parties send messages in dependance of the messages sent by honest parties *in the same round*. Since all ITMs terminate their current activation in polynomial time and both $\mathcal{Z}$ and $\mathcal{A}$ halt after a polynomial number of activations, a protocol run as described above ends after a polynomial number of steps. When we speak of a *protocol*, we mean a set of parties P_i running together as above. The output distribution of $\mathcal{Z}$ when run on security parameter k in the $\{\mathcal{F}_i\}$-hybrid model with protocol π and an adversary $\mathcal{A}$ is denoted by $\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k)$. Now we are ready to state the first part of our security definition, which relates two protocols.

Definition 1 *Let π be an n-party protocol formulated in the $\{\mathcal{F}_i\}$-hybrid model and let τ be an n-party protocol formulated in the $\{\mathcal{G}_j\}$-hybrid model. We say that π* securely realizes *τ (written $\pi \geq \tau$) iff for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ (called simulator) such that for every environment $\mathcal{Z}$ the function*

$$\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k) = 1) - \mathbf{P}(\mathcal{Z}(\{\mathcal{G}_j\}, \tau, \mathcal{S}, k) = 1)$$

is negligible[4] *in k. If this holds even with respect to $\mathcal{Z}$ which are* not *necessarily polynomially bounded (but still halt after polynomially many activations), we say that π securely realizes τ* unconditionally *(written $\pi \gtrsim \tau$).*

Note that for the unconditional case, we have chosen to allow an unbounded *environment*, but *not* an unbounded adversary. When considering unbounded adversaries, there is a practical need for an unbounded environment, as known proof techniques for composition don't seem to apply when only the adversary, but not the environment is

[4] $f : \mathbb{N} \to \mathbb{R}$ is called *negligible*, iff $\forall c \in \mathbb{N}\ \exists k_0 \in \mathbb{N}\ \forall k > k_0 : |f(k)| < k^{-c}$.

unbounded. On the other hand, when considering an unbounded environment, the security notion which allows for unbounded adversaries is *strictly weaker* than the same notion with polynomially bounded adversaries. Also, when allowing both unbounded adversaries and environments, the resulting security notion does *not* imply the seemingly weaker bounded security notion. (Our notion "$\gtrsim$", though, *does* imply "$\geq$" trivially.)

2.2 Ideal Model

In contrast to the real model, the ideal model reflects an idealization of a given protocol task. For simulation-based approaches, such an idealization is generally modelled as a single ideal functionality $\mathcal{F}$ which reads all input and secretly computes output accordingly, possibly in a reactive manner. This can be modelled in the $\{\mathcal{F}\}$-hybrid model with a set $D(\mathcal{F})$ of n identical *dummy parties*. (Here, the number n of parties is implicitly determined by the specification of $\mathcal{F}$.) Each dummy party relays its local input to $\mathcal{F}$ and locally outputs whatever it receives from $\mathcal{F}$. For polynomially boundedness, we assume that each dummy party terminates its activation after it has copied as much input to $\mathcal{F}$ as $\mathcal{F}$ can read in one activation (analogously for the output $\mathcal{F}$ may have written). Now we are ready to define the *$\mathcal{F}$-ideal model* as the $\{\mathcal{F}\}$-hybrid model with dummy parties $D(\mathcal{F})$ *and* an additional party computation step after the functionality computation step. This is to reflect immediate output generation. The output of an environment $\mathcal{Z}$ run on security parameter k in the $\mathcal{F}$-ideal model (as described above) and an adversary $\mathcal{A}$ will be denoted $\mathcal{Z}(\mathcal{F}, \mathcal{A}, k)$. The second part of our security definition allows us to specify when we consider a protocol a secure implementation of an ideal functionality.

Definition 2 *Let π be an n-party protocol formulated in the $\{\mathcal{F}_i\}$-hybrid model and let $\mathcal{F}$ be an n-party ideal functionality. We say that π* securely realizes *$\mathcal{F}$ (written $\pi \geq \mathcal{F}$) iff for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any $\mathcal{Z}$, the function*

$$\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k) = 1) - \mathbf{P}(\mathcal{Z}(\mathcal{F}, \mathcal{S}, k) = 1)$$

is negligible in k. If this holds even with respect to $\mathcal{Z}$ which are not *necessarily polynomially bounded (but still halt after polynomially many activations), we say that π securely realizes $\mathcal{F}$* unconditionally *(written $\pi \gtrsim \mathcal{F}$).*

From the definitions, it is clear that the relations "$\geq$" and "$\gtrsim$" are transitive relations on n-party protocols. Furthermore, $\pi \geq \tau$ in conjunction with $\tau \geq \mathcal{F}$ implies $\pi \geq \mathcal{F}$, analogously for "$\gtrsim$".

2.3 Composition

Due to space limitations, here we only note our security notion for protocols behaves well under universal composition. For details, cf. the full version [HMQ04].

2.4 Shell Constructs

In contrast to [Can01, CLOS02], our model does not allow the adversary to block messages, not even those sent from or to an ideal functionality. This allows formulating functionalities in a very specific way, but often it might be necessary for simulation to leave delivery—to a certain degree—up to the simulator. Therefore, we adapt the idea of [Bac03] to equip a machine with a "coat", or "shell", which manages message delivery to and from it. (In [Bac03], an "asynchronous coat" was used to investigate synchronously formulated machines in an asynchronous setting.)

Namely, for an ITM M (one should have in mind a party or an ideal functionality here), we define M's *asynchronization* $[M](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ (often denoted $[M]$ when the context is clear), with $p_{\mathsf{recv}}, p_{\mathsf{send}} \in \mathbb{Z}[x] \cup \{\infty\}$ and $\mathsf{clk} \in \{\mathsf{async}, \mathsf{sync}\}$. Internally, $[M]$ keeps a simulation of M and relays local in- and output as well as communication of M with $\mathcal{A}$ directly, with some exceptions explicitly noted below. Upon an incoming message m from a sender $S \neq \mathcal{A}$, $[M]$ writes a message "`request receive` j `from` S" to $\mathcal{A}$; here, j simply denotes a running number assigned by $[M]$. If $[M]$ receives a message "`allow receive` j" from $\mathcal{A}$, where j has been assigned before, $[M]$ relays the corresponding message m to the simulated M. Also, any message is automatically relayed to M after $p_{\mathsf{recv}}(k)$ activations of $[M]$—or, if M is an ideal functionality, after $p_{\mathsf{recv}}(k)$ local "`computation`" inputs. (There is *no* automatic message delivery if $p_{\mathsf{recv}} = \infty$.) Similarly, if M wants to send a message m to a recipient $R \neq \mathcal{A}$, $[M]$ first generates a "`request send` j `to` R" message to $\mathcal{A}$ and actually sends m to R upon an "`allow send` j" message from $\mathcal{A}$ or—whatever happens first—after $p_{\mathsf{send}}(k)$ rounds (i. e., $[M]$-activations, resp. local "`computation`" inputs). M is activated exactly once in *every* $[M]$-activation if $\mathsf{clk} = \mathsf{sync}$. Otherwise, M is activated only if one or more messages or local input are relayed to it in the respective $[M]$-activation; in that case, M is activated once for local input other than "`computation`", and each incoming message. The order is: local input first, then incoming messages ordered by sender identity. (Formally, we assume M only to process *interleaved* messages, as guaranteed by the delivery process in our modelling.)

$[M]$ halts when M has halted and all messages *from* M have actually been sent. Clearly, $[M]$ halts after a polynomial number of activations (resp., rounds in the case of ideal functionalities) if and only if M does so, $\mathsf{clk} = \mathsf{sync}$ and $p_{\mathsf{send}} \neq \infty$. To make $[M]$ polynomially bounded in each activation, we first mandate that $[M]$ reads in each activation only *one* local input and *one* message per sender $S \neq \mathcal{A}$, truncated to the maximum size which M is able to process in one activation. (We assume that by the time of construction of the "shell", the number n and the identities of parties and adversary are already fixed.) Additionally, at most one "`allow receive`" message from $\mathcal{A}$ per sender and at most one "`allow send`" message per recipient is processed; also, at most one message from $\mathcal{A}$ to M is read and truncated if "too long" for M. Processing of $\mathcal{A}$'s messages stops as soon as "too long" or "too many"

messages are encountered. Clearly, these restrictions limit the generality of $[M]$, yet in many cases—as, e. g., the case $M = \mathcal{F}_{\mathrm{SFE}}$ of an ideal functionality for secure function evaluation—this might be considered condonable.

By adding shells to the parties of a protocol, one can catch the notion of reliable or even asynchronous networks, the former which deliver messages after a polynomial number of steps. Furthermore, ideal functionalities may be formulated asynchronously in the first place, and later a shell may be added to leave message delivery factually up to the adversary (while it is possible to fix certain maximum latency times for messages sent to and from the functionality).

2.5 Relation to Other Models

Due to space limitations, we only note that a large class of protocols secure in the sense of [Can01] or [CLOS02] can be sensibly embedded into our model. This embedding is security-preserving. For details, cf. [HMQ04].

3 Global Bit Commitment is Impossible

In [BG90, GL91, CvdGT95], different protocols for realizing general secure function evaluation based only on oblivious transfer and broadcast were given—yet the notion of security used in these contributions is not simulatability-based; furthermore, these protocols can be aborted by a single party.

In this section we will show that in a reliable network with a broadcast functionality with guaranteed delivery, the primitive oblivious transfer (together with a broadcast primitive) is not complete as soon as three or more parties are involved. Namely, oblivious transfer and a broadcast channel will be proven not to be sufficient to implement a version of global bit commitment for which the output of the uncorrupted parties upon abort allows to identify who did not cooperate. Cryptographic primitives which upon abort allow to **identify** a corrupted party (which deviated from the protocol) are of special interest as they could be used to expell "disruptors" and replace their input by some default value until the protocol terminates successfully.

The constructions of [CLOS02] do not build up protocols from given primitives like oblivious transfer or broadcast, but allow to translate protocols which are secure with respect to passive adversaries into protocols which can tolerate an actively corrupted majority. The compiler in [CLOS02] is designed for an asynchronous model and no party or functionality can know if a message is missing due to deviation of a corrupted party from the protocol, or if this message is simply not delivered by the asynchronous network.

In contrast to that, the synchronous model of communication developed in this work reflects the properties of a completely reliable network. Intuitively, this makes it impossible for the adversary to let his actions appear as network problems. Hence,

in a setting with authenticated links an uncorrupted party or a functionality is always able to unambigiously identify the sender of a faulty message or a party who refuses to send a message as required in the protocol. This allows to define multi-party primitives which cannot be implemented with the primitives oblivious transfer and broadcast. The functionality introduced here is a version of the primitive *global bit commitment*, which allows to identify parties giving no proper input.

It is an interesting problem if a synchronous version of the compiler used in [CLOS02] allows to securely implement general functionalities with cheater identification in a reliable network.

Settings in which oblivious transfer or secure channels are not complete were considered in the literature before. In [FGMO01] a complete three party primitive (oblivious two cast) was presented which can implement all secure function evaluations in presence of a corrupted minority without using a broadcast channel. However, oblivious two cast cannot implement oblivious transfer if one drops the assumption of an uncorrupted majority.[5] In [MQ02], a quantum cryptographic protocol, which implements an unconditionally secure signature scheme along the line of [PW92] was presented. In the protocol of [MQ02], uncorrupted parties can decide from their view if the signer refused to sign a document or if some other party aborted the computation. As shown there, this is impossible when using only classical secure channels and a broadcast channel. The unpublished draft [MQI00] which inspired part of this work informally sketches a multi-party primitive *anonymous oblivious transfer* which is claimed to be more powerful than oblivious transfer.

Next we will describe a (single use per party) functionality $\mathcal{F}_{\mathrm{GCOM}}$, intended to formalize *global bit commitment*. Here, one party can be committed to all parties to the same bit. Moreover, using the delivery guarantee of our synchronous model, a party is either committed to all honest parties or all honest parties can deduce that P_i did not use the functionality $\mathcal{F}_{\mathrm{GCOM}}$. We will show that this functionality, which intuitively allows to detect misuse, cannot be securely realized in the $\{[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}], [\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]\}$-hybrid model (cf. Appendix A for a description of the broadcast functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$ and the oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$).

Theorem 3 *In the $\{[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}], [\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]\}$-hybrid model (for arbitrary but fixed choices of shell parameters and polynomials $p_{\mathrm{OT}}(k), p_{\mathrm{BC}}(k), \ell(k)$), there is* no *functionality $[\mathcal{F}_{\mathrm{GCOM}}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ (with $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ and $\mathsf{clk} \in \{\mathsf{async}, \mathsf{sync}\}$) which can be securely realized for $n \geq 3$ parties.*

Proof. A proof can be found in the full version [HMQ04]. We provide a very rough sketch: A party P_1 commits to P_2, P_3 using a protocol π assumed to realise $[\mathcal{F}_{\mathrm{GCOM}}]$. But either P_1 or P_2 blocks all point-to-point communication between P_1 and P_2, especially the oblivious transfer channel. As P_3 can from its view not decide who is

[5] Then, a collusion of all parties but the sender of an oblivious cast can reconstruct everything that was sent.

Functionality $\mathcal{F}_{\mathrm{GCOM}}$

$\mathcal{F}_{\mathrm{GCOM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. Messages not covered here are simply ignored.

- **Commit**: When receiving "(commit, b)" from a party P_i with $b \in \{0,1\}$, store the tuple (P_i, b) and send "(receipt, P_i)" to all parties and to the adversary. Ignore any future "commit" messages from this party P_i as well as all messages not of the form "(commit, P_i, b)".
- **Reveal**: When receiving a message "(reveal)" from a party P_i: If a tuple (P_i, b) was previously recorded, then send the message "(reveal, P_i, b)" to all parties and to $\mathcal{S}$. Otherwise, ignore.

Figure 1: Functionality $\mathcal{F}_{\mathrm{GCOM}}$

cheating, the protocol π must produce a valid commitment without using any point-to-point communication between P_1 and P_2. A variant of the attack of [CF01] on universally composable bit commitment can be mounted in this situation. Thus, π must be insecure. □

Here a remark is in place: functionalities like $\mathcal{F}_{\mathrm{GCOM}}$ and its shell-equipped variant $[\mathcal{F}_{\mathrm{GCOM}}]$ considered above may be hard to realize for trivial reasons, since real and ideal model must be indistinguishable even if both respective adversaries have already halted. Also in some cases, it might be more suitable to restrict to functionalities which halt after a polynomial number of *rounds*. However, the result of Theorem 3 remains true when restricting to explicitly "round-bounded" functionalities $[\mathcal{F}_{\mathrm{GCOM}}]$, $[\mathcal{F}_{\mathrm{OT}} \| p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)} \| p_{\mathrm{BC}}]$, which halt after a polynomial number of *rounds*. Namely, the number of rounds each of the environments constructed in the proof of Theorem 3 runs depends *only* on the choices of the shell parameters p_{recv} and p_{send} of $[\mathcal{F}_{\mathrm{GCOM}}]$, but *not* on π. In fact, the proof holds literally for "round-bounded" functionalities $[\mathcal{F}_{\mathrm{GCOM}}]$ which halt after $2 \cdot (p_{\mathsf{recv}} + p_{\mathsf{send}} + 1)$ rounds. Moreover, any protocol π realizing a functionality $[\mathcal{F}_{\mathrm{GCOM}}]$ in a hybrid model with round-bounded $[\mathcal{F}_{\mathrm{OT}} \| p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)} \| p_{\mathrm{BC}}]$ implies a protocol π' which does so in a hybrid model with unbounded (regarding the number of rounds) $[\mathcal{F}_{\mathrm{OT}} \| p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)} \| p_{\mathrm{BC}}]$. Summarizing, the theorem holds also when restricting to round-bounded ideal functionalities.

If one allows computational assumptions as well as use of a common reference string (in form of an ideal functionality with guaranteed delivery), the functionality $\mathcal{F}_{\mathrm{GCOM}}$ *may* become realizable even in our synchronous network by a synchronous version of a protocol of [CLOS02] using a broadcast channel with guaranteed delivery. For this, one could use a non-interactive bit commitment and broadcast the com-

mitment to all parties. As the broadcast functionality guarantees delivery this might realize a guaranteed-delivery version of $[\mathcal{F}_{\mathrm{GCOM}}]$. This *would* in particular imply that such a common reference string functionality cannot be realized by oblivious transfer and broadcast functionalities alone.

4 Conclusions and Open Questions

In this contribution a synchronous model of security was developed as an abstraction of a reliable network with guaranteed delivery. Our model allows for universal composability.

In the synchronous model a shell concept $[M](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ was introduced for ideal functionalities as well as for parties. A shell allows to delay incoming messages to M up to p_{recv} rounds and outgoing messages from M up to p_{send}. The parameter clk can be set to sync to have the machine M in the shell activated in each round even if no new message is to be received. For $\mathsf{clk} = \mathsf{async}$ the machine M in the shell is only activated if a message is delivered to it.

Also, shell constructs can be used for asynchronous specification of ideal functionalities. In particular, completely asynchronous executions of protocols can be modelled. It was proven that a large class of secure realizations in the setting of [Can01, CLOS02] can be transferred to secure realizations in our model if the shell parameters are set accordingly.

This work showed that security in our synchronous model with $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ is not completely covered by the the asynchronous definitions of [Can01, CLOS02]. A variant of global bit commitment was given as an example of a functionality which allows the identification of a party who did not give input to the protocol. The functionalities oblivious transfer and broadcast do not suffice to securely realize this global bit commitment in our synchronous model. This especially implies that the primitives oblivious transfer and broadcast are not complete in the synchronous model presented here.

We also raised questions with respect to security in reliable networks like the one considered here. Does a synchronous version of the compiler of [CLOS02] yield general realizations of ideal functionalities which allow cheater identification? Which ideal functionalities are complete for the synchronous model presented here?

References

[Bac03] Michael Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. In Roberto Amadio and Denis Lugiez, editors, *Concurrency Theory, Proceedings of CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 350–365. Springer-Verlag, 2003.

[BG90] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In Gilles Brassard, editor, *Advances in Cryptology, Proceedings of CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 589–590. Springer-Verlag, 1990.

[BPSW02] Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *15th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2002*, pages 160–174. IEEE Computer Society, 2002.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *Advances in Cryptology, Proceedings of CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer-Verlag, 1995.

[FGMO01] Matthias Fitzi, Juan A. Garay, Ueli Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 80–100. Springer-Verlag, 2001.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A.

Vanstone, editors, *Advances in Cryptology, Proceedings of CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer-Verlag, 1991.

[HMQ04] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. IACR ePrint Archive, January 2004.

[HMQS03] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On modeling IND-CCA security in cryptographic protocols. IACR ePrint Archive, February 2003.

[MQ02] Jörn Müller-Quade. Quantum pseudosignatures. *Journal of Modern Optics*, 49(8):1269–1276, July 2002.

[MQI00] Jörn Müller-Quade and Hideki Imai. Anonymous oblivious transfer. lanl.arXiv.org ePrint Archive, December 2000.

[PW92] Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In Alain Finkel and Matthias Jantzen, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings of STACS 92*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer-Verlag, 1992. Extended abstract.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security, Proceedings of CCS 2000*, pages 245–254. ACM Press, 2000.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2001*, pages 184–200. IEEE Computer Society, 2001.

A Functionalities

Here we describe the oblivious transfer and broadcast functionalities used in Section 3.

Functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$

$\mathcal{F}_{\mathrm{BC}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathrm{BC}}$ is parameterized with a polynomial $\ell(k) \in \mathbb{Z}[k]$ bounding the length of broadcasted messages. All messages not covered here are ignored.

- When receiving $m \in \{0,1\}^l$ (with $1 \leq l \leq \ell(k)$) from a party P_i, send m to all parties $P_1, \ldots, P_n$ and to $\mathcal{S}$. Process at most one message per party.

Figure 2: The broadcast functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$

Functionality $\mathcal{F}_{\mathrm{OT}}$

$\mathcal{F}_{\mathrm{OT}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. All messages not covered here are ignored.

- When receiving "`(sender,` P_i`,` P_j`,` x_1`,` x_2`)`" from P_i, such that $x_1, x_2 \in \{0,1\}$ and there has been no such message before for this value of (P_i, P_j), record this tuple and send "`(sender,` P_i`,` P_j`)`" to $\mathcal{S}$.
- When receiving "`(receiver,` P_i`,` P_j`,` m`)`" from P_j, such that $m \in \{1,2\}$ and there has been no such message before for this value of (P_i, P_j), record this tuple and send "`(receiver,` P_i`,` P_j`)`" to $\mathcal{S}$.
- If at any time, there are tuples "`(sender,` P_i`,` P_j`,` x_1`,` x_2`)`" *and* "`(receiver,` P_i`,` P_j`,` m`)`" for some P_i, P_j recorded, send the message "`(transferred,` P_i`,` P_j`,` x_m`)`" to P_j and the message "`(transferred,` P_i`,` P_j`)`" to $\mathcal{S}$.

Figure 3: The oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$

Design of a CIL Connector to SPIN

Li Yongjian[1,2], Xue Rui[2]
Institute for Software, Chinese Academy of Sciences[1]
The State Key Laboratory of Information Security, Beijing, China 100080[2]
Beijing, China ⋆

Abstract

The CAPSL Integrated Protocol Environment effort aims at providing an intuitive and expressive language for specifying cryptographic authentication and key distribution protocols and supporting interfaces to various analysis tools. The CAPSL Intermediate Language CIL has been designed with the emphasis on simplifying translators from CIL to other analysis tools. In this paper we describe the design of a CIL-to-Spin connector. We describe how CIL concepts are translated into Spin and propose a general method to model the behaviors of honest principals and the intruder. Based on the method, a prototype connector has been implemented in Gentle, which an automatically translate CIL specification to promela code and LTL formula, thus greatly simplifying the modelling and analysis process.

1 Introduction

A cryptographic protocol is a series of carefully defined messages, often encrypted, between two or more participants designed so that when it is complete, a specified goal like authentication and secrecy has been achieved, even in the presence of an intruder who can perform malicious acts. However, the design of these protocols is error prone, and incorrectly designed protocols may become ideal entry points for various attacks. Over the last few years, formal methods have proven helpful for both cryptographic protocol design and analysis. The use of formal languages supports the rigorous formalization of protocol models and their properties. Moreover, they also provide a basis for using tools such as model checkers and theorem-provers to prove protocols correct or uncover security flaws. Unfortunately, most formal analysis tools

⋆Li and Xue are supported by Fundamental Research Projects in Institute of Software, Chinese Academy of Sciences under contract No. cxk25056 and National Hi-Tech Program in China under contract No. 2002aa144050, and NSFC project under Grant No.60173020 and Grant No.60373048.The email address of the contacting author is lyj238@ios.ac.cn

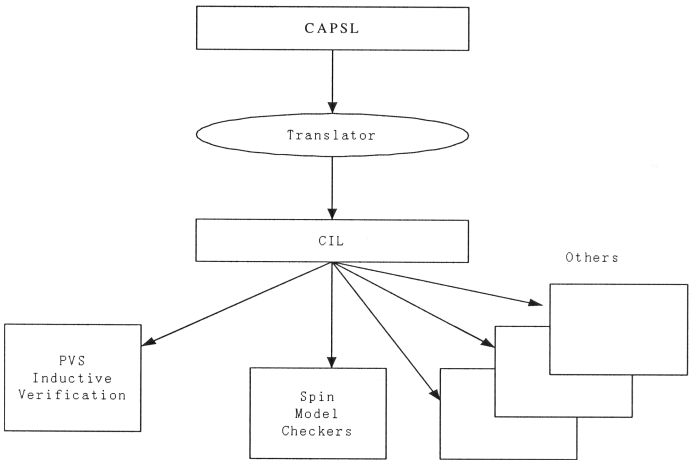


Figure 1: CAPSL translation Plan

require protocols to be expressed in a tool-specific language, with specific conventions peculiar to cryptographic protocol analysis. The language may be low-level and based on the general-purpose language that the tool was implemented in or designed to support.

The CAPSL Integrated Protocol Environment is an effort to provide an intuitive and expressive language for specifying cryptographic authentication and key distribution protocols and support interfaces to various analysis tools [1]. The general architecture of the CAPSL Integrated Protocol Environment is illustrated in Figure 1. In this environment, CAPSL, a Common Authentication Protocol Specification Language, has been designed as a high-level language for security protocols. The core feature of it is a message list like the ones that are often used to present cryptographic protocols in articles and textbooks. Then CAPSL is translated into CIL [2], a CAPSL Intermediate Language, expressing state transition with term-rewriting rules. It turns out that CIL is close to the state-transition representation used by almost theorem-proving and model-checking tools. CIL is the core of the strategy by which CAPSL can be adapted for use by various protocol analysis tools. It serves two purposes: to define the semantics of CAPSL, and to act as an interface through which protocols specified in CAPSL can be analyzed using different tools. The CAPSL Integrated Environment provides parser and type checkers for CAPSL and CIL and a CAPSL-to-CIL translator.

Connectors are being written to adapt CIL to supply input to a variety of formal analysis tools, including PVS for inductive verification and Maude for model-checking, and so on. For each connector one has to solve several issues in order to deliver specifications that fulfill the following two characteristics: (1) syntactical and semantic correctness. One has to decide which CIL pieces are necessary for the trans-

lation into the targeted tool, and how they are translated correctly in both syntax and semantics. (2) Practicability. Executability, non-determinism, and performance aspects of the transformed specification have to be taken into account for efficient and practical analysis. The connector algorithm might need to perform alternations and optimization to meet these criteria.

In this paper, we describe the design and optimization solutions for a CIL connector to Spin [3], which is the most powerful general purpose model checker and has achieved great success in the area of verification of software and communication protocols. In spite of many work that has been done in this area, there is few successful works at using spin to verify cryptographic protocols, and we still think it important for us to implement a tool like this. There are the following reasons to explain our motivation. Firstly, there is a lack of free tools aiding us to analyze security protocols, especially for those free researchers without commercial support in university. For example, FDR[4], NRL[5], Athena[6] can only be obtained after we pay expensive fees for licenses. But Spin is open and free, and powerful. By our work, we show that we can produce a free tool based on Spin as much powerful as those above if we use some additional encoding techniques. Secondly, by the detail work to model security protocols, we will know the advantages and limitation in applying classical model checking technique to analyze security protocols. And this will help us to put model checking technique in a proper place in this area. Thirdly, the CAPSL Integrated Protocol Environment provides a unified framework to both provide an intuitive and expressive language for specifying cryptographic protocols and support interfaces to various analysis tools. We believe that it is a significant effort and it may help us to combine different techniques to analyze security protocols. In particular, implementing a connector from CIL to Spin can help us to understand details of the technique which CIL used to formalize all semantic aspects of security protocols.

In the following we describe the design of a CIL-to-Spin connector that has been implemented in Gentle[7]. Such a connector automatically translates CIL specifications into Promela code and LTL formula which are suitable for model checking in Spin. The remainder of this paper is organized as follows. Section 2 informally introduces the syntax and semantics of CAPSL and CIL. Section 3 specifies the way the connector builds the Promela model and formalizes the security property. Section 4 is related work and conclusion. Appendix gives the detail about the experiments results which the connector has done for NSPK protocol.

2 CAPSL and CIL

A CAPSL specification is made up of three kinds of subspecifications: *type*, *protocol*, and *environment*, usually in that order. The CAPSL specification of Needham-Schroeder public key (abbr. as NSPK) protocol is listed in figure 2.

Type specifications define cryptographic operators and other functions axiomat-

```
TYPES PKUser: Principal;
FUNCTIONS
pk(PKUser): Pkey;
sk(PKUser): Pkey, PRIVATE,
                CRYPTO;
VARIABLES
X: Field;
U: PKUser;
AXIOMS
{{X}pk(U)}sk(U) = X;
{{X}sk(U)}pk(U) = X;
INVERT {X}pk(U): X | sk(U);
INVERT {X}sk(U): X | pk(U);
END;
```

(a)Type specification

```
PROTOCOL NSPK;
 VARIABLES
 A, B: PKUser;
 Na, Nb: Nonce, CRYPTO;
ASSUMPTIONS
   HOLDS A: B;
MESSAGES
 A -> B: {Na, A}pk(B);
 B -> A: {Na, Nb}pk(A);
 A -> B: {Nb}pk(B);
GOALS
 SECRET     Nb: B, A;
 PRECEDES B : A | Na
END;
```

(b) Protocol specification

```
ENVIRONMENT Test1;
  IMPORTS   NSPK;
  CONSTANTS
   Alice, Bob: PKUser;
   Intruder: PKUser, EXPOSED;
  AGENT S12 HOLDS
    A = Alice;      Na=S12Na;
     B = Intruder; Nb=S12Nb;
  AGENT S13 HOLDS
     B = Bob;         Nb=S13Nb;
  ORDER (S12||S13)

END;
```

(c)Environment specification

Figure 2: NSPK CAPSL specification

ically, and are also used to define different types of principals. There is a standard CAPSL "Prerelude" defining several commonly used operators, such as the familiar abstract symmetric-key and public-key encryption. For simple protocols, the prelude contains enough type specifications for them; but for more complex protocols, we may need to provide additional datatype specifications.

A protocol specification includes a message list. The message list is preceded by programming language-like type declarations for protocol variables and assumptions about initial conditions. It is followed by a list of security goals. In figure 2 (b), there is the protocol specification of NSPK protocol.

There are three messages in this protocol. In the first message, principal A sends the encrypted concatenation of a newly generated nonce *Na* and its address to another principal B. A uses B's public key for encryption. If the secret key of B is not compromised then only B can decrypt this message. Principal B replies by sending the nonce *Na* and a newly generated nonce *Nb* back to agent A. In the third message principal A acknowledges the receipt of nonce *Nb*. There are two types of goals specified: secrecy and agreement goals. Nonce *Nb* should remain secret to the agents in roles A and B. A secrecy goal is violated, if an intruder knows a nonce which is used by an honest agent in a session with another honest agent. The agreement goal uses the keyword PRECEDES. PRECEDES $B : A \mid Na$ means that if a B-agent reaches its final state, then there exists an A-agent that holds the same values as the B-agent on

the variables A, B, and *Na*.

An environment specification module contains specific information for model checking, which is relevant to setting up protocol sessions to be checked, defining the agents taking part in the actual system, the actual parameters to be used, and the intruder's initial knowledge. Agents are specified here by assigning constants to protocol variables whose values are initially held by that agent. The first assignment must be to the owner variable. Nonces could be assigned values here or not, depending on the needs of the analysis tool.

CIL [2] is a term-rewriting based notation that is capable of expressing all aspects of security protocols in a succinct, precise, and uniform way. The CIL representation of a CAPSL protocol, automatically generated by the CAPSL-to-CIL translator, consists of symbol table, slot table, axioms, localized assumptions, protocol rewrite rules, localized goals, and environment information. The compiled CIL file of the NSPK protocol specification in figure 2 is shown in Appendix A.

3 CIL-Spin Connector

The CIL-Spin connector takes CIL specifications as described in Section 2, and produces an output file, containing Promela code to model protocol agents and LTL formula to model security protocols, which are suitable for model checking using Spin. In this section, we give the details of the way the connector works protocol in Spin, and explain how the output files can work. We assume some familiarity with Spin.

The first key issues arise in keeping the data of facts involved both finite and manageable for Spin. In order to keep the set of facts finite, we only consider the facts possibly used by the intruder to launch attacks in the finite protocol sessions defined in the environment specification. Those facts can be computed automatically by our connector according to the protocol specification. Even with such restriction on the facts possibly used in the formal model, but we still find that these facts are structural (or recursively defined) and difficult to handle for Spin. As a general purpose model checker, Spin does not directly support the recursively defined symbolic data types, so it is not directly manageable for Spin, and we need special technique to encode them.

Our Promela model is based on a datatype—called *Fact_type*—representing the set of the facts that can be formed from the atoms by sequencing, encryption, application of hash functions, and so on. For the NSPK specification in figure 2, this is defined by:

Principal:: =Alice| Bob| Intruder Nonce::= NaS12|NaS13|NbS12|NbS13

Fact = Principal| Nonce|pk(Principal)|sk(Principal)cat(Fact,Fact)|
pke(Pk(Principal), Fact)

where Principal and Nonce are extracted from environment parts, and *pk*, *sk*, *cat*, *pke* operator, which are needed in NSPK, are extracted from the symbol table and message formats in the protocol rewrite rules of CIL input. Obviously, the datatype *Fact* is still infinite. To keep data of facts finite used for model-checking, the connec-

tor only considered those used by the intruder to launch attacks in the finite protocol sessions, which can be computed by message formats defined by the protocol rewrite rules and assignments in the environment specification. In the following, we assume that *Facts* is the set of all the facts defined by the datatype. Obviously, we only need a finite subset of *Facts*. We assume that *PVars* is the set of protocol variables defined in the symbol table, and *Msg_Formats* is the set of message formats used in the protocol rewrite rules, *Atoms* is the set of all the atom facts defined by the datatype Fact, *Intruder_InitialKnowledge* is a set of all the facts known to the intruder initially, which can be computed by the parts of environment specification and symbol table. And an assignment A: *PVars*→ *Atoms* is a function that assign constant values to protocol variables, an interpretation is a function I:*Msg_Formats*→(*Assignment* →*Facts*) that interpret the meaning of a message format into an instance of fact under some assignment.

Our connector will compute the set of all the possible assignments defined in the environment specification — denoted by *All_Assignments*, and the set of all the instances of message formats under all assignments — denoted by *All_Msg_Insts*, the set of all the subfacts of *All_Msg_Insts* plus *Intruder_InitialKnowledge* – denoted by *All_Facts*. For the NSPK in Figure 2, this set has 41 members.

Obviously, the facts in the set *All_Facts* are still recursively defined, it is not directly manageable for Spin. In order to manipulate them by Spin, we adopt the following encoding strategy:

- A meta type is created for each atomic fact, each construct, each kind of protocol message.
- All the facts possibly used in a sample protocol instance are represented by elements of a '*Facts*'array. And the '*Facts'* array is defined in figure 3. In fact, we have represented all the above facts by a binary tree encoded in the *Facts*'array.

where *total_fact* is the constant to define the total number of the set *All_Facts*.

```
#define total_fact
typedef Tfact{
mtype oper; int operand1; int operand2;}
TFact Facts[total_fact]
```

Figure 3 the data types of facts

Each element of the array "*Facts*" represents a fact, and the index of an element represents the unique encoding number of the corresponding fact. In the definition of data type "*TFact*", the field "*oper*" is used to indicate the type of the corresponding fact. For an atomic fact, the field "*oper*"is assigned to the meta type created for the

atomic fact; whereas for a compound fact, it is assigned to the meta type for the corresponding construct of the compound fact. And the fields "*operand1*" and "*operand2*" are used to record the encoding number of the sub_facts of the compound fact. For an atomic fact, the two fields are both assigned to the special value -1.

To encode the facts possibly used in the NSPK protocol specification, we define

mtype={Bob,Alice,Intruder,S12Na,S13Na,S13Nb,S12Nb,cat,ped,pk,sk,Msg0,
Msg1,Msg2}

#define total_fact 41

TFact Facts[total_fact]

where *Msg01, Msg1, Msg2* are corresponding to 3 kinds of the protocol messages.

The full content of the array *Facts* and their interpretation can be seen in Appendix, here we just analyze a fragment of it. *Facts*[2] and *Facts*[7] are used to represent the atom fact *S12Na*, *Alice*, and *Fact*[25] represent the composite fact *cat*(*S*12*Na*,*Alice*), they are shown in figure 4. Our connector will generate the definitions of all meta types defined for atom facts, and operators, message kinds, and the type definition for TFact, and the fact array *Facts* which holds all the encoding number of the facts in the set *All_Facts*.

In our Promela model, all the processes communicate with each other through a shared channel *comm*, which is defined as following:

chan comm = [0] of {mtype, short};

For the NSPK protocol instance in Figure 2, if a process sends message 2 whose encoding number is *Msgno*, then the statement *comm!Msg*2(*Msgno*) will be used; and *comm?Msg*2(*Msgno*) will be used to receive message 2 from the channel.

Now it is the step to define the agents acting various roles in the protocol specification. Typically a cryptographic protocol involves several honest processes (often two: an initiator and a responder) and perhaps a server that performs some service such as key generation, translation or certification. In our model, such processes must be parameterized with the data that may change from session to session and from instance to instance. And their behaviors include three aspects: (1)the messages exchange described by the protocol; (2) actions that the processes do with the protocol messages such as generation of keys and nonces, encryption and decryption, and deciding when a message that has been received is correct so as to allow the protocol to continue. (3)some additional signal events to indicate the agent's beliefs, which are relevant to verifying of security property. Here we just introduce (1), and (2). And (3) will be shown later in this section.

Our connector will generate a proctype definition for every role defined in the protocol specification. And it includes three parts: (1) parameters definition; (2) definition of local variables to store the atoms in the received messages; (3)statements to represent actions of this role.

For the NSPK protocol, the connector will define proctype *roleA* for the initiator A as follows:

proctype roleA(mtype Na;mtype A;mtype B)

```
{ short Msgno; TFact Msg;
mtype Loc_Na;mtype Loc_A;mtype Loc_B;
mtype Loc_Nb; mtype Memory_Nb;
atomic{
cons_Msg0(B,Na,A,Msg,Msgno);
comm!Msg0(Msgno); };
atomic{
comm?Msg1(Msgno);
destruct_Msg1(Loc_A,Loc_Na,Loc_Nb,Msg,Msgno);
Memory_Nb=Loc_Nb;
A==Loc_A;
Na==Loc_Na;} ;
atomic{
cons_Msg2(B,Memory_Nb,Msg,Msgno);
comm!Msg2(Msgno); }
}
```

The knowledge of the intruder is represented by the facts known to the intruder. In order to model the knowledge held by the intruder, we use a Boolean array "*Spy_know*" with the same length as the array "*Facts*"to represent the knowledge of the intruder. Each element of the array "*Spy_know*" is related to a unique fact, and the value of it is used to indicate whether the intruder has known the corresponding fact ("1"is set if the intruder has known the fact, otherwise "0"is set). So the knowledge of the intruder is represented as following:

bit Spy_Known[total_fact]

The initial value of the array *Spy_Known* represents the initial knowledge of the intruder, and the value of the array *Spy_Known* is changing during the run of protocol sessions because the intruder may infer new facts according to the inference rules and the messages intercepted from the network. The connector will generate the definition the array *Spy_Known* and the initialization code for the array.

Now we discuss the representation of the rules of deduction for the intruder. In essence, a deduction is a pair $<$*Conclusion, Assumptions*$>$ where *Assumptions* is a finite non-empty subset of *All_Facts*, which are not in the initial knowledge of the intruder, and *Conclusion* is an element in *All_Facts*. Informally speaking, the meaning of a pair $<$*Conclusion, Assumptions*$>$ is that *Conclusion* can be derived if all the facts in the *Assumptions* has been obtained. Note that we always assume that the intersection between *Assumptions* and the initial knowledge of the intruder is empty because the initial knowledge of the intruder are already known and need not been obtained again. More formally,

$Deductions \subseteq \{ < Conclusion, Assumptions > \mid Intruder\_InitialKnowledge \cap Assumptions = \emptyset, .\emptyset. \subset Assumptions. \subseteq All\_Facts , Conclusion. \in All\_Facts \}$

where the set *Intruder_InitialKnowledge* is the initial knowledge of the intruder;

Note that all the deductions used by the intruder in the sessions defined in protocol environment specification can be statically computed by our connector.

For all the deductions, we encode them in our Promela model as following:

typedef TDeduction{

bit Deduced; int Conclusion;int $Assumption_1$,$Assumption_2$;};

TDeduction Deductions[DeductionNum];

where *DeductionNum* is the total number of all the deductions involved in the protocol sessions. And in the definition of the type *TDeduction*,the field *Deduced* shows whether the rule has been used, and fields *Conclusion*, $Assumption_1$, $Assumption_2$ are the encoding number of facts of the conclusion and *assumptions* respectively.

Having defined data structures used by the intruder, we can define the algorithm to model the behaviors of the intruder. Our connector defines proctype PI which describes the behaviors of a general intruder as follows:

```
proctype PI ()
{ TFact M; short Mno,headAssumption; short A_q_head=0, A_q_tail=0;
short D_q_tail=0, index=0;TDeduction CurDeduction;
Assumptionqueue[total_fact];
do :: atomic { comm?Msg0(Mno);   addKnowledge(M,Mno);}
  ::atomic  { sel_Msg(Msg0,Mno);  comm!Msg0(Mno);}
  ::atomic { comm?Msg1(Mno);   addKnowledge(M,Mno);}
  ::atomic { sel_Msg(Msg1,Mno);   comm!Msg1(Mno);}
  ::atomic { comm?Msg2(Mno);    addKnowledge(M,Mno);}
  ::atomic { sel_Msg(Msg2,Mno);   comm!Msg2(Mno);}
od;}
```

The body of the intruder process is a never ending loop, and each branch of the repetition is either a message input action followed by deductions to infer new facts, or selecting a message from facts in its knowledge possession and sending the message.

Macro addKnowledge(M,Mno) is to model the inference system of the intruder. In essence inferences are to derive new facts from the facts known to the intruder to date under the deduction rules. New facts just derived will be used as assumptions again to infer new facts, and this procedure do not stop until no new fact is inferred. If we define a function $close(S)$ which calculates all the facts that are buildable from S under the deduction rules.

close(S)=let S'=$\{f \quad | \quad X \subseteq S, < X, f > \in$ *Deductions*$\}$ in if $S' \subseteq S$ then S
else *close(S'*$\cup S$)

where S is the knowledge of the intruder to date, and the intruder intercept a message M from the system, then the knowledge of the intruder will evolve to *close*($\{M\} \cup S$). Macro *addKnowledge*(M) will just model all the deductions which augment the intruder's knowledge from S to *close*($\{M\} \cup S$). Due to limitation in space, here we just introduce *addKnowledge*(M) in an algorithm level, its Promela code can be obtained in Appendix.

In the following algorithm, we assume that:

1. *Spy_Known*$\subseteq$*Facts*: the set of the facts known to the intruder;
2. *Unused_Deductions* $\subseteq$ *Deductions*: deductions which have not been used by the intruder;
3. *Assumption_Queue* : a queue that are used to store new facts that has just been derived and will be used to deduce new facts as assumptions;

```
Function addKnowledge(Fact M)
{Assumption_Queue:=null; put(Assumption_Queue, M);
Spy_Known:= Spy_Known .∪{M};
repeat
  Head_Fact:=get(Assumption_Queue);/*get the head out of Assumption_ Queue
  and assign it to Head_Fact*/
  Deduces_HF:={< Conclusion,Assumptions> | < Conclusion,Assumptions>∈
               Unused_Deductions, Conclusion∉Spy_Known,
               Head_Fact∈ Assumptions, Assumptions⊆ Spy_Known };
  Unused_Deductions := Unused_Deductions- Deduces_HF;
  for all <Conclusion, Assumptions>∈ Deduces_HF {
      put(Assumption_Queue, Conclusion);
      Spy_Known:=Spy_Known ∪ {Conclusion};}
until (Assumption_Queue<>null)}
```

On the other side, macro *sel_Msg*(*Msgk,Mno*) randomly selects an message of the kind *Msgk*, which is known to the intruder, and assign the its encoding number to *Mno*. The detail of Macro *sel_Msg*(*Msgk,Mno*) can also be seen in Appendix.

Our connector will generate all the data structure and promela code of the intruder for the specified protocol system. In fact, most of this procedure is rather common for all kinds of protocols due to our encoding style of facts and the ability of the intruder for cryptographical protocols. The difference between the intruders in the different protocol specification lies in the initial_knowledge and deduction rules mastered by the intruder, and the kinds of the messages the intruder can sent and receive. But the type definition and the code to model deduction system of the intruder is the same.

Now we can define the checked system which is defined in environment specification. It is simply specified by initializing facts, deductions, and introducing a process instantiation statement in the init process for each principal acting some role of the system respectively.

For NSPK in Figure 2, the connector defines the following init process:

```
init {..... (/*the initialization of array Facts,Deductions,Spy_ Known, and so on*/)
atomic { if atomic {run roleA(S12Na,Alice,Intruder);
            run roleB(S13Nb,Bob);
            run PI();}}
```

In this paper, we concentrate our attention on secrecy and authentication properties, which are dealt with by LTL now. These are safety properties, in the sense that

they require that bad things should not happen – that a data item is not leaked and an agent does not incorrectly accept the identity of another agent. In order to define secrecy properties, we need to introduce additional informal information into protocol descriptions to enable a description of what is expected of the system at particular points during a run of the protocol. This information might be thought of as capturing the intention of the designer of the protocol.

Secrecy goal like loc(nodes(node(roleA,3),node(roleB,3)),secret(Nb,ids(B,A))) will be expressed as the requirement that if honest principals a, b acting roles *roleA* , *roleB* finished a session, then the actual value of the protocol nonce variable *Nb* should not be detected by the intruder provided that both a and b are honest. Our connector always assumed that the protocol variable *Nb* is a fact that originated by a unique principal of a or b, and the secrecy should also be self-known to its originator. If a principals acting role *RoleA* originates a message containing secrecy m, then we explictly introduce actions of claiming secrecy in the proctype definition of *RoleA*. In detail, we adopt the following strategy to check secrecy properties:

- If *RoleA* originates a message containing secrecy m, i.e., secret(m,ids(..., A, ...))) is defined in the protocol goal specification, and m is originated by principals acting as role A, then the connector allocates an array *RoleA_Claim_secret_m*[*totol_inst_of_RoleA*] to store all the encoding number of the possible instances of the protocol variable m claimed by the honest principals acting the role *RoleA*, where *totol_inst_of_RoleA* is the total number of the honest principals acting the role. The intial value of each element in the array *RoleA_Claim_secret_m* is set to -1.

- The connector assigns a unique index for every principal acting the role *RoleA*. And this index will be computed by a special promela macro code *index_RoleA_Inst* according to the identity of the principal. The macro code *index_RoleA__Inst* is also generated by our connector.

- In the definition of the proctype *RoleA*, the connector inserts codes to explicitly expressing actions of *RoleA* claiming secrecy. These codes are introduced just after *RoleA* originates a message including the protocol variable m. For the secret goal ,secret(Nb,ids(B,A)) in NSPK protocol, the connector will insert the following codes after the statement of sending message {na,nb}pk{A} in the proctype RoleB.

```
if
::(B!=I)->{look_up_atom(Nb,Msg,Msgno);
index_Role_Inst(B, BIndex);
RoleB_Claim_secret_Nb[BIndex]=Msgno ; }
::else->skip;
fi;
```

The property which the role *RoleB* cliams that *Nb* should be secret can be expressed in the following LTL formalism:

[](!(RoleB_Claim_secret_Nb[0]==-1)

-> Spy_Known[RoleB_Claim_ secret_Nb[0]]==0)&&.....

!(RoleB_Claim_secret_Nb[totol_inst_of_RoleB-1]==-1)

->Spy_Known[RoleB_Claim_secret_Nb[totol_inst_of_RoleB-1]]==0))

Authentication properties like loc(nodes(node(roleA,3),node(roleB,3)), precedes(B,A,ids(Na))) will be expressed as the requirement that if an honest principal $b$acting the role *roleB* finished a session, then there exists an honest principal a acting role *roleA* holds the same values as b on the protocol variables A, B, and *Na*. Here we also assume that the nonce *Na* must be originated by a. In order to verify this correspondence,

- the connector allocates an array *Commit_on_Na*[*totol_inst_of_RoleB, totol_inst_of_RoleA*][1] to store all the encoding number of the possible instances of the protocol variable *Na* which are commited by$b$ acting the role *RoleB, i.e., Commit_on_Na*[*BIndex, AIndex*] stores the encoding number of instance of *Na* held by the honest principals b acting the role *RoleB* , and$a$is the principal who is thought as the communication party by b in the session, where *BIndex, AIndex* are the unique index of b and a respectively. The initial value of all the elements in this array are all set to -1. And the connector will insert the following codes into the body of proctype *RoleB* after the statement of the action of receiving the message that contains *Na* in the first time.

```
if
::(B!=Intruder)->{ index_RoleA(B, BIndex);
index_RoleA(Memory_A, AIndex);
look_up_atom(Loc_Na,Msg,Msgno);
Commit_on_Na[BIndex, AIndex]=Msgno;}
::(B==Intruder)->skip;
fi;
```

These codes will assign the value of protocol *Na* to *Commit_on_Na*[*BIndex, AIndex*] and the value is held by a principal B acting the role *RoleB*, and thought to be agreed with principal *Memory_A* by B.These codes are use to explicitly the belief of principal B on the value *Na.*

- the connector also allocates an array *Init_on_Na*[*totol_inst_of_RoleA, totol_inst_of_RoleB*] to store all the encoding number of the possible instances of the protocol variable *Na* origninated by$a$ acting the role *RoleA, i.e., Init_on_Na*[*AIndex,*

[1] In fact, the syntax of multi-dimensional arrays can not be directly declared as *Commit_on_Na*[i, j], and they should be constructed indirectly with the use of typedef definitions. Here we use the syntax in order to make our context more readable.

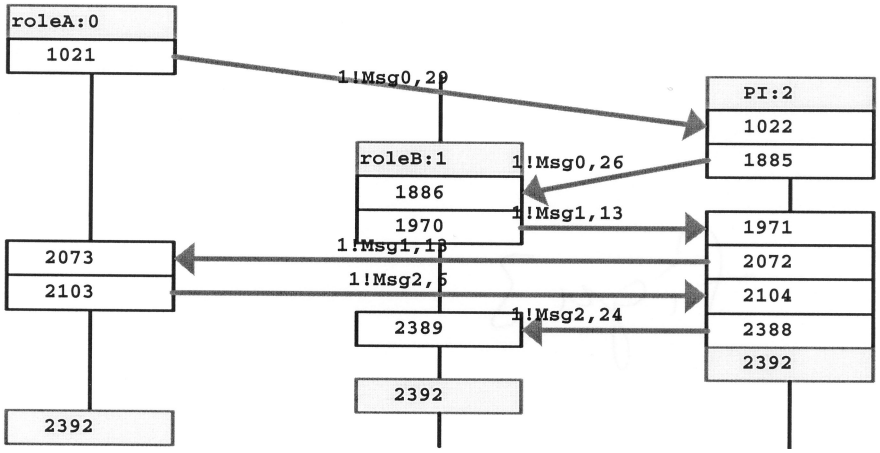


Figure 3: An attack on the NSPK protocol

BIndex] stores the encoding number of instance of *Na* held by the honest principals a acting the role *RoleA*, and$b$is the principal who is thought as the communication party by a in the session. The initial value of all the elements in this array are all set to -1. And the connector will insert the following codes into the body of proctype *RoleA* after the statement of the action of sending the message that contains *Na* in the first time.

if
::(A!=Intruder&&B!= Intruder)->{ index_Role_Inst(A, AIndex);
index_Role_Inst(B, BIndex);
look_up_atom(Na,Msg,Msgno);
Init_on_Na[AIndex, BIndex]=Msgno;}
::else ->skip;
fi;

The property precedes(B,A,ids(Na))) should be secret can be expressed in the following LTL formalism:

[](!(Commit_on_Na[0, 0] ==-1)
-> Commit_on_Na[0, 0]== Init_on _Na[0, 0]&&.....
!(Commit_on_Na[i, j] ==-1)-> Commit_on_Na [i,j]== Init_on_Na[j, i] &&.....)
where 0.≤i< *totol_inst_of_RoeB*, 0.≤j< *totol_inst_of_RoleA*

Spin can automatically find the attack shown in Figure 3 for the secrecy goal *secret*(*Nb*,ids(B,A)).

4 Related work and Conclusion

Although there are quite a few good works in the area of model checking cryptographic protocols, it seemed that there is few successful work by using Spin to model checking cryptographic protocols. In [9], Paolo Maggi used spin to model check security protocols, and to statically analyze the configuration of the sample protocol instance to simplify the intruder knowledge representation. And Zhang [10] ported the approach developed by G.Lowe in [11]to Spin environment. Both of them have checked the authentication properties for the classical case, i.e., the Needham-Schroeder Public key Authentication Protocol. Although they obtained the expected result, but both of their model are ad hoc to the Needham-Schroeder Public key Authentication Protocol, and are not easy to be adapted to other protocols. The crux is in that they did not propose a general approach to represent the facts used in the checked protocol sessions, so it is very difficult to automatically generate the promela model for protocol environment specification from its use-friendly specification if we follow their approach.

To solve this issue, we have developed the following approaches:

- We have developed a general technique to encode all the facts involved. Basing on it, we can easily represent the protocol messages, the intruder's knowledge and deduction rules;
- We have developed a general approach to model all honest principals. Basing on it, we have shown how to develop the code of theirs.
- We have developed a general approach to model the intruder. Basing on it, we have successfully modeled both data structures and the algorithms of the inference system of the intruder.
- Our approach described above can be adapted to a wide variety of protocols,and be employed to automatically generate Pomela model and LTL formula from a CIL specification. In fact, we have developed a CIL-Spin connector to help us with such work.

Similar work has been done by G.Lowe, who have developped FDR-CASPER [12] to model check cryptographic protocols. There are three main lessons out of his approach. The first is the uniformed way facts are defined symbolically for a variety of cryptographic protocols in CSP_M , the modeling language of FDR. The second is that a standard deduction system of the intruder is effectively emulated by a parallel process, which exploits the multi-way synchronizations allowed in CSP_M . The third is the way renaming operator has been used to wire all the nodes of the network together. The above have been integrated into the design of Casper, which takes protocols described in a more abstract notation as input, and outputs a CSP_M file to check various forms of correctness. All of this has made FDR into arguably the most powerful tool available

at the time of writing for checking properties of cryptographic protocols. Compared with the work , the way that the facts in a sample protocol instance are encoded in our framework is more complex than that in CSP_M because Promela does not directly support recursively defined symbolic data types to model the facts, whereas CSP_M does so. Despite the complexity of the way of encoding facts, we still consider it direct and useful for the applications of model checking involves in recursively defined symbolic data types. And the way we model the intruder and wire the network together is much simpler than that in [12], because the former only require the analyzer to have knowledge about simple imperative constructs such as loops, conditionals, and so on; whereas the latter requires that the analyzer should have profound knowledge in CSP. For example, in order to understand the way of modeling the deduction system of the intruder by a parallel process in CSPM, we should have good understanding about the multi-way synchronization operator, which is too complex to master for a people who are not experienced in CSP.

Another issue is efficiency. The initialization code and the code of the intruder's deduction system seem long enough, but the check we carried out to find the errors listed in this paper are well within Spin′s limit: the checks typically took about one second and searched about 565 states. The detail of verification results are listed in the appendix. So we think that our method is a practical and useful way of analyzing cryptographic protocols. We believe that the high efficiency is obtained by our fine-tuned model. First, we have minimized the set of symbolic facts as much as possible, as mentioned in section 3 Second, the encoding of intruder's deduction system is efficient enough to guarantee that no redundant deduction and no redundant facts will be generated, which minimizes the searching space as much as possible. Third, we use atomic steps as much as possible to minimize the amount of allowed interleaving. The above experiment is also an evidence that it is extremely important to optimize the model for us to obtain high efficiency in model-checking. In fact, we have adopted another two important approaches to achieve further efficiency, which is mentioned in [13]. The reason that we do not list he is in that the codes are not as readable as those here, and the interpretation of the attacker graph may not as clear as that here.

The last, but not the least is limitation of the technique of applying classical applying classical model checking technique to analyze security protocols. In principle, this technique models each honest principal taking part in the protocol and the intruder as a process, then examining all possible execution traces of the checked system to test whether the security properties are satisfied. Obviously, this technique is constrained by the intrinsic limitations: only finite systems of reasonably small size can be tackled. Here, "a finite system" means that only a finite number of runs of protocol sessions can be modeled, finite principals are involved, and each principal can be engaged in a finite number of runs of the protocol. Typically a finite system only accounts for at most three or four principals, including the intruder. However, if the system of limited size does not suffer any attacks, it is not obvious that neither does the system of arbitrary size. Another limitation of the technique is the expressibility for protocol.

No protocol goals have been discussed in detail except secrecy and authentication by classical model checking technique. But the principle underlying the design of a security protocols, in particular for a large security protocols, is very subtle, it does not necessarily mean that all its goals are achieved even if secrecy and authentication goals are achieved. It is difficult to express all protocol goals using the logical formalisms provided by the classical model checker. Nevertheless, model checking technique is more suitable for finding attacks, and in fact it is the main contribution that this technique has provided in this area. And Spin has provided message sequence charts to show attacks, which has better visualization than others and make it clearer to interpret the attacks. So we think it may be more proper if we use model checkers to find flaws for security property, and use theorem prover or others to prove that the design of a security protocol has achieved its goals. If we can not prove that the security properties has achieved, we can collect the environment information from the proof, set up the checked agents, and model check them to find whether there are attacks. Integration of different methods may lead to cleaner and deeper understanding for security protocols.

References

[1] J.Mitchell, M. Mitchell. CAPSL: common authentication protocol specification language. Technical report MP 97B48, The MITRE Corporation, 1997.

[2] Denker and Millen. CAPSL Intermediate Language, Workshop on Formal Methods and Security Protocols (FMSP'99). July 5, 1999, Trento, Italy.

[3] Gerard Holzmann. The Spin Model Checker, IEEE Trans. on Software Engineering, Vol. 23,No. 5, May 1997, pp. 279-295.

[4] Failures-Divergence Refinement v2.28 User Manual. Formal System (Europe) Ltd. Oct, 1997.

[5] Catherine Meadows, The NRL Protocol Analyzer: An Overview. Journal of Logic Programming. 26(2):113-131,1996.

[6] Dawn Song and Sergey Berezin and Adrian Perrig, Athena: a novel approach to efficient automatic security protocol analysis, Journal of Computer Security, volume 9, No. 2, pages 47-74, 2001.

[7] Friedrich Wilhelm Schroder. The GENTLE Compiler Construction System. R. Oldenbourg Verlag, Munich and Vienna, 1997.

[8] A.W. Roscoe, G. Lowe. Using CSP to detect errors in the TMN protocol. IEEE Transactions on Software Engineering , 1997.

[9] Paolo Maggi. Using Spin to verify security properties of cryptographic protocols. Lecture Notes in Computer Science, Volume 2318, Springer Verlag. April 2002, ISSN 0302-9743, ISBN 3-540-43477-1.

[10] Zhang Weihui. Promela model of Needham-Schroeder protocol. http://lcs.ios.ac.cn/∼zwh/promela/Promela.html, 2002.

[11] G.Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In Proceedings of the TACAS, pages 147–166, 1996.

[12] G. Lowe. Casper: a compiler for the analysis of cryptographic protocols, Proceedings of 1997 IEEE computer security foundations workshop, IEEE computer society press, 1997.

[13] V. Shmatikov, U. Stern. Efficient Finite-State Analysis for Large Security Protocols. 11th IEEE Computer Security Foundations Workshop (CSFW-11), pages 106-115, 1998.

Appendix

The detail version of this paper, the verification results are available in http://lcs.ios.ac.cn/˜lyj238/compac.html.

A Speech Act-Oriented Paradigm for Key Exchange Protocol Design

Phan Minh Dung
Phan Minh Thang
Department of Computer Science and Information Management
Asian Institute of Technology
GPO Box 4, Klong Luang, Pathumthani 12120, Thailand
dung,thangphm@cs.ait.ac.th

Abstract

We propose a novel multi-layers paradigm for the design of key exchange protocols. In the top layer, protocols are specified in a high-level, declarative, formal language using speech acts as the basic components. The declarative semantics of speech acts are specified by their preconditions and effects like in Hoare logics. No reference to cryptography is made at this level. High-level speech act-oriented protocols are translated into lower-level message exchanging protocols by a "protocol compiler" that implements speech acts by sending and receiving appropriate encrypted messages.

Under the Dolev-Yao assumption of perfect cryptography, the proposed new paradigm offers two novelties:

1) Using the language of speech acts, protocol designers could develop their protocols in a modular and compositional way that are correct from the outset without any reference to cryptography

2) The secrecy of exchanged keys is guaranteed by simply requiring that a speech act should not be executed if its preconditions are not satisfied, a general but easily verifiable condition called *well-designedness*.

1. INTRODUCTION

In general, security protocols often have structures that determines the composition of the intention and semantics of the individual messages. Recognizing these structure could allow us to create a radically new modular approach to security protocol development in which high-level security designs are translated stepwise to code. The obtained protocols at all levels are guaranteed to be correct from the outset. To illustrate this idea, let us look at an example

Example 0.1 Consider a protocol for distributing a fresh session key K generated by a server S, to two principals A and B. At the end of the protocol, both A and B are expected to get K and also to know that the other has got K as well.

(1) $A \rightarrow S:\ req, newkey, A, Init, S, Server, N_a, B$
(2) $S \rightarrow A:\ \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}}$
(3) $S \rightarrow B:\ \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(4) $B \rightarrow A:\ \{req, keyconfirm, B, Resp, A, Init, N_b, Hash(K), S\}_{K_A}$
(5) $A \rightarrow B:\ \{rep, keyconfirm, A, Init, B, Resp, N_b\}_{K_B}$
(6) $B \rightarrow A:\ \{inf, keyconfirm, B, Resp, A, Init, S\}_K$

Note that K_X (resp. K_{XY}) represents a public (resp. secret common) key of X (resp. between X and Y). $\{g\}_k$ denotes the encryption of g using key k. Following the prudent engineering practice of Abadi and Needham [2] each message is designed to contain explicitly the identity of the sender, receiver together with their roles in the protocol as well as the purpose of the message.

The first message is a request from A, as initiator, to S, as server, for a fresh session key with B. The keywords *req, newkey* indicate that the message represents a request for a new key.

S replies by sending a newly generated session key K to A in the second message. The keywords *rep, newkey* indicate that the message is a reply to an earlier request for a new key. After getting the reply message, A knows that K is a fresh session key between A and B generated by S.

S informs B about key K in the third message (B plays the role of a responder). The keywords *inform, newkey* indicate that the message is intended to inform B about a new key. After receiving the third message, B could only say that it has been informed about K without being sure whether K is fresh or not. This is because B has no knowledge about K before receiving the third message and hence, B could not verify whether this message is just sent recently or it has been replayed by a penetrator.

To verify the information it has just obtained, B requests A in the fourth message to confirm that A knows that K is indeed a fresh session key between A and B generated by S. The keywords *req, keyconfirm* indicate that the message represents a request for confirming the status of the included key.

A replies to B by sending the fifth message to confirm that K is indeed a fresh session key between A and B. The keywords *rep, keyconfirm* indicate that the message represents a reply to an earlier request to confirm the status of K. After getting this message, B knows that what it has been informed before is correct.

B sends A the sixth message to confirm that B now knows that K is a secret session key between A and B. The keywords *inform, keyconfirm* indicate that the intention of the message is for B, to confirm to A that it knows about K.

The above protocol could be viewed as an implementation of the following more abstract one using speech acts as the basic building blocks[1]:
S_1 : *A requests S to generate a new session key for communication with B*
S_2 : *S replies to A by sending A a newly generated key K*

[1]Speech acts have been proposed as the basic primitives for declarative agent communication language in the AI community [14]

S_3 : *S informs B that K is a new session key between A and B generated by S*
S_4 : *B requests A to confirm that K is a fresh session key between them generated by S*
S_5 : *A replies to B to confirm that K is indeed a fresh session key between them generated by S*
S_6 : *B informs A to confirm that B knows that K is a fresh session key between them generated by S*
where A,B,S play the roles of initiator, responder and key server respectively.

The new approach proposed in this paper allows protocol designers to develop their protocols in an abstract speech act-oriented language like above without worrying about cryptography. A "protocol compiler" would translate automatically the abstract protocols into lower-level ones.

◇

In this paper, we propose a multi-layered approach in the design of key exchange protocols. In the top layer, protocols are specified using speech acts as the basic building blocks. No reference to cryptography is made at this level. Like in Hoare logics, the declarative semantics of speech acts are specified by their preconditions and effects. High-level protocols are translated automatically into the lower-level message-exchanging protocols by "protocol compilers" where speech acts are implemented by sending and receiving appropriate encrypted messages. Like in conventional programming, "protocol compilers" are developed by the designers of speech act languages, not by their users. Ensuring the correctness of "protocol compilers" is hence a responsibility of the speech act language designers, not of the protocol programmers. Under the Dolev-Yao assumption of perfect cryptography, the proposed new paradigm offers two key novelties :

1) Using the language of speech acts, protocol designers could develop their protocols in a modular and compositional way that are correct from the outset.

2) The secrecy of exchanged keys is guaranteed by simply requiring that a speech act should not be executed if its preconditions are not satisfied, a general but easily verifiable condition called *well-designedness*.

The paper is organized as follows. In chapter 2, we present a language of speech acts and the call of well-designed protocols. In chapter 3, a protocol logic called ProtoLog that is based on the modal system S5 [13, 9] is introduced. A "protocol compiler" is introduced in chapter 4 and the soundness and completeness of the protocol logic ProtoLog wrt the translation implemented by the compiler are discussed in chapter 5. In chapter 6, we discuss related works.

2. SPEECH ACTS AND WELL-DESIGNED PROTOCOLS

2.1 KL: A Keys Logic

We first propose a simple logical language for specifying the mental states of protocol principals. In contrast to BAN-style logics [4, 19], we do not employ a belief

operator. Instead we have two operators: ”being informed” and ”knowing”. Intuitively, to say that an agent knows something is to say that the agent's information about this something is correct and the agent is also aware about the correctness of the information [9] while to say that an agent is informed of something simply indicates that the principal has obtained a piece of information without giving any hint about the correctness of the information as well as the awareness of the agent about it.

We assume the existence of pairwise disjoint sets **NONCE, NVAR, KEY, KVAR** and **PI, PVAR** of nonces, nonce variables, keys, key variables and principals and principal variables respectively. There is a distinguished identifier $\mathbf{PE} \in PI$ denoting the *penetrator*. A principal that is not the penetrator is called *regular*. In security protocols, principals often play different roles like the roles of initiators, responders or key servers. The roles of the principals involved in a protocol form an important component in its semantics. We assume the existence of a finite set **RO** of predefined role identifiers. A *principal term* is either a principal from PI, or a principal variable from PVar. *Nonce term* and *key terms* are defined similarly.

Let A,B,C,X,Y,Z be principal terms, n be a nonce term and K be a key term and ρ be a role. A *basic formula* has one of the following forms:

1. $GK_{A,\rho}(K, B, C)$ stating that principal A acting in role ρ has generated a fresh key K as a session key between B and C
2. $GN_{A,\rho}(n)$ stating that principal A acting in role ρ has freshly generated nonce n
3. $HN_{A,\rho}(n)$ stating that principal A acting in role ρ has nonce n
4. $Informed_{A,\rho}(K, X, Y, Z)$ stating that principal A acting in role ρ has been informed that K is generated recently by Z as a session key between X,Y.
5. $Key(K, A, B, C)$ stating that K is a session key between principals A and B generated recently by C.
6. $Access(A, K)$ stating that A has access to key K

A *formula* is composed from basic formulas using the logical operators $\wedge, \vee, \neg, \rightarrow$ together with the knowledge operator $Know_{A,\rho}F$ stating that A acting in the role ρ, knows that F holds.

One may wonder of whether it is possible to replace the notion Key(K,A,B,C) by just Key(K,A,B) without mentioning explicitly the generator C of K. Example 0.1 is a case where an employment of Key(K,A,B) would lead the responder B to believe in the trustworthiness of K even if S is the penetrator and A mistakenly believes S is honest while B is aware about the true identity of S.

We introduce now KL (Keys Logic), a specialized version of the modal logic S5, for reasoning about keys and its association to principals. The logic is needed to define

well-designed protocols. Let A,B,C,D be regular principal identifiers and X be a principal identifier. KL extends the system S5 in modal logic [13, 9] with the following axioms:

A1 $Key(K, A, B, C) \rightarrow Key(K, B, A, C)$
A2 $Informed_{A,\rho}(K, B, C, D) \rightarrow Informed_{A,\rho}(K, C, B, D)$
A3 $GK_{A,\rho}(K, B, C) \rightarrow GK_{A,\rho}(K, C, B)$
A4 $GN_{A,\rho}(n) \rightarrow HN_{A,\rho}(n)$
A5 $GK_{A,\rho}(K, B, C) \rightarrow Key(K, B, C, A)$
A6 $F \rightarrow Know_{A,\rho}F$ *for all basic formula F of the form* $GK_{A,\rho}(K, B, C)$, $GN_{A,\rho}(n)$, $HN_{A,\rho}(n)$ *and* $Informed_{A,\rho}(K, B, C, D)$
A7 $Know_{A,\rho}Key(K, B, C, D) \rightarrow Informed_{A,\rho}(K, B, C, D)$
A8 $Informed_{A,\rho}(K, B, C, D) \rightarrow Access(A, K)$

Axioms A1-A5 and A8 follow directly from the intuitions of the involved basic formulas. The intuition of axiom A6 is that if a principal is doing something then it is also aware about it. Axiom A7 relates the knowing and being informed operators, stating the obvious that if A knows that K is a fresh session key between B,C generated by D then A is also informed about it. The axioms and proof rule in modal system S5 [13, 9] are adapted to KL as follows:

$Know_{A,\rho}F \rightarrow F$

$Know_{A,\rho}F \wedge Know_{A,\rho}(F \rightarrow G) \rightarrow Know_{A,\rho}G$

$Know_{A,\rho}F \rightarrow Know_{A,\rho}Know_{A,\rho}F$

$\neg Know_{A,\rho}F \rightarrow Know_{A,\rho}\neg Know_{A,\rho}F$

From $\vdash F$ infer $\vdash Know_{A,\rho}F$

2.2 Speech Acts and Well-Designed Protocols

We introduce a set of speech acts consisting of Request, Reply and Inform acts that we believe form a core of primitives that capture the most essential structures in security protocols. This set is extensible to capture further structures with new acts.

There are two types of speech acts: *newkey, keyconfirm*. Intuitively, an act of type *newkey* is used for distributing a new key while an act of type *keyconfirm* is used for confirming the knowledge of a new key. For example, the first (resp. last) three acts in the abstract protocol in example 0.1 are of type *newkey* (resp. *keyconfirm*).

Speech acts are defined as speech act forms that contain no variables. The structure of speech act forms are given in the following table

Request	**Reply**	**Inform**
Sender: A	**Sender: A**	**Sender: A**
Role of Sender: ρ	**Role of Sender:** ρ	**Role of Sender:** ρ
Receiver: B	**Receiver:** B	**Receiver:** B
Role of Receiver: τ	**Role of Receiver:** τ	**Role of Receiver:** τ
Type: t	**Type:** t	**Type:** t
Content: Con	**Content:** Con	**Content:** Con
Reply-With: n	**Reply-To:** n	

where A,B are principal terms, ρ, τ are roles , n is a nonce term, $t \in \{newkey, keyconfirm\}$, and Con represents the content of the acts. It is required that the roles of sender and receiver in an act must be different, i.e. $\rho \neq \tau$. The nonce n in a request act is generated randomly by A when A performs the act.

The content of a request of type *newkey* has the form *Key(?,A,C,B)* stating that A requests B to generate a new session key for A to communicate with C.

The content of a request of type *keyconfirm* has the form *Key(K,A,B,C)* stating that A asks B to confirm that B knows that K is a fresh session key between A and B generated by C.

In a reply or inform act of type *newkey*, the sender A sends the receiver B a freshly generated session key for communication with C. If the key has not been generated, then the sender has to generate it first and then sends it. In this case the content of the act has the form $\nu K.Key(K, B, C, A)$. Otherwise it has the form $Key(K, B, C, A)$ in a reply act or the form $Key(K, B, C, A)$ or $Key(K, B, A, C)$ (depending on whether K is generated by A or by C) in an inform act.

The content of a reply or inform act of type *keyconfirm* is simply of the form $Key(K, A, B, C)$ affirming that K is indeed a fresh session key between A and B generated by C.

The declarative semantics of speech acts is specified by their preconditions and effects. The preconditions describe the necessary information and knowledge a honest principal should have to be able to send the acts and for the receiver to accept and process it *without leaking any secret information or being fooled by the penetrator*. The effects of a speech act describe the information and knowledge a principal gains after sending it (for the sender) or receiving it (for the receiver). Regular principals are assumed to be honest.

Let S be a speech act form as defined above. The preconditions and effects of the event of sending (resp. receiving) S are denoted by Pre(+S) and Eff(+S) (resp. Pre(-S) and Eff(-S)) and formalized in the tables below.

The definition of key exchange protocols is based on a notion of conversation form where a **conversation form** is a pair (Req, Rep) of request and reply act forms of the same type and with the same nonce such that 1) the sender (resp. receiver) and its role in Req coincide with the receiver (resp. sender) and its role in Rep respectively, and 2) if the acts in the conversation are of type *keyconfirm* then their contents coincide, and 3) if the acts in the conversation are of type *newkey* and the content of the request has the form Key(?,A,C,B) then the content of the reply has the form Key(K,A,C,B) or $\nu K.Key(K, A, C, B)$

Definition *A* **speech act oriented key exchange protocol** *(or simply key exchange protocol) is a sequence of speech act forms $S_1, \ldots, S_k$ such that*

1) For each request (resp. reply) S_i, there is exactly one reply (resp. request) S_j, $j > i$ (resp. $j < i$) such that S_i, S_j (resp. S_j, S_i) form a conversation form.

2) Each principal term has at most one role, i.e. for each principal term P in AP,

if there exist S_i, S_j such that P appears as a sender or receiver in both S_i and S_j then P has the same role in both acts. [2]

3) Every role in AP is occupied by exactly one principal term, i.e. for all principal terms P,Q in AP, if there exist S_i, S_j and $\rho \in RO$ such that P has role ρ in S_i and Q has role ρ in S_j then P = Q.

4) Variables representing nonces of different conversation forms are standardized apart.

5) The penetrator identifier does not appear in any S_i.

6) There is exactly one key term appearing in the protocol and the first act in which it appears is a reply or inform act of type newkey that is also the only act in the protocol with a content of the form $\nu K.Key(K, A, B, C)$

Now we can introduce well-designed protocols. For each speech act form S, define $Pre(S) = Pre(+S) \wedge Pre(-S)$ and $Eff(S) = Eff(+S) \wedge Eff(-S)$.

Definition *A key exchange protocol AP = $S_1, \ldots, S_k$ is said to be* **well-designed** *if speech acts are scheduled for execution only when their preconditions are satisfied, that means following conditions are satisfied:*

1) $Pre(S_1) = True$

2) For each $2 \leq i \leq k$: $Eff(S_1) \wedge \ldots \wedge Eff(S_{i-1}) \vdash Pre(S_i)$

Act S	Request	
	newkey	keyconfirm
Pre(+S)	True	$Informed_{A,\rho}(K, A, B, C)$
Eff(+S)	$GN_{A,\rho}(n)$	$GN_{A,\rho}(n)$
Pre(-S)	True	$Know_{B,\tau}Key(K, A, B, C)$
Eff(-S)	$HN_{B,\tau}(n)$	$Know_{B,\tau}Informed_{A,\rho}(K, A, B, C) \wedge HN_{B,\tau}(n)$

[2] This condition ensures that normally different principals have different roles in a run of the protocol.

Act	Reply		
	newkey		
S	$Con = \nu K.Key(K, B, C, A)$	$Con = Key(K, B, C, A)$	keyconfirm
Pre(+S)	$HN_{A,\rho}(n)$	$HN_{A,\rho}(n) \wedge GK_{A,\rho}(K, B, C)$	$Know_{A,\rho}Key(K, A, B, C) \wedge HN_{A,\rho}(n)$
Eff(+S)	$GK_{A,\rho}(K, B, C)$	True	True
Pre(-S)	$GN_{B,\tau}(n)$		$GN_{B,\tau}(n) \wedge Informed_{B,\tau}(K, A, B, C)$
Eff(-S)	$Know_{B,\tau}GK_{A,\rho}(K, B, C) \wedge Know_{B,\tau}HN_{A,\rho}(n)$		$Know_{B,\tau}HN_{A,\rho}(n) \wedge Know_{B\tau}Know_{A\rho} Key(K, A, B, C)$

Act	Inform		
	newkey		
S	$Con = \nu K.Key(K, B, C, A)$	$Con = Key(K, B, X, Y)$ where $(X, Y) = (A, C)$ or (C, A)	keyconfirm
Pre(+S)	True	$Know_{A,\rho}Key(K, B, X, Y)$	$Know_{A,\rho}Key(K, A, B, C)$
Eff(+S)	$GK_{A,\rho}(K, B, C)$	True	True
Pre(-S)	True	True	$Know_{B,\tau}Key(K, A, B, C)$
Eff(-S)	$Informed_{B,\tau}(K, B, C, A)$	$Informed_{B,\tau}(K, B, X, Y)$	$Know_{B,\tau}Know_{A,\rho} Key(K, A, B, C)$

It is not difficult to see that the abstract protocol in example 0.1 is well-designed. The following example illustrates how a violation of the well-designed condition could lead to serious flaws.

Let $S_1, \ldots, S_6$ be the acts of the abstract protocol in example 0.1. Consider a new protocol $P' = S'_1, S'_2, S'_3, S'_4, S'_5$ defined by $S'_i = S_i$ for $i \in \{1, 2, 3\}$ and $S'_4 = S_6$ and S'_5 is an inform act of type *keyconfirm* from A to B to confirm the receipt of K. P' is in fact an abstraction of the Needham-Schroeder protocol with symmetric key (NSPS) [6]. P' is not well-designed as the preconditions of S'_4 do not follow from the effects of the previous acts in it. Similar to NSPS, P' is subjected to a replay attack in which

the penetrator replays an old message S_3' to B whose key K has become stale. The penetrator then intercepts the message S_4' that B sends to A and then sends S_5' to B as specified by the protocol. B is then tricked into falsely believing that it has a secret common key for communicating with A generated by S.

3. PROTOLOG: A LOGIC FOR WELL-DESIGNED PROTOCOLS

We introduce now ProtoLog, a protocol logic, for reasoning about the mental states of principals participating in well-designed protocols. Let $\rho \in RO$ be a role in a protocol $AP = S_1, \ldots, S_n$. The ρ-**track of** AP is the sequence $AP_\rho = \sigma_{i_1} S_{i_1}, \ldots, \sigma_{i_k} S_{i_k}$ where $\sigma_{i_j} \in \{+, -\}$, $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, such that for each $1 \leq j \leq n$, $+S_j \in AP_\rho$ (resp.$-S_j \in AP_\rho$) iff ρ is the role of the sender (resp. receiver) in S_j. For illustration, the initiator-track of the abstract protocol in example 0.1 is $+S_1, -S_2, -S_4, +S_5, -S_6$ while its responder-track is $-S_3, +S_4, -S_5, +S_6$. A **run** of principal A in the role ρ according to a protocol AP is a ground instance of a prefix of the ρ-track AP_ρ of AP in which A is the principal having the role ρ. A **proper-regular** run is a run where the penetrator PE does not appear in any of its acts. Let $R = E_1 \ldots E_k$ be a run. Define $Effect(R) = Effect(E_1) \wedge \ldots \wedge Effect(E_k)$.

To reason about the effects of runs of protocols , we introduce new formulas called **protocol formulas** of the form $\mathbf{AP.R}\,[\mathbf{F}]$ stating that formula F holds after run R according to AP has been executed.

Let AP be a protocol, E be an event of sending or receiving a speech act, R be a proper-regular run of A in a role ρ according to AP, and F,F' be formulas. Further let A,B,C,D be regular principal identifiers and X be a principal identifier. Protolog extends KL with following axioms and proof rules:

A9 $Key(K, B, C, A) \rightarrow GK_{A,\rho_1}(K, B, C) \vee \ldots \vee GK_{A,\rho_k}(K, B, C)$
where $RO = \{\rho_1, \ldots, \rho_k\}$

A10 $GK_{A,\rho}(K, B, C) \wedge GK_{A',\rho'}(K, B', C') \rightarrow (A, \rho) = (A', \rho') \wedge (B, C) = (B', C')$

A11 $A \neq B \wedge A \neq C \wedge A \neq D \rightarrow \neg Informed_{A,\rho}(K, B, C, D)$

A12 $Key(K, B, C, A) \wedge X \neq A \wedge X \neq B \wedge X \neq C \rightarrow \neg Access(X, K)$

Effect Rule

$$\frac{}{\vdash AP.R\,[Effect(R)]}$$

Consequence Rule

$$\frac{\vdash AP.R\,[F],\ \vdash F \rightarrow F'}{\vdash AP.R\,[F']}$$

Axiom A9 states that if a key K is generated by A then K is generated by A acting in some role. Axioms A10,A11,A12 are based on the honesty assumption of regular principals. The intuition of axiom A10 is that generated keys are random and hence it is impossible for an agent to generate the same key twice. The axiom A11 follows from the structure of the speech acts that requires that when a key is sent in a speech

act, information about the association of the key to the receiver of the act must be included. Hence a honest principal would never receive a key that is not associated with it. Axiom A12 states that freshly generated session keys are accessible only by their generator and the principals for whom they are generated. This axiom does not hold for every protocol but it holds for well-designed protocols. In other words, well-designed protocols protect the secrecy of the exchanged keys.

Let AP be the abstract protocol in example 0.1. Let R_0, R_1 be complete runs of A and B in the initiator and responder roles respectively. It is easy to see that
$\vdash AP.R_0[Know_{A,Init}Know_{B,Resp}Key(K,A,B,S)]$ and
$\vdash AP.R_1[Know_{B,Resp}Know_{A,Init}Key(K,A,B,S)]$ hold

4. IMPLEMENTING SPEECH ACTS

A speech act is implemented by sending and receiving certain message. The message used to implement the events of sending and receiving a speech act S is denoted by $\mathbf{m_S}$. We also often say that m_S *represents* S. Taking a hint from prudent engineering [2], a representation of a speech act should contain vital information about its type, its content, the identity and role of its sender and receiver and other information like reply-to, and reply-with-nonces. There are many ways to implement a speech act. In this chapter, we give some of them. A systematic study of all possible ways to implement speech acts is beyond the scope of this paper.

Let S be a speech act from a sender A in role ρ to a receiver B in role τ.

If S is an inform act of type *newkey* whose content contains the formula Key(K,B, X,Y) (where (X,Y) = (A,C) or (C,A)) then

$$m_S = \{inf, newkey, A, \rho, B, \tau, K, B, X, Y\}_{K_{AB}}, \text{ or}$$
$$m_S = \{inf, newkey, A, \rho, B, \tau, K, B, X, Y\}_{K_B},$$

If S is an inform act of type *keyconfirm* with a content of the form Key(K,A,B,C) then

$$m_S = \{inf, keyconfirm, A, \rho, B, \tau, C\}_K \text{ , or}$$
$$m_S = \{inf, keyconfirm, A, \rho, B, \tau, Hash(K), C\}_{K_B}$$

The representation of a request (resp. reply) act depends on the representation of the reply (resp. request) act in the same conversation. Let (S,S') be a conversation form and n be the nonce in S,S'.

If (S,S') is a conversation of type *newkey* and the content of S' contains the formula $Key(K,A,C,B)$, then there are at least two different ways to represent (S,S'):

- $m_S = \{req, newkey, A, \rho, B, \tau, n, C\}_{K_B}$ and
  $m_{S'} = \{rep, newkey, B, \tau, A, \rho, n, K, C\}_{K_A}$
- $m_S = req, newkey, A, \rho, B, \tau, n, C$ and
  $m_{S'} = \{rep, newkey, B, \tau, A, \rho, n, K, C\}_{K_{AB}}$,

If (S,S') is a conversation of type *keyconfirm* and the content of S is Key(K,A,B,C), then there are at least two different ways to represent (S,S'):

- $m_S = \{req, keyconfirm, A, \rho, B, \tau, n, Hash(K), C\}_{K_B}$
  and $m_{S'} = \{rep, keyconfirm, B, \tau, A, \rho, n\}_{K_A}$
- $m_S = req, keyconfirm, A, \rho, B, \tau, n, Hash(K), C$
  and $m_{S'} = \{rep, keyconfirm, B, \tau, A, \rho, n, Hash(K), C\}_{K_{AB}}$

As there are many different ways to implement speech acts, a speech act-oriented protocol could have many different implementations. For example, there are at least 16 different but equivalent ways to implement the abstract protocol in example 0.1.

Role Topology and Message Forwarding

Many security protocols do not allow principals acting in certain roles to communicate directly. An example is the well-known Otway-Rees protocol [6] that does not allow initiators to communicate directly with servers. All messages between principals in these roles are routed through the responders.

In general, each security protocol P assumes the existence of a directed graph $G = (RO, V)$, $V = RO \times RO$ describing the connection topology of the roles in P. A direct link from role ρ to role τ in G represents a direct communication channel from principals acting in role ρ to principals acting in role τ. A question that immediately arises is how to implement a speech act if the role topology forbids a direct communication between its sender and receiver.

Let P be a well-designed speech act oriented protocol and $G = (RO, V)$ be a given connection topology for the roles in P. Let S be a speech act in P and ρ, τ be the roles of the sender A and receiver B in S respectively. Further let $\rho, \rho_1, \ldots, \rho_k, \tau$ be a shortest path from ρ to τ with $A_1, \ldots, A_k$ being the principal terms occupying the roles $\rho_1, \ldots, \rho_k$ in P. S is implemented by forwarding the message m_S from A through $A_1, \ldots, A_k$ to B as: $A \xrightarrow{m_S} A_1 \xrightarrow{m_S} \ldots \xrightarrow{m_S} A_k \xrightarrow{m_S} B$. P is translated into a message exchanging protocol by successively translating each of its acts as above.

Consider the abstract speech act oriented protocol in example 0.1. Suppose that the connection topology forbids direct communication between responders and servers. In this case, the speech act (3) in which S informs B about the new key K is implemented by letting S (server) send message to B (responder) through A (initiator). The abstract protocol is implemented by the following message exchanging protocol:

(1) $A \rightarrow S:\ req, newkey, A, Init, S, Server, N_a, B$
(2) $S \rightarrow A:\ \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}}$
(3.1) $S \rightarrow A:\ \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(3.2) $A \rightarrow B:\ \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(4) $B \rightarrow A:\ \{req, keyconfirm, B, Resp, A, Init, N_b, Hash(K), S\}_{K_A}$
(5) $A \rightarrow B:\ \{rep, keyconfirm, A, Init, B, Resp, N_b\}_{K_B}$
(6) $B \rightarrow A:\ \{inf, keyconfirm, B, Resp, A, Init, S\}_K$

Steps (2) and (3.1) could be combined by letting S sending both m_{S_2}, m_{S_3} to A in one step.

5. SOUNDNESS AND COMPLETENESS OF PROTOLOG

Strand spaces have been introduced by Fabrega, Herzog and Guttman [8] to give an operational semantics for message-exchanging protocols under the Dolev-Yao assumption of perfect cryptography. We adapt the strand space model to our framework to give an operational semantics to the lower-level message-exchanging protocols. The protocol logic Protolog is then interpreted over the adapted strand models.

Let AP be an arbitrary but fixed speech act oriented key exchange protocol. For simplicity and due to the limited space, we assume that AP has a role topology in which there is a direct link between every pair of different roles (all the results in this chapter are proved for the general case in the full paper). A strand is a sequence of nodes labelled by actions $\pm m$ of sending or receiving messages m. The ith node of a strand s is denoted by (s,i). Act(s,i) denotes the ith action in s. The message of the action labelling (s,i) is denoted by $term(s,i)$. Let $\Rightarrow = \{((s,i),(s,i+1)) \mid s$ is a strand $\}$

Let S be a set of strands and (s,i) be a node in S. A key or nonce b is said to *originate at (s,i)* if Act(s,i) is a sending action and b occurs in term(s,i) and for all $j < i$, b does not occur in term(s,j). b is said to *uniquely originate at (s,i) in S* if (s,i) is the only node in S at which b originates.

There are two kinds of strands, *regular strand* and *penetrator strands*. A node lying on a regular (resp. penetrator) strand is called a *regular (resp. penetrator) node*. We often say that *a regular node N implements a speech act event* $\pm S$ if the action labelling N is $\pm m_S$. For a regular principal A and a role $\rho \in RO$, an *AP-regular strand* of A in role ρ is a regular strand $s = N_1, \ldots, N_m$ such that each N_i implements the event E_i in a run $R = E_1, \ldots, E_m$ of A in role ρ according to AP. We also say that *strand s implements run R*. There are eight kinds of penetrator strands:

Key-strand: $\langle +K \rangle$ where $K \in KEY \setminus \mathcal{K}$	S-strand: $\langle -gh$+g+h$\rangle$
RO-strand: $\langle +\rho \rangle$ where $\rho \in RO$	E-strand: $\langle -K - h + \{h\}_K \rangle$
PI-strand: $\langle +A \rangle$ where $A \in PI$	C-strand: $\langle -g - h + gh \rangle$
Nonce-strand: $\langle +n \rangle$ where $n \in NONCE$	D-strand: $\langle -K^{-1} - \{h\}_K + h \rangle$

where $\mathcal{K}$ consists of all secret shared keys K_{AB}, and private keys K_A^{-1} of regular principals.

With the exception of the Key, and RO-strands, the other definitions of penetrator strands coincide with the definitions given in [8, 11]. In [8, 11], the penetrator is assumed to possess initially a set of keys and the key the penetrator could pick up in a key strand should belong to this set. We do not impose this restriction on the penetrator as we assume that the penetrator has access to any algorithm that could be used for key or nonce generation. The assumption that the keys or nonces generated by regular principals are random and hence could not be generated by the penetrator is captured by their unique origination requirement in the defintion of bundles (introduced shortly below)

A **bundle of AP** is is defined as a pair $(S, \rightarrow)$ whereas S is a finite set of strands,

$\rightarrow$ is a binary relation over the set of nodes in S such that the following conditions are satisfied:

1) Every regular strand in S is AP-regular.

2) For each node (s,i) in S, if Act(s,i) is a receiving action then there is exactly one node (s',i') such that $Act(s', i')$ is a sending action and $(s', i') \rightarrow (s, i)$ and $term(s, i) = term(s', i')$ hold

3) The transitive closure of the relation $\rightarrow \cup \Rightarrow$ is acyclic

4) For each regular node (s,i) in S, if a nonce or a key b is generated in the speech act implemented by (s,i), then b uniquely originates at (s,i) in S.

5) If a key or nonce b originates at a regular node (s,i) then b is also generated at (s,i).

Note that the assumption that the keys or nonces generated by principals during a run of the protocol are random is captured by their unique origination requirement.

We say that a *key K is associated with principals A,B,C* in a speech act S if the content of S contain the formula Key(K,A,B,C) or Key(K,B,A,C). It is not difficult to see that for each regular node N, term(N) represents uniquely a speech act. We say that a *key K is associated with principal identifiers A,B,C at a node N* if K is associated with principal identifiers A,B,C in the speech act represented by term(N).

Definition(**Model Semantics**) Let $\mathcal{B}$ be a bundle of a well-designed protocol, A,B,C be regular principals, D be an arbitrary principal and ρ be a role. Further let n be a nonce and K be a key.

1. $\mathcal{B} \models HN_{A,\rho}(n)$ iff there exists a regular strand $s \in \mathcal{B}$ of A in role ρ such that n is contained in an action in s.

2. $\mathcal{B} \models GN_{A,\rho}(n)$ iff there exists a regular strand $s \in \mathcal{B}$ of A in role ρ such that nonce n is generated in some sending action in s.

3. $\mathcal{B} \models Informed_{A,\rho}(K, B, C, D)$ iff there is a regular strand $s \in \mathcal{B}$ of A in role ρ and a node N in s such that N implements the (sending or receiving) event of an inform or reply act and K is associated with B,C,D at N.

4. $\mathcal{B} \models GK_{A,\rho}(K, B, C)$ iff there exists a regular strand $s \in \mathcal{B}$ of A in role ρ such that key K is generated at some sending node N in s and K is associated with B,C,A at N

5. $\mathcal{B} \models Access(D, K)$ iff

   (a) D is a regular principal and there exists a regular strand $s \in \mathcal{B}$ of D such that K is contained in an action in s, or

   (b) D is the penetrator and there exists a penetrator node N such that K = term(N)

6. $\mathcal{B} \models Key(K, A, B, C)$ iff there exists a regular strand $s \in \mathcal{B}$ of C such that key K is generated at some sending node N in s and K is associated with A,B,C at N

Let $\mathcal{B}$ and $\mathcal{B}'$ be bundles of the same well-designed protocol, and A be a regular principal identifier and ρ be a role. Further let Ω (resp. Ω') be the set of nodes of A in role ρ in $\mathcal{B}$ (resp. $\mathcal{B}'$). We say that $\mathcal{B}, \mathcal{B}'$ are (A, ρ)-*indistinguishable,* denoted by $\mathcal{B} \equiv_{A,\rho} \mathcal{B}'$ if there is a bijection ϕ between Ω and Ω' such that for all nodes $N, N' \in \Omega$ following conditions hold:

1) The actions labelling N and $\phi(N)$ coincide.

2) $N \Rightarrow N'$ iff $\phi(N) \Rightarrow \phi(N')$

What a principal in a role ρ knows at a certain state of the world depends on what it considers to be a possible world at this state [9]. Syverson [17] defined a world state as a bundle. Similarly, we define a world state as a bundle where a possible world for a principal A acting in a role ρ in a state $\mathcal{B}$ is a historical bundle $\mathcal{B}'$ such that $\mathcal{B}, \mathcal{B}'$ are (A, ρ)-indistinguishable.

Definition(**Model Semantics**, continued)
$\mathcal{B} \models Know_{A,\rho} F$ iff for all $\mathcal{B}'$ s.t. $\mathcal{B}, \mathcal{B}'$ are (A, ρ)-indistinguishable: $\mathcal{B}' \models F$

Let R be a run of a principal A according to a well-designed AP. Define
$\models \mathbf{AP}.\mathbf{R}[\mathbf{F}]$ iff for each bundle $\mathcal{B}$ of AP, if $\mathcal{B}$ contains a recent strand implementing R then $\mathcal{B} \models F$.

Let AP be a well-designed protocol, $\mathcal{B}$ be a bundle of AP, F be a formula and R be a proper-regular run of A in role ρ according to AP. The soundness as well as a limited completeness of the protocol logic ProtoLog are established in the following theorems. The proofs are given in the full version of this paper.

Theorem 1(Soundness)

1) If $\vdash F$ then $\models F$

2) If $\vdash AP.R[F]$ then $\models AP.R[F]$

Theorem 2(Limited Completeness)

If $\models AP.R[Know_{A,\rho} Key(K, B, C, D)]$
then $\vdash AP.R[Know_{A,\rho} Key(K, B, C, D)]$

6. RELATED WORKS

There is a good body of work on designing security protocols. Guttman [10] has argued that the framework of authentication tests could be used in the design of security protocols. The notion of conversation we introduced could be viewed as a high-level declarative embodiment of the idea of authentication tests. Gong and Syverson [12] has developed an informal method for developing fail-stop security protocols. Meadows [15] has suggested that the design of cryptographic protocols should be layered in which a abstract model is used at the top layer and each successive layer is an implementation of the layer above it. A requirement language for security protocols has been given by Syverson and Meadows in [18]. Buttyan, Staamann and Wilhelm [5] has proposed an abstract BAN-like logic to be used in the protocol design. But it is not clear how to translate a protocol specified in the abstract logic into a lower level protocol and how to verify the correctness of such translation. Boyd and Mao [3] have

discussed informally set-theoretic guidelines for the design of key exchange protocols. Abadi and Needham [2] have proposed a set of informal guidelines for authentication protocol design. Perrig and Song [16] has developed tools based on the idea of strand space for analyzing protocols and later applied it in the design of protocols. A more recent work by Datta, Derek, Mitchell and Pavlovic [7] is especially relevant to our work. Though they do not propose a high-level language for protocol programming comparable to our language of speech acts, the techniques they study could turn out to be especially useful in the development and optimization of ”protocol compiler”. Abadi work on secrecy by typing [1] seems to be closely related to our result on the secrecy of exchanged keys of well-designed protocols.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their constructive comments and criticisms that have been very helpful in shaping later versions of this paper. The paper has been partially supported by a grant from the Royal Thai Government.

References

[1] M. Abadi Secrecy by typing in security protocols, JACM, 46, 5, 1999, 749-786

[2] M. Abadi, R. Needham. Prudent engineering practices for cryptographic protocols, IEEE Transactions on SE, 22(1): 6-15, 1996

[3] C. Boyd, W. Mao. Designing secure key exchange protocol, Esorics’94, pp 93-105

[4] M. Burrows, M. Abadi, R. Needham. A logic of authentication. ACM Transactions on Computer Systems, 8(1): 18-36, 1990

[5] L. Buttyan, S. Staamann, U. Wilhelm A simple logic for authentication protocol design, Proceedings of the 11th IEEE Computer Security Foundation Workshop, 153-162, 1998

[6] J. Clark, J. Jakob. A survey of authentication protocol literature, version 1, Department of Computer Science, University of York, Nov 1997

[7] A. Datta, A. Derek, J. Mitchell, D. Pavlovic A derivation system for security protocols and its logical formalization , IEEE Computer Security Foundation Workshop, 2003

[8] F. J.T. Fabrega, J.C. Herzog, J.D. Guttman. Strand spaces: Why is a security protocol correct ? Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp 160-171, 1998, IEEE Computer Scociety Press

[9] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi. Reasoning about knowledge. MIT Press, 1995

[10] J.D. Guttman. Security protocol design via authentication test, Proceedings of the 15th IEEE Computer Security Foundation Workshop, 2002

[11] J.D. Guttman, F. J.T. Fabrega Authentication tests and the structure of bundles, Theoretical computer science, 2001

[12] L. Gong, P. Syverson. Fail-stop protocols: An approavch to designing secure protocols, Proceedings of teh 5th International Conference on Dependable Computing for Critical Applications, 1995, pp 44-55

[13] G.E. Hughes, M.J. Cresswell. An introduction to modal logic, Methuen, London and NewYork, 1985

[14] Y. Labrou, T. Finin, Y. Peng. Agent communication languages: The current landscape, IEEE Intelligent Agents, March/April 1999, 45-52

[15] C. Meadows. Formal verification of cryptographic protocols: A Survey, pp 135-150, Asiacrypt 1994,

[16] A. Perrig, D. Song Looking for a diamond in the desert: Extending automatic protocol generation to three-party auhtentication and keyagreement protocols, Proceedings of the 13th IEEE Computer Security Foundation Workshop, 2000

[17] P. Syverson. Towards a strand semantics for authentication logic, Electronic Notes in Theoretical Computer Science, 20,2000

[18] P. Syverson, C. Meadows A formal language for cryptographic protocol requirements. Design, Codes and Cryptography, 7(1 and 2): 27-59, 1996

[19] P. Syverson, P.C. van Oorshot. On unifying some cryptographic protocols, Proceedings of the 1994 IEEE Symposium on Security and Privacy, 14-28

Session V

Language-Based Security I

A Monadic Analysis of Information Flow Security with Mutable State

Karl Crary* Aleksey Kliger Frank Pfenning
{crary,aleksey,fp}@cs.cmu.edu
Carnegie Mellon University

We explore the logical underpinnings of higher-order, security-typed languages with mutable state. Our analysis is based on a logic of information flow derived from lax logic and the monadic metalanguage. Thus, our logic deals with mutation explicitly, with impurity reflected in the types, in contrast to most higher-order security-typed languages, which deal with mutation implicitly via side-effects.

More importantly, we also take a *store-oriented* view of security, wherein security levels are associated with elements of the mutable store. This view matches closely with the operational semantics of low-level imperative languages where information flow is expressed by operations on the store. An interesting feature of our analysis lies in its treatment of upcalls (low-security computations that include high-security ones), employing an "informativeness" judgment indicating under what circumstances a type carries useful information.

1 Introduction

Security-typed languages use a type system to track the flow of information within a program to provide properties such as secrecy and integrity. Secrecy states that high-security information does not flow to low-security agents, and integrity dually states that low-security agents cannot corrupt high-security information. In this paper, we will restrict our attention to secrecy properties. A variety of security-typed languages have been proposed, and several of them are both higher-order (*i.e.*, support first-class functions) and provide mutable state [4, 9, 11, 17].

However, when adopting one of these languages to the Typed Assembly Language [8] setting, one faces a tension between the high-level view of information flowing from the values of sub-terms to the result value of a complete term and the assembly-language imperative view of instructions operating on a mutable store. What is needed

*This material is based on work supported in part by NSF grants CCR-9984812 and CCR-0121633. Any opinions, findings, and conclusions or recommendations in this publication are those of the authors and do not reflect the views of this agency.

is a typed calculus in which values have structure (*i.e.,* like in high level languages) but information flows through the store (*i.e.,* like in a low-level language).

In this paper, we explore this *store-oriented* view of information flow: one of the steps towards a TAL with information flow, we look at a language with a clean separation between values and computations. A suitable starting point is Moggi's monadic metalanguage [6, 7] and its corresponding logic (via a Curry-Howard isomorphism).

Our presentation of lax logic is based on that of Pfenning and Davies [10]. The principal distinctive feature of Pfenning and Davies's account is a syntactic distinction between *terms* and *expressions*, where terms are pure and expressions are (possibly) effectful. They show that this distinction allows the logic to possess some desirable properties (local soundness and local completeness) that state in essence that the logic's presentation is canonical. Although these properties are not particularly important here, the distinction also provides a clean separation between the pure and effectful parts of our analysis, which greatly simplifies our system.

Our system bears some resemblance to the work of Abadi *et al.* [1], who also use a monadic structure to reason about information flow. However whereas we use monads in a conventional manner to separate values from computations, they use a monad to endow values with a security level. It is not clear how to adopt their work to a low-level setting where the values and operations ought to correspond to those of a real machine.

A natural question is whether this store-oriented security discipline limits the expressive power of our account relative to ones based on a value-oriented discipline, but we show (in Section 5) that it does not.

Overview The static semantics of our analysis is based on two typing judgments, one for terms (M) and one for expressions (E). Recall that terms are pure and that security is associated with effects, so the typing judgment for terms makes no mention of security levels. Thus, the typing judgment takes the form $\Sigma; \Gamma \vdash M : A$ (where A is a type, Γ is the usual context and Σ assigns a type to the store).

Expressions, on the other hand, may have effects and therefore may interact with the security discipline. Each location in the store has a security level associated with it indicating the least security level that is authorized to read that location. Thus, the typing judgment for expressions tracks the security levels of all locations an expression reads or writes. Only the reads are of direct importance to the security discipline (recall that we do not address integrity), but writes must also be tracked since they provide a means of information flow. The judgment takes the form: $\Sigma; \Gamma \vdash E \div_{(r,w)} A$ indicating that r is an upper bound to the levels of E's reads, and w is a *lower* bound to the levels of its writes, and also that E has type A. Naturally we require that $r \sqsubseteq w$, or else E could manifestly be leaking information.

So our language can be seen as a conservative extension of purely functional languages such as Haskell. Existing terms continue to be type-safe. On the other hand new effectful code that makes use of the security discipline can be cleanly separated.

In lax logic, expressions are internalized as terms using the monadic type $\bigcirc A$. Terms of type $\bigcirc A$ are suspended expressions of type A. Thus, the introduction form for the monadic type is a term construct, and the elimination form (which releases the suspended expression) is an expression construct. Similarly, our expressions are internalized as terms using a monadic type written $\bigcirc_{(r,w)} A$. Since the effects of the suspended expression will be released when the monad is eliminated, the levels of those effects must be recorded in the monad type.

Most of the rules in our account follow from the intuitions above. One remaining novelty deals with the information content of types. Ordinarily, an expression would be deemed to be leaking information if it were to read from a high-security location, use the result of the read to form a value, and pass that value to a low-security computation. However, that expression would *not* be leaking information if one could show that the type of that value contained no information, or contained information usable only by a high-security computation (who could have performed the read anyway). Thus the type system contains a judgment $\vdash A \nearrow a$ stating that the type A contains information only for computations at the level a at least. This notion of *informativeness* is essential to accounting for the key issue of upcalls (low-security computations that include high-security computations).

The remainder of this paper is organized as follows: In Section 2 we present our basic logical account, including static and dynamic semantics, but omitting the key issue of upcalls. In Section 3 we extend our account to deal with upcalls. In Section 4 we state and prove a non-interference theorem. In Section 5 we show that our store-oriented account provides at least the expressive power of value-oriented accounts by embedding one value-oriented language into our language. Section 6 discusses some related work, Section 7 offers some concluding remarks.

2 The Secure Monadic Calculus

We begin by describing the syntax, evaluation rules and an initial set of typing rules for our language. While this language will be secure, as we will see in the next section its type system rules out too many programs to be practical. By including some additional rules, we will be able to make it more useful while still retaining the required secrecy property.

As in other work on information flow, we have in mind an arbitrary lattice $(\mathcal{L}, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ of security levels.

Operation levels To track the flow of information, we classify expressions not only by the value that they return, but also by the security levels of their effects. In particular, we keep track of an *operation level* $o = (r, w)$, for each expression. The security level r is an upper bound on the security levels of the store locations that the expression

$$
\begin{array}{rcll}
A, B, C & \in & \textit{types} & ::= \quad 1 \mid \mathsf{bool} \mid A \to B \mid \mathsf{ref}_a\, A \mid \mathsf{refr}_a\, A \mid \mathsf{refw}_a\, A \mid \bigcirc_o A \\
M, N & \in & \textit{terms} & ::= \quad x \mid * \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \\
 & & & \qquad\qquad \mid \lambda x : A.M \mid MN \mid \ell \mid \mathsf{val}\ E \\
E, F & \in & \textit{expressions} & ::= \quad [M] \mid \mathsf{let\ val}\ x = M\ \mathsf{in}\ E \mid \mathsf{ref}_a\,(M : A) \mid\, !M \mid M := N \\
\Gamma & \in & \textit{contexts} & ::= \quad \cdot \mid \Gamma, x : A \\
\Sigma & \in & \textit{store types} & ::= \quad \{\} \mid \Sigma\{\ell : A\} \\
V & \in & \textit{values} & ::= \quad * \mid \mathsf{true} \mid \mathsf{false} \mid \lambda x : A.M \mid \ell \mid \mathsf{val}\ E \\
H & \in & \textit{stores} & ::= \quad \{\} \mid H\{\ell \mapsto V\} \\
S & \in & \textit{states} & ::= \quad (H, \Sigma, E)
\end{array}
$$

Figure 1: Syntax

reads, while w is a lower bound on the security level of the store locations to which it writes.

Since expressions that write at a security level below their read level may obviously be insecure, henceforth we restrict our attention only to operation levels (r, w) with $r \sqsubseteq w$.

The operation levels have a natural ordering $(r, w) \preceq (r', w')$. Given some expression E, if it reads from level at most r, then it surely reads from level at most r', provided that $r \sqsubseteq r'$. Similarly, if it writes at level at least w, then it writes at level at least w', provided that $w' \sqsubseteq w$. That is, operation levels are covariant in the reads and contravariant in the writes: $(r, w) \preceq (r', w')$ iff $(r \sqsubseteq r'$ and $w' \sqsubseteq w)$

2.1 Syntax

The full syntax of our language is given in Figure 1. The language is split into two syntactic categories: pure terms M that are evaluated to values V and expressions E that are executed for effect as part of compuation states S.

Terms At the term level, we have variables, unit, booleans and conditional terms, function abstractions and applications. For simplicity, we did not include a mechanism for defining recursive terms, although the inclusion of such a facility would not pose a problem. Store locations are also terms, with each location ℓ having a fixed security level $\mathsf{Level}(\ell)$. The store associates locations with the values they contain. A subtyping relation, allows us to treat store cells as either read-write, read-only, or write-only. The term $\mathsf{val}\ E$ allows expressions to be included at the term level as an element of the monadic type $\bigcirc_o A$. Since terms are pure, a $\mathsf{val}\ E$ does not execute the expression E, but rather represents a suspended computation.

Expressions The expressions include a trivial return expression $[M]$. The return expression has no effect, and simply returns the value to which M evaluates. In general, when an expression has no read effects, we say its read level is $\bot$, and if an expression has no write effects, we say its write level is $\top$. Accordingly, the operation level of $[M]$

is $(\bot, \top)$. Note that $(\bot, \top)$ is the least element in the $\preceq$ ordering, so our subsumption principle will let us weaken the operation level of $[M]$ to any operation level.

The sequencing expression $\mathsf{let}\ \mathsf{val}\ x = M\ \mathsf{in}\ F$ evaluates M down to some $\mathsf{val}\ E$, and executes E followed by F. The return value of expression E is bound to the variable x in F. If E and F both have operation level o, then so does the sequencing expression.

We will often write $\mathsf{let}\ x = E\ \mathsf{in}\ F$ as syntactic sugar for $\mathsf{let}\ \mathsf{val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F$, and $\mathsf{run}\ M$ for $\mathsf{let}\ \mathsf{val}\ y = M\ \mathsf{in}\ [y]$.

In addition, there are expressions that allocate, read from, and write to the store. A read expression $!M$ has operation level $(a, \top)$, where a is the security level of the store location being read, and returns the contents of the store location. Dually, a write expression $M := N$ has operation level $(\bot, a)$ and updates the store location with the value of N; it does not return an interesting value (*i.e.*, it returns unit).

Store allocation $\mathsf{ref}_a\,(M : A)$ specifies the security level a and type A of the new store location.

Allocation cannot leak information. Evidently, it is not a read operation. Less obviously, it is not a write operation either. With a write, another expression may learn something about the current computation by observing a change in the value stored at a particular store location. However, the key to this scenario is that the same location is mentioned by more than one expression. On the other hand, allocation creates a new location that is not aliased. Thus, there can be no implicit flow of information via an allocation expression. As a result, allocation has operation level $(\bot, \top)$. Of course if there were a primitive mechanism in place to distinguish locations (for example by comparing locations for equality), allocation would once again be observable.

Although there is not a primitive mechanism for recursion at the level of expressions, recursion can be encoded at the level of expressions using back-patching, see an example in Section 3.2.

States A computation state is a partially executed program, and consists of a triple (H, Σ, E) of a store H, a store type Σ and a closed expression E. The store maps locations to values, and the store type maps locations to the types of those values.

We assume that in a state (H, Σ, E), the store binds occurrences of store locations ℓ in H and E, and we identify computation states up to level-preserving renaming of store locations. In addition, as usual, we identify all constructs up to renaming of bound variables.

2.2 Static Semantics

The type system of our language consists of two main mutually recursive judgments for typing terms and expressions, and some judgments for typechecking stores, and computation states. The first judgment $\Sigma; \Gamma \vdash M : A$ says that the term M has

$\boxed{\Sigma; \Gamma \vdash M : A}$

$$\frac{}{\Sigma; \Gamma \vdash x : \Gamma(x)}\ (1) \qquad \frac{}{\Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)}\ (2)$$

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{bool} \quad \Sigma; \Gamma \vdash N_1 : A \quad \Sigma; \Gamma \vdash N_2 : A}{\Sigma; \Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A}\ (3)$$

$$\frac{\Sigma; \Gamma, x : A \vdash M : B}{\Sigma; \Gamma \vdash \lambda x : A.M : A \to B}\ (4) \qquad \frac{\Sigma; \Gamma \vdash M : A \to B \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash M\ N : B}\ (5)$$

$$\frac{\Sigma; \Gamma \vdash E \div_o A}{\Sigma; \Gamma \vdash \mathsf{val}\ E : \bigcirc_o A}\ (6) \qquad \frac{\Sigma; \Gamma \vdash M : A \quad \vdash A \leq B}{\Sigma; \Gamma \vdash M : B}\ (7)$$

Figure 2: Typing rules (terms).

type A in the context Γ, where the store has type Σ. The jugment for expressions $\Sigma; \Gamma \vdash E \div_o A$ says that E returns a value of type A and performs only operations within level o.

We assume that contexts Γ are well-formed, that is, they contain at most one occurrence of each variable x. We tacitly rename bound variables prior to adding them to a context to maintain well-formedness. Similarly, we assume that store types are well-formed, that is, they contain at most one occurrence of each store location ℓ.

Terms The typing rules for terms are unsurprising for a simply-typed lambda calculus with unit, bool and function types. A store location ℓ (provided that it is in $\mathrm{dom}(\Sigma)$) has type $\mathsf{ref}_{\mathsf{Level}(\ell)}\Sigma(\ell)$. A computation term $\mathsf{val}\ E$ has the type $\bigcirc_o A$, provided the expression E has type A and operation level o. The rules are summarized in figure 2.

Expressions The typing rules for expressions (given in Figure 3) follow our informal description. Trivial computations have the type of their return value, and operation level $(\bot, \top)$ (rule 8). By rule (9), the sequencing expression $\mathsf{let}\ \mathsf{val}\ x = M\ \mathsf{in}\ E$ is well-typed provided both of the sub-computations have the same operation level (which may require using rule (13) to weaken the operation level of the sub-computations). Allocation (rule 10) returns a new read/write store location. For read and write expressions (rules 11 and 12) we only require that the corresponding store location is readable or writable, respectively.

Subtyping Subsumption (rules 7, 14) allows us to weaken the type A of a term M or an expression E, provided A is a subtype of B. Selected subtyping rules are given in Figure 4.

$\boxed{\Sigma; \Gamma \vdash E \div_o A}$

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash [M] \div_{(\bot,\top)} A} \ (8) \quad \frac{\Sigma; \Gamma \vdash M : \bigcirc_o A \quad \Sigma; \Gamma, x : A \vdash E \div_o A}{\Sigma; \Gamma \vdash \mathsf{let\ val}\ x = M\ \mathsf{in}\ E \div_o A} \ (9)$$

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash \mathsf{ref}_a\ (M : A) \div_{(\bot,\top)} \mathsf{ref}_a\ A} \ (10) \quad \frac{\Sigma; \Gamma \vdash M : \mathsf{refr}_a\ A}{\Sigma; \Gamma \vdash !M \div_{(a,\top)} A} \ (11)$$

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{refw}_a\ A \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash M := N \div_{(\bot,a)} 1} \ (12) \quad \frac{\Sigma; \Gamma \vdash E \div_o A \quad o \preceq o'}{\Sigma; \Gamma \vdash E \div_{o'} A} \ (13)$$

$$\frac{\Sigma; \Gamma \vdash E \div_o A \quad \vdash A \leq B}{\Sigma; \Gamma \vdash E \div_o A} \ (14)$$

Figure 3: Typing rules (expressions).

$\boxed{\vdash A \leq B}$

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{refr}_a\ A \leq \mathsf{refr}_b\ B} \ (15) \quad \frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{refw}_a\ A \leq \mathsf{refw}_b\ B} \ (16)$$

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{ref}_a\ A \leq \mathsf{refr}_b\ B} \ (17) \quad \frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{ref}_a\ A \leq \mathsf{refw}_b\ B} \ (18)$$

$$\frac{\vdash A \leq B \quad o \preceq o'}{\vdash \bigcirc_o A \leq \bigcirc_{o'} B} \ (19)$$

Figure 4: Selected subtyping rules.

$\boxed{S \to S'}$

$$\frac{M \to M'}{(H, \Sigma, \mathsf{let\ val}\ x = M\ \mathsf{in}\ E) \to (H, \Sigma, \mathsf{let\ val}\ x = M'\ \mathsf{in}\ E)}\ \textsc{LetVal1} \qquad \frac{(H, \Sigma, E) \to (H', \Sigma', E')}{(H, \Sigma, \mathsf{let\ val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F) \to (H', \Sigma', \mathsf{let\ val}\ x = \mathsf{val}\ E'\ \mathsf{in}\ F)}\ \textsc{LetValVal}$$

$$\frac{}{(H, \Sigma, \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ E) \to (H, \Sigma, E[V/x])}\ \textsc{LetVal}$$

$$\frac{\ell \notin \mathrm{dom}(H) \quad \mathsf{Level}(\ell) = a}{(H, \Sigma, \mathsf{ref}_a\ (V : A)) \to (H\{\ell \mapsto V\}, \Sigma\{\ell : A\}, [\ell])}\ \textsc{Ref} \qquad \frac{}{(H, \Sigma, !\ell) \to (H, \Sigma, [H(\ell)])}\ \textsc{Bang}$$

$$\frac{\ell \in \mathrm{dom}(H)}{(H, \Sigma, \ell := V) \to (H\{\ell \mapsto V\}, \Sigma, [*])}\ \textsc{Assn} \qquad \frac{M \to M'}{(H, \Sigma, [M]) \to (H, \Sigma, [M'])}\ \textsc{Ret1}$$

Figure 5: Operational Semantics (Expressions, selected rules)

Stores and states A store H is well-typed with store type Σ, provided that each value V_i in the store is well typed under Σ and the empty context, where Σ has the same domain as H. A computation state (H, Σ, E) is well-typed provided that the store and the expression are each well-typed with the same store type:

$$\frac{\mathrm{dom}(\Sigma) = \{\ell_1, \ldots, \ell_n\} \quad \Sigma; \cdot \vdash V_i : \Sigma(\ell_i)\ \text{ for } 1 \leq i \leq n}{\vdash \{\ell_1 \mapsto V_1, \ldots \ell_n \mapsto V_n\} : \Sigma} \quad (20)$$

$$\frac{\vdash H : \Sigma \quad \Sigma; \cdot \vdash E \div_o A}{\vdash (H, \Sigma, E) \div_o A} \quad (21)$$

2.3 Operational Semantics and Safety

A computation state is called *terminal* if it is of the form $(H, \Sigma, [V])$. An evaluation relation $S \to S'$ gives the small-step operational semantics for computation states. We write $S \downarrow$ if for some terminal state S', $S \to^* S'$. Since terms are pure, their evaluation rules may be given simply by the relation $M \to M'$ (no store is required). The evaluation rules for terms are entirely standard, and are omitted. The evaluation rules for expressions are given in Figure 5. We write $M[N/x]$ and $E[N/x]$ for the capture-avoiding substitution of N for x in the term M or expression E. We write $H\{\ell \mapsto V\}$ for finite map that extends H with V at ℓ.

A computation state S is *stuck* if it is not terminal and there is no S' such that $S \to S'$. In the extended version of the paper [2] we show the expected type safety

theorem: whenever S is well-typed, if $S \rightarrow^* S'$, then S' is not stuck.

3 Upcalls

Although the approach discussed so far is secure, it falls short of a practical language. There is no way to include a computation that reads from the high-security store in a larger low security computation. In any program with a high security read, the read level of the entire program is pushed up. However, many programs that contain *upcalls* to high security computations followed by low security code are secure.

Consider the program $\mathsf{let}\ z = P\ \mathsf{in}\ E$ where $P \div_{(\top,\top)} 1$ and E has operation level $(\bot, \bot)$. P does not leak information because 1 carries no useful information, and P's writes are above E's reading level. Thus we would like to give the entire program the operation level $(\bot, \bot)$. However the type system we have presented so far would instead promote the operation level of E and the entire program to $(\top, \top)$.

In order to have a logic of information flow, we must offer an account of upcalls. Indeed, the power to perform high security computations interspersed in a larger low-security computation is the *sine qua non* of useful secure programming languages. We offer a detailed analysis of two cases where upcalls do not violate our intuitive notion of security. From these examples, we develop a general principle for treating upcalls — our notion of *informativeness* — discussed in Section 3.2. We take up the question of non-interference in Section 4.

3.1 A more general example

Now consider a computation E with operation level (r, w), but this time, suppose that E has type $\mathsf{ref}_a\, B$ for some type B. Are there any situations where E may be given a different operation level?

Suppose that $r \sqsubseteq a$. In that case, any computation that may read the $\mathsf{ref}_a\, B$ is also able to read any store locations that E may read. Again, any computation can either do what E does itself, or it cannot gain information from E's return value.

On the other hand, consider the case where $r \not\sqsubseteq a$. The particular value of type $\mathsf{ref}_a\, B$ that E returns may carry information from store locations at security level r. For example, E may return one of two such store locations ℓ_1 or ℓ_2 from level a based on some boolean value V from a store location at security level r. In that case, a computation that reads at security level a may learn something about E's reads (at level r) by reading from E's return value. Since $r \not\sqsubseteq a$, this represents a violation of secure information flow.

So if E returns a $\mathsf{ref}_a\, B$, we can demote its reading level whenever $r \sqsubseteq a$, because any computation that wishes to make use of that return value would need a read level of at least r. In other words, a $\mathsf{ref}_a\, B$ is informative only to computations that may read at least at some security level (namely a) above r.

$\boxed{\vdash A \nearrow a}$

$$\frac{}{\vdash A \nearrow \bot}\ (23) \quad \frac{\vdash A \nearrow a \quad b \sqsubseteq a}{\vdash A \nearrow b}\ (24) \quad \frac{\vdash A \nearrow a \quad \vdash A \nearrow b}{\vdash A \nearrow a \sqcup b}\ (25) \quad \frac{\vdash B \nearrow a}{\vdash A \to B \nearrow a}\ (26)$$

$$\frac{}{\vdash \mathsf{ref}_a\, A \nearrow a}\ (27) \quad \frac{\vdash A \nearrow a}{\vdash \mathsf{ref}_b\, A \nearrow a}\ (28) \quad \frac{\vdash A \nearrow a}{\vdash \mathsf{refr}_b\, A \nearrow a}\ (29)$$

$$\frac{}{\vdash \mathsf{refr}_b\, A \nearrow b}\ (30) \quad \frac{}{\vdash \mathsf{refw}_a\, A \nearrow a}\ (31) \quad \frac{\vdash A \nearrow a}{\vdash \bigcirc_{(r,w)} A \nearrow w \sqcap a}\ (32)$$

Figure 6: Informativeness judgement.

This observation suggests a new subsumption rule for expressions that alters the operation level:

$$\frac{\Sigma; \Gamma \vdash E \div_{(r,w)} A \quad \vdash A \nearrow r}{\Sigma; \Gamma \vdash E \div_{(\bot,w)} A}\ (22)$$

where the new *informativeness* judgment $\vdash A \nearrow r$ formalizes the idea that values of type A, if they are informative at all, are informative only at level r or above.[1]

In terms of this new judgment, our earlier observations are that $\vdash 1 \nearrow r$ for any r, and $\vdash \mathsf{ref}_a\, A \nearrow r$ whenever $r \sqsubseteq a$.

3.2 Informativeness

We now consider some properties of the new judgment $\vdash A \nearrow a$ (see Figure 6). Several structural rules (23,24,25) for the judgment are immediate. If A is informative at all, then it's informative only at $\bot$ or above. Also, if A is informative only at or above a and if $b \sqsubseteq a$, then A is informative only at or above b. That is, we may choose to discard some knowledge about when a type is informative. Finally, suppose A is informative only above a, and A is informative only above b. Then for any r if values of type A are informative to computations that read at r, we know that both $a \sqsubseteq r$ and $b \sqsubseteq r$. Therefore, for any such r, $a \sqcup b \sqsubseteq r$. So, A is informative only above $a \sqcup b$.

A value of type bool is informative for any computation at all, since it may be trivially analyzed with a conditional. So aside from the structural axiom $\vdash A \nearrow \bot$, there should be no other rules for bool. We would give a similar account of other types that may be analyzed by branching (*e.g.*, sum types $A + B$ or integers int).

Since functions are used by application, a value of type $A \to B$ is useful exactly when B is.

We have already alluded to rule (27) for $\mathsf{ref}_a\, A$. There is an additional rule for refs. Even if a computation can read from a store location of type $\mathsf{ref}_b\, A$ (*i.e.*, its read level

[1] Informativeness is closely related to *protectedness* in DCC [1] and to the tampering levels of [5]. We discuss the relationship in Section 6.

```
λc : ◯(⊤,⊤)bool.
 val let wref = ref⊤ (val [∗] : ◯(⊥,⊤)1) in
  let w = [val (let b = run c in run (if b then val (let w′ = !wref in run w′) else val [∗]))] in
  let _ = wref := w in
  run w
```

Figure 7: $untilFalse : \bigcirc_{(\top,\top)}\mathsf{bool} \to \bigcirc_{(\bot,\top)}1$

is above b), only if A is informative at its operation level, can $\mathsf{ref}_b\ A$ be informative.

Read-only store locations are useful only to computations that may read from them. Consequently, by an argument similar to the one for read-write store cells, they have analogous rules.

For write-only store cells $\mathsf{refw}_a\ A$, we have to consider aliasing. One way that a computation may learn whether two store locations are aliases is by writing a known value to one of them, and then reading out the value from the other. Because of subtyping, if a lower-security computation has a store location ℓ of type $\mathsf{refr}_a\ A$, a value of type $\mathsf{refw}_a\ A$ may be informative if the computation can read from (the seemingly unrelated) ℓ.

Finally, consider the type $\bigcirc_{(r,w)}A$. A value of this type is informative both to computations that may read at least security level w (that is, the level the suspended expression writes to), and to computations for which the type A is informative.

With informativeness in hand, many more useful terms become well-typed. Consider, for example, the term in Figure 7. The function $untilFalse$ takes as argument a computation that reads and writes high before returning a boolean, and runs that computation repeatedly until it returns false. Recursion is accomplished using backpatching: a store location with a dummy value is allocated and is bound to $wref$, recursive calls in the body of the loop dereference $wref$ and run the contents. The recursive knot is tied by overwriting the contents of $wref$ with the real loop body w.

Interestingly, although $untilFalse$ takes a high-security computation as an argument, our type system is able to give it the type $\bigcirc_{(\top,\top)}\mathsf{bool} \to \bigcirc_{(\bot,\top)}1$, that is its return type is a low-security computation. Intuitively, even if f is a high-security computation, $untilFalse\ f$ does not leak any information to low-security since any information gained from f's return value is used only within the loop.

4 Non-interference

Informally, non-interference says that computations that have a low read level do not depend on values in high security store locations. As in similar arguments [17, 16], "low" means below some fixed security level ζ, and "high" means not below ζ.

Operationally, the low security sub-computations of a program should behave identically irrespective of the values in the high security store locations. On the other hand, it is alright for high security sub-computations to behave differently depending on values in high security store locations. However once a high security sub-computation completes, the low security behavior should again be identical modulo the parts of the computation state that are "out of view" of the low security part of the program.

4.1 Equivalence property

Formally, we define an equivalence property of computation states (and term and expressions) such that two states are equivalent whenever they agree on the "in view" parts of the computation state. Then, in the style of a confluence proof, we show that this equivalence property is preserved under evaluation.

Stores and States Certainly values in high security store locations are out of view. Less obviously, some values in the low security locations are out of view as well: if a low security store location appears only out of view, its value is also out of view. We parametrize the store equivalence judgment by a set U of in view store locations. Two (well-typed) stores are equivalent only if their in view values are equivalent:

$$\frac{\vdash H_1 : \Sigma_1 \quad \vdash H_2 : \Sigma_2 \quad \Sigma_1 \restriction U = \Sigma_2 \restriction U \quad \Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell) \text{ for } \ell \in U}{\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)} \quad (33)$$

Where the notation $\Sigma \restriction X$ means Σ restricted to locations in the set X.

For a pair of computation states, only low security locations that are common to both computations are in view. Since allocation does not leak information, it is possible for two programs to allocate different low security locations while executing high security sub-computations. However such locations are out of view for the low security sub-computation.

Pairs of computation states are equivalent if their stores are equivalent on the in-view locations, and if they have equivalent expressions (where $\downarrow(\zeta) = \{\ell \mid \mathsf{Level}(\ell) \sqsubseteq \zeta\}$ is the set of all low security locations) :

$$\frac{\vdash (H_1 : \Sigma_1) \approx_\zeta^{\mathrm{dom}(H_1) \cap \mathrm{dom}(H_2) \cap \downarrow(\zeta)} (H_2 : \Sigma_2) \quad \Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o A}{\vdash (H_1, \Sigma_1, E_1) \approx_\zeta (H_2, \Sigma_2, E_2) \div_o A} \quad (34)$$

Terms and Expressions High security sub-computations of a program may return different values to the low security sub-computations. However, by the upcall rule, the type of those values must be informative only at high security.

Values of a type that is informative only at high security are out of view. As a result, any two values of such a type are equivalent since two such values vacuously agree on their in view parts:

$$\frac{\Sigma_1; \Gamma \vdash V_1 : A \quad \Sigma_2; \Gamma \vdash V_2 : A \quad \vdash A \nearrow a \quad a \not\sqsubseteq \zeta}{\Sigma_1; \Sigma_2; \Gamma \vdash V_1 \approx_\zeta V_2 : A} \ (35)$$

The remaining rules for term and expression equivalence are congruence rules that merely require corresponding sub-terms or sub-expressions to be equivalent. They are given in the extended paper.

4.2 Non-interference theorem

The main result necessary to establish non-interference is the so-called "Hexagon Lemma" : given two equivalent computation states that each take a step, we show that in zero or more steps we can reach two computation states that are again equivalent.

As previously noted, while a program is executing a high security sub-computation, it may behave differently based on the contents of the high-security store. However, as the following preliminary lemma shows, during any such high security steps, the in view parts of the stores remain equivalent.

Lemma 4.1 (High Security Step (HSS)). *Given two states* (H_1, Σ_1, E_1) *and* (H_2, Σ_2, E_2) *such that* $\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)$ *where* $U = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap \downarrow(\zeta)$, *and for* $i = 1, 2$, *there exist* C_i *and* $o_i = (r_i, w_i)$ *such that* $\Sigma_i; \cdot \vdash E_i \div_{o_i} C_i$ *and* $w_i \not\sqsubseteq \zeta$. *If* $(H_i, \Sigma_i, E_i) \rightarrow^* (H_i', \Sigma_i', E_i')$ *for* $i = 1, 2$ *then* $\vdash (H_1' : \Sigma_1') \approx_\zeta^{U'} (H_2' : \Sigma_2')$ *where* $U' = \mathrm{dom}(\Sigma_1') \cap \mathrm{dom}(\Sigma_2') \cap \downarrow(\zeta)$

The proof of this lemma appears in the extended version of the paper [2].

Lemma 4.2 (Hexagon Lemma). *For all* ζ, *if* $o = (r, w)$ *with* $r \sqsubseteq \zeta$, *and if* $\vdash S_1 \approx_\zeta S_2 \div_o C$ *and* $S_1 \rightarrow S_1'$, $S_2 \rightarrow S_2'$ *where* $S_1' \downarrow$ *and* $S_2' \downarrow$ *then there exist* S_1'', S_2'' *such that* $S_1' \rightarrow^* S_1''$, $S_2' \rightarrow^* S_2''$ *and* $\vdash S_1'' \approx_\zeta S_2'' \div_o C$

Proof. By Inversion on $\vdash S_1 \approx_\zeta S_2 \div_o C$, we get that each computation state S_i is a triple (H_i, Σ_i, E_i), and that the two stores and the two expressions are equivalent $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$. We prove the theorem by induction on this derivation. We consider one case below, the remaining cases are proved in the extended paper [2].

Case: $$\frac{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(r', w)} C \quad \vdash C \nearrow r'}{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(\bot, w)} C} \ (36)$$

If $r' \sqsubseteq \zeta$, we can invoke the induction hypothesis to get two equivalent computation states with operation level (r', w), and then use the upcall rule to construct the desired derivation (with operation level (r, w)).

On the other hand, if $r' \not\sqsubseteq \zeta$, then since $r' \sqsubseteq w$, it follows that $w \not\sqsubseteq \zeta$ and so, by the High Security Step Lemma, running both of the computation states to completion produces equivalent stores. Since we also know that their return values are out of view, we can show that the resulting terminal states are equivalent. □

$$
\begin{array}{rcll}
t & \in & \textit{types} & ::= \quad 1 \mid \mathsf{bool} \mid s_1 \xrightarrow{\mathsf{pc}} s_2 \mid \mathsf{ref}\ s \\
s & \in & \textit{security types} & ::= \quad (t, a) \\
bv & \in & \textit{base values} & ::= \quad * \mid \mathsf{true} \mid \mathsf{false} \mid \ell \mid \lambda[\mathsf{pc}]x : s.e \\
e & \in & \textit{expressions} & ::= \quad x \mid bv_a \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mid e_1 e_2 \mid \mathsf{ref}\ (e : s)) \mid\ !e \mid e := e'
\end{array}
$$

Figure 8: $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ Syntax

Theorem 4.3 (Non-interference). *If* $\vdash H : \Sigma$ *and* $\Sigma; x : A \vdash E \div_{(r,w)} B$ *and if* $\Sigma; \Sigma; \cdot \vdash V_1 \approx_r V_2 : A$ *then if* $(H, \Sigma, E[V_1/x]) \rightarrow^* S_1$ *and* $(H, \Sigma, E[V_2/x]) \rightarrow^* S_2$ *and both* S_1, S_2 *are terminal, then* $\vdash S_1 \approx_r S_2 \div_{(r,w)} B$

Proof. By some easy structural properties, we can show that $\Sigma; \Sigma; \cdot \vdash E[V_1/x] \approx_r E[V_2/x] \div_{(r,w)} B$. By repeated application of the Hexagon Lemma, the two computations evaluate to equivalent terminal states. Since the operational semantics are deterministic (upto renaming of bound store locations), those terminal states are S_1 and S_2, respectively. □

5 Encoding a value-oriented language

A natural question is whether we sacrifice expressive power in comparison to value-oriented secure languages. In such languages, terms are classified by security types: pairs of an ordinary type and a security level. The type system ensures that each term is assigned a security level at least as high as the security level of the terms contributing to it. In our account only the store provides security. We consider the language $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ (summarized in Figure 8) of Zdancewic [15] and show that it can be encoded into our language.

In an imperative setting, information gained via control-flow may leave an expression non-locally (*e.g.,* via a write to the store). As a result, it becomes necessary to track such *implicit flows* of information. Secure imperative languages use a so-called *program counter security level*, pc, as a lower bound on the information that a computation may gain via control flow. Consequently, the results and effects of each expression must be at least as secure as any information gained via control flow.

The typing rules for $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ are unsurprising for a value-oriented language (see the extended paper). The rule for lambda captures the program counter annotation in the arrow type, and an application expression releases the effects provided that they do not leak information through control flow or the return value.

$$\frac{\Sigma; \Gamma, x : s[\mathsf{pc}] \vdash e : s'}{\Sigma; \Gamma \vdash \lambda[\mathsf{pc}]x : s.e\ : s \xrightarrow{\mathsf{pc}} s'}$$

$$\frac{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 : (s' \xrightarrow{\mathsf{pc}'} s, a) \quad \Sigma; \Gamma[\mathsf{pc}] \vdash e_2 : s' \quad \mathsf{pc} \sqcup a \sqsubseteq \mathsf{pc}'}{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 e_2 : s \sqcup a}$$

Encoding In order to emulate the sealing behavior of value-oriented languages in our store-oriented discipline, we embed source-language values of security type $s = (t, a)$ into read-only refs in our language $\overline{s} = \mathsf{refr}_a\, \overline{t}$.

In a $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ function of type $s \xrightarrow{\mathsf{pc}} s'$ the program counter annotation pc is a conservative approximation of the information gained by the body of the function. Therefore, values written by the body must have security level at least pc. Thus, the corresponding writes in the translation must have write level at least pc. Consequently, the corresponding translated type for a function is $\overline{s} \to \bigcirc_{(\bot,\mathsf{pc})}\, \overline{s'}$.

The encoding for $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ expressions is given by a pair of judgments $\Sigma; \Gamma \vdash bv : t \Rightarrow M$ and $\Sigma; \Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E$, given in the extended paper. In [2] we show that the translation preserves typing, that is (extending $\overline{\cdot}$ pointwise to store types and to contexts) whenever $\Sigma; \Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E$, it follows that $\overline{\Sigma}; \overline{\Gamma} \vdash E \div_{\bot,\mathsf{pc}} \overline{s}$.

Non-interference Of course a type correct (but insecure) embedding could be constructed by ignoring the security levels of the source and placing everything at level $\bot$. We wish to show that the embedding is actually secure. To do so, we show that an instance of non-interference for $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ is preserved by our translation.

Theorem 5.1 ($\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ non-interference). *Suppose* $\Sigma_0; x : (t, a)[b] \vdash f : (\mathsf{bool}, b) \Rightarrow F$ *where* $a \not\sqsubseteq b$, *and suppose that* H, Σ *are such that* $\Sigma \supseteq \overline{\Sigma_0}$, *and* $\vdash H : \Sigma$. *If* $\Sigma; \cdot \vdash \ell_i : \mathsf{refr}_a\, \overline{t}$ *for* $i = 1, 2$ *and if there exist* $H_1, H_2, \Sigma_1, \Sigma_2, V_1, V_2$ *such that* $(H', \Sigma', F[\ell_i/x]) \to^* (H_i, \Sigma_i, [V_i])$ *for* $i = 1, 2$, *then* $V_i = \ell'_i$ *and* $H_1(\ell'_1) = H_2(\ell'_2)$ *as booleans.*

Proof. From the type-correctness of the translation, and since the argument locations ℓ_i are out of view, by the non-interference theorem we conclude that $\vdash (H_1, \Sigma_1, [V_1]) \approx_b (H_2, \Sigma_2, [V_2]) \div_{(b,b)} \mathsf{refr}_b\, \mathsf{bool}$.

By inversion and by a canonical forms lemma, each V_i must be some store location $\ell'_i \in \mathrm{dom}(\Sigma_i)$ and $\Sigma_1; \Sigma_2; \cdot \vdash \ell'_1 \approx_b \ell'_2 : \mathsf{refr}_b\, \mathsf{bool}$. Since each $\Sigma_i(\ell'_i)$ must be a subtype of $\mathsf{refr}_b\, \mathsf{bool}$, each $\mathsf{Level}(\ell'_i)$ must be below b, and the two locations must be in-view. Therefore, $\ell'_1 = \ell'_2$ and furthermore, the values in those locations must, in turn, be equivalent $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell'_1) \approx_b H_2(\ell'_2) : \mathsf{bool}$. Since bool is informative at any security level, by inversion, it must be the case that $H_1(\ell'_1) = H_2(\ell'_2)$. □

6 Related Work

There is a large body of existing work on type systems for secure information flow. Volpano, Smith and Irvine [14] first showed how to formulate an information flow analysis as a type system. An excellent survey by Sabelfeld and Myers [12] outlines the key ideas in the design of secure programming languages.

Our account is most related to the Dependency Core Calculus [1]. Like our language, DCC uses a family of monads to reason about information flow. However in

DCC, terms of monadic type are used to seal up values at a security level. In our account, monads are used in a more traditional role as a means of threading state through a program. Central to DCC is the notion of *protectedness* of a type at a security level. If T is protected at a then T is at least as secure as a. This is closely related to our notion of informativeness.

When viewed through the lens of the encoding of (a pure subset of) $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$, the two relations serve the same purpose, ensuring that a computation's output is at least as secure as its inputs. In DCC, this is done directly. In our account, this occurs indirectly: to access a value carrying information only at a particular level, a computation must adopt a read level at least as high. (However, our account also offers the facility — not employed in the $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ embedding — not to seal all computations' return values in order to obtain a $\bot$ effective read level).

The definitions of protectedness and informativeness are the same on the standard type operators, but do not include the idiosyncratic cases: our language has no analog of DCC's monad, nor does DCC contain references or a traditional (*i.e.,* effects-oriented) monad. Moreover, if it did, we conjecture that DCC's definition for these would be somewhat different from ours. Nevertheless, the similarity between the two suggests that our account might be profitably combined with DCC to produce a language capable of expressing security in both value-oriented and store-oriented fashions.

A further similarity exists between the *tampering levels* of Honda and Yoshida [5] and informativeness. They work in a concurrent setting of a typed π-calculus, and the tampering level of a process represents the least security level that may observe the effects of a process of a given type. They present a calculus in the style of [13] extended with local variables, reference types and higher-order procedures and a translation of it into their typed process calculus. Much of the complexity of their language stems from tracking the action set of a command, that is, the references (conflated with program variables) that a command may read or write. Our language may be seen as a restatement of their language in a more conventional monadic style. In the setting of [5], our upcall rule (exploiting the informativeness judgment) would correspond to leaving out the information that a command read from some variables from its action set whenever the command does not tamper below a certain security level.

Harrison *et al.* [3] observed that monads and monad transformers may be used to separate pieces of the state with different security levels, thus ensuring a kind of non-interference via the monad laws. However by construction their system does not allow computations to access any state with a different security level.

7 Conclusion

We give an account of secure information flow in the context of a higher-order language with mutable state. Moreover, motivated by a low-level store-oriented view of

computation, we arrive at a view of security based on lax logic. Rather than sealing values at a security level, we instead associate security with the store. A family of monadic types is used to keep track of the effects of computations. To account for upcalls, we classify the informativeness of types at particular security levels.

Since we treat terms apart from the effectful expressions, our approach can straightforwardly encompass additional type constructors. The question of how to account for additional effects requires further work. From the point of view of non-interference, effects introduce the possibility of different behavior from seemingly related expressions. We expect that by further refining the monadic type to restrict the behavior of related terms, we may be able to account for effects such as I/O or non-local control transfers.

Certain complications beyond those discussed in this paper remain in developing a typed assembly language that tracks information flow. One problem to be dealt with is the re-use of registers between low-security and high-security computations. Any mutation of a register by a high security computation could potentially be observed once it returns to a low-security caller. As a result it is necessary to exploit informativeness to ensure that the contents of registers are not informative to the caller. We conjecture that informativeness in conjunction with linear continuations [17] will prove invaluable to the design of a secure TAL.

Our formulation of the monadic language is in the style of Pfenning and Davies [10]. One avenue of future work is to study whether there is a formulation of information flow in a modal logic that decomposed our monad into the possibility and necessity modalities.

Acknowledgments Thanks to Matthew Harren and Steve Zdancewic for their comments and suggestions on earlier drafts of this paper.

References

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Twenty-Sixth ACM Symposium on Principles of Programming Languages*, pages 147–160, San Antonio, Texas, Jan. 1999.

[2] K. Crary, A. Kliger, and F. Pfenning. A monadic analysis of information flow security with mutable state. *Journal of Functional Programming*, 2004. To Appear. An earlier version is available as Carnegie Mellon University, School of Computer Science technical report CMU-CS-03-164.

[3] W. Harrison, M. Tullsen, and J. Hook. Domain separation by construction. In *Foundations of Computer Security Workshop(FCS'03)*, Ottowa, Canada, June 2003.

[4] N. Heintze and J. G. Riecke. The SLam calculus: Programming with secerecy and integrity. In *Twenty-Fifth ACM Symposium on Principles of Programming Languages*, pages 365 – 377, San Diego, California, Jan. 1998.

[5] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Twenty-Ninth ACM Symposium on Principles of Programming Languages*, pages 81–92, Jan. 2002.

[6] E. Moggi. Computational lambda-calculus and monads. In *Fourth IEEE Symposium on Logic in Computer Science*, pages 14–23, 1989.

[7] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[8] G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):527–568, May 1999. An earlier version appeared in the 1998 Symposium on Principles of Programming Languages.

[9] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Twenty-Sixth ACM Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, Texas, Jan. 1999.

[10] F. Pfenning and R. Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[11] F. Pottier and V. Simonet. Information flow inference for ML. *ACM Transactions on Programming Languages and Systems*, 25(1):117–158, Jan. 2003.

[12] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5 – 19, Jan. 2003. special issue on Formal Methods in Security.

[13] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. *Twenty-Fifth ACM Symposium on Principles of Programming Languages*, pages 355 – 364, 1998.

[14] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[15] S. Zdancewic. *Programming Languages for Information Security*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 2002.

[16] S. Zdancewic and A. C. Myers. Secure information flow and CPS. In *Tenth European Symposium on Programming*, volume 2028 of *Lecture Notes in Computer Science*, pages 46 – 61. Springer-Verlag, Apr. 2001.

[17] S. Zdancewic and A. C. Myers. Secure information flow via linear continuations. *Higher Order and Symbolic Computation*, 15(2-3):209–234, Sept. 2002.

Session VI

Language-Based Security II

A Modal Foundation for Secure Information Flow

Kenji Miyamoto Atsushi Igarashi
Graduate School of Informatics, Kyoto University
{miyamoto, igarashi}@kuis.kyoto-u.ac.jp

Abstract

Information flow analysis is a program analysis that detects possible illegal information flow such as the leakage of (partial information on) passwords to the public. Recently, several type-based techniques for information flow analysis have been proposed for various kinds of programming languages. Details of those type systems, however, vary substantially and even their core parts are often slightly different, making the essence of the type system unclear.

In this paper we propose a typed lambda calculus $\lambda_s^{\Box}$ as a foundation for information flow analysis. The type system is developed so that it corresponds to a proof system of an intuitionistic modal logic of validity by the Curry-Howard isomorphism. The calculus enjoys the properties of subject reduction, confluence, and strong normalization. Moreover, we give a very simple proof of the noninterference property, which guarantees that, in a well-typed program, no information on confidential data is leaked to the public. We also demonstrate that a core part of the SLam calculus by Heintze and Riecke can be encoded into $\lambda_s^{\Box}$.

Keywords: Curry-Howard isomorphism, information flow analysis, modal logic, noninterference, and type systems.

1 Introduction

Background. Increasing demands for security in software have recently been stimulating the research on language-based security. Among such work is a program analysis technique called *information flow analysis* [8, 24], which statically checks whether or not secret information, such as passwords, is leaked by program execution.

Information flow analysis keeps track of how information on confidential data flows. In the literature, information flow is classified into two: *explicit* and *implicit* flow. Explicit flow occurs when, for example, confidential data are assigned to public variables while implicit flow arises from the control structure of a program: it occurs when confidential data control which conditional branch to take. For example, consider the following program fragment:

```
pub := if length(passwd) > 5 then 1 else 2;
```

and suppose `pub` is a public variable, `passwd` is a confidential string, and `length` is a function to compute the length of a given string. In information flow analysis, this program is considered insecure—even though `passwd` itself is not leaked, its partial information, that is, whether `passwd` is more than five characters, is leaked to `pub`. A principal correctness property of information flow analysis is called *noninterference*. This property intuitively means that, given a program that takes a confidential input and yields a public output, the output remains the same no matter what value is given as the input.

Recently, a lot of *type-based* techniques for information flow analysis have been proposed for various kinds of languages, including procedural [27, 26], functional [10, 1, 23], object-oriented [19, 3, 4] and, concurrent languages [25, 11, 12, 22, 15]. The basic idea of type-based analysis is as follows: (1) types are extended so that they include security information as well as standard information on the kinds of values, such as *int* or $\mathit{int} \to \mathit{int}$; (2) typing rules are constructed by taking security information into account—for example, a conditional expression whose test involves a high-security type is well typed only if both branches are given high-security types to prevent implicit flow; and, finally, (3) a type reconstruction algorithm for the type system is developed. Type-based information flow analysis has been drawing much attention, partially because it cleanly separates the specification and algorithm of the analysis as a type system and type reconstruction, respectively.

Details of those type systems, however, vary substantially (apart from the difference of their base languages) and even their functional core parts are often slightly different. Also, fairly complicated techniques are used to prove noninterference (especially in those for functional languages): some use denotational semantics [10, 1, 3] and some use a non-standard operational semantics [23]. It makes it difficult not only to compare those techniques but also to grasp the essence of the type system and the noninterference property.

Our Goal and Approach. Our goal here is to give a foundational account for type-based information flow analysis. To achieve this goal, we, inspired by the following observation, develop a natural extension of the Curry-Howard isomorphism between a type system for information flow analysis and a certain modal logic.

Modal logic is a language to talk about truth relative to time or places, etc., which are abstracted as possible worlds. Here, we talk about things like at what level the information on certain values can be available. So, it seems to natural to relate the notion of security levels to possible worlds. Since information available at a lower level is also available at a higher level, a suitable modality seems to be *local validity* (or *validity* for simplicity) $\Box_\ell$, which means "it is true at every level higher (or equal to) the security level ℓ that ..." So, the fact that ℓ_1 is higher than ℓ_2 (or information can flow from ℓ_2 to ℓ_1) can be regarded as that a possible world ℓ_1 is reachable from ℓ_2. It is expected that the proposition $\Box_\ell A$ naturally corresponds to the type that represents

the values of type A at the security level ℓ.

Our Contributions. The contributions we make in this paper are summarized as follows:

- As discussed above, we point out an informal correspondence between a type system for information flow analysis and a certain modal logic.
- To make this correspondence more formal, we develop a typed λ-calculus $\lambda_s^{\Box}$ with *modal types* of the form $\Box_\ell A$ and prove that it enjoys subject reduction, Church-Rosser, and strong normalization.
- We also prove noninterference, which is essential to information flow analysis. Our proof is very simple, without using complex denotational techniques or non-standard reduction relations as in previous work.
- To demonstrate that $\lambda_s^{\Box}$ can be a foundation for information flow analysis, we develop encoding from (a purely-functional subset of) the SLam calculus [10] to $\lambda_s^{\Box}$.

Structure of the paper The rest of the paper is organized as follows. Section 2 introduces $\lambda_s^{\Box}$ with its syntax, type system and reduction. Section 3 proves its properties including noninterference. After showing how the SLam calculus can be encoded to $\lambda_s^{\Box}$ in Section 4, we discuss related work in Section 5 and give concluding remarks in Section 6. Most of the proofs are omitted for brevity.

2 The System $\lambda_s^{\Box}$

In this section, we define the typed λ-calculus $\lambda_s^{\Box}$. We first briefly introduce a proof system of the modal logic discussed in the previous section and then proceed to the formal definition of $\lambda_s^{\Box}$.

2.1 Modal Logic of Local Validity

The proof system is partially inspired by Pfenning and Davies' formalization [21], based on the notion of judgments.

The basic idea is to consider judgments to assert truth and local validity separately. Accordingly, a judgment has two kinds of assumption sets and that of truth is written $A_1^{\ell_1}, \ldots, A_n^{\ell_n}; B_1, \ldots, B_m \vdash^{\ell} C$. It means C is true at ℓ under the assumption that A_i is true *everywhere reachable from* ℓ_i and B_j is true at the current level. (In what follows, we write Δ for $A_1^{\ell_1}, \ldots, A_n^{\ell_n}$ and Γ for $B_1, \ldots, B_m$.) A locally valid assumption can be used only when the current level is reachable from the level of the assumption, while the rule for (ordinary) truth assumptions are as usual, as the following two rules:

$$\frac{\ell_i \sqsubseteq \ell}{A_1^{\ell_1}, \ldots, A_i^{\ell_i}, \ldots, A_n^{\ell_n}; \Gamma \vdash^{\ell} A_i} \qquad \Delta; B_1, \ldots, B_j, \ldots, B_m \vdash^{\ell} B_j$$

Validity is expressed by a judgment of truth with zero truth assumptions. The introduction rule for $\Box_\ell A$ amounts to internalizing a judgment of validity as a proposition and the elimination rule turns "$\Box_\ell A$ is true" into "A is valid at ℓ":

$$\frac{\Delta; \cdot \vdash^{\ell} C}{\Delta; \Gamma \vdash^{\ell'} \Box_\ell C} \qquad \frac{\Delta; \Gamma \vdash^{\ell'} \Box_\ell A \qquad \Delta, A^{\ell}; \Gamma \vdash^{\ell'} C}{\Delta; \Gamma \vdash^{\ell'} C}$$

The rules for other logical connectives are straightforward. For example, the elimination rule of implication is as follows.

$$\frac{\Delta; \Gamma \vdash^{\ell} A \to B \qquad \Delta; \Gamma \vdash^{\ell} A}{\Delta; \Gamma \vdash^{\ell} B}$$

Note that the levels of the judgments must be the same.

Keeping this in mind, we proceed to the formal definition of our calculus.

2.2 Syntax

We first assume the countably infinite set **OVar** of *ordinary variables*, ranged over by x, y, z, and **MVar** of *modal variables*, ranged over by u and v. We also assume the partially ordered set $(\mathcal{L}, \sqsubseteq)$ of levels; elements of $\mathcal{L}$ are ranged over by ℓ.

The types of $\lambda_s^{\Box}$ are simple types with the unit type, product types, sum types, and modal types.

2.2.1 Definition [Types]: The set of types, ranged over by A, B, and C, is defined by the following grammar:

$$A ::= \mathit{unit} \mid A \to A \mid A \times A \mid A + A \mid \Box_\ell A$$

The precedence of type constructor is given by the decreasing order $\Box_\ell > \times > + > \to$ and the function type constructor is right associative. For example, $A \to \Box_\ell B \to A \times C$ stands for $A \to ((\Box_\ell B) \to (A \times C))$ and $\Box_\ell A \times \Box_\ell C + B \to A$ for $(((\Box_\ell A) \times (\Box_\ell C)) + B) \to A$.

2.2.2 Definition [Terms]: The set of terms, ranged over by M and N, is defined by the grammar:

$$\begin{aligned} M \quad ::= \quad & x \mid u \mid () \mid \lambda x : A.M \mid M\ M \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M) \mid \iota_1(M) \\ & \mid \iota_2(M) \mid (\textbf{case}\ M\ \textbf{of}\ \iota_1(x) \Rightarrow M \mid \iota_2(x) \Rightarrow M) \mid \textbf{box}_\ell\, M \\ & \mid \textbf{let box}_\ell\, u = M\ \textbf{in}\ M \end{aligned}$$

We omit parentheses according to the usual convention and assume that application is left associative. Also, bound variables and their scopes are defined in a customary manner: $\lambda x : A.M$ bounds x in M; **case** M **of** $\iota_1(x_1) \Rightarrow N_1 \mid \iota_2(x_2) \Rightarrow N_2$ bounds x_1 and x_2 in N_1 and N_2, respectively; **let box**$_\ell\, u = M$ **in** N bounds u in N.

Terms are mostly those of the simply typed λ-calculus with unit, products, and sums. We have two kinds of term variables as the logic has two kinds of assumptions: truth and validity. As we shall see in typing rules, **box**$_\ell\, M$ corresponds to an application of the introduction rule for $\Box_\ell$ when viewed as a proof term, and can be thought as a *sealing* operation where the sealed value is accessed at security level ℓ. Similarly, **let box**$_\ell\, u = M$ **in** N corresponds to an application of the elimination rule, when viewed as a proof term, and can be thought as unsealing. Operationally, if M reduces to **box**$_\ell\, M_0$, then **let box**$_\ell\, u = M$ **in** N reduces to N in which M_0 is substituted for u.

Free variables are defined as usual except that there are two kinds of variables. We write *FMV*(M) (*FOV*(M), resp.) for modal (ordinary, resp.) variables that occur free in M. For example, *FMV*$(\langle \iota_2(z), \lambda x : A.x\, u\rangle) = \{u\}$ and *FOV*$(\langle \iota_2(z), \lambda x : A.x\, u\rangle) = \{z\}$ and *FMV*(**let box**$_\ell\, u = x\, v$ **in** $u\, y) = \{v\}$.

We write $[M/x]$ ($[M/u]$, resp.) for the capture-avoiding substitution of M for the ordinary variable x (the modal variable u, resp.).

2.3 Type System

As discussed above, the judgment form of the logic is $A_1^{\ell_1}, \ldots, A_n^{\ell_n}; B_1, \ldots, B_m \vdash^\ell C$. Accordingly, the type judgment of $\lambda_s^\Box$ is of the form $\Delta; \Gamma \vdash^\ell M : A$, read as "$M$ is given type A at level ℓ under modal context Δ and ordinary context Γ." A modal context, which corresponds to $A_1^{\ell_1}, \ldots, A_n^{\ell_n}$, is a sequence of the form $u ::^\ell A$ where modal variables in the sequence are pairwise distinct. Similarly, an ordinary context, which corresponds to $B_1, \ldots, B_m$, is a sequence of the form $x : B$ where variables in the sequence are pairwise distinct. We often write '$\cdot$' for the empty modal/ordinary context. When both contexts in a judgment are empty, they are simply omitted and the judgment is written $\vdash^\ell M : A$. We write $u ::^\ell A \in \Delta$ when Δ includes $u ::^\ell A$. Similarly for ordinary contexts.

The whole set of typing rules is given in Figure 1. We explain key rules in detail; other rules are fairly standard (modulo two kinds contexts and the annotation of a level).

The rule T-OVAR for ordinary variable reference is just as usual. On the other hand, modal variables can be used only when the level of the variable is lower than or equal to the current level (the rule T-MVAR). From the viewpoint of information flow, a program at a lower level cannot refer to a variable of a higher security level, preventing confidential information from flowing into irrelevant levels. It may first appear that the rule T-MVAR is too strict because a term containing a (modal) variable of high security as a free variable is regarded as confidential computation even if the

variable is just discarded. However, with a certain programming style, we can remedy it: (a core of) the SLam calculus can be encoded into $\lambda_s^{\Box}$ as we discuss in Section 4. In this sense, $\lambda_s^{\Box}$ is as expressive as the SLam calculus.

The rule T-BOX introduces modal types.Since "$\Box_\ell A$ is true" stands for "A is true everywhere reachable from ℓ," the premise must be a judgment of validity, that is, the ordinary context must be empty. Since the judgment does not depend on any assumptions of a particular level, it holds at any level, hence ℓ'. For weakening, any ordinary context can be placed in the conclusion. With the terminology of information flow, a sealed piece of code may be used at an arbitrary level above or equal to ℓ, so it cannot refer to (ordinary) variables available only at a particular security level.

The last rule T-LETBOX eliminates modal types. It turns "$\Box_\ell A$ is true" into "A is valid at ℓ" and adds $u ::^{\ell} A$ to the modal context to deduce B. It might look odd that ℓ is not necessarily related to ℓ' at which M is unsealed. Actual access restriction is enforced by T-MVAR and, if u is used in N (not under $\mathbf{box}_\ell$), it should be the case that $\ell \sqsubseteq \ell'$.

2.3.1 Example: If $\ell \sqsubseteq \ell'$, then the type judgment $\vdash^{\ell'} \lambda x : \Box_\ell A.\mathbf{let\ box}_\ell\, u = x\ \mathbf{in}\ u : \Box_\ell A \to A$ can be derived. By ignoring levels, this type can be viewed as to a variant of the axiom M.

2.3.2 Example: If $\ell_1 \sqsubseteq \ell_3$ and $\ell_2 \sqsubseteq \ell_3$, then

$$\begin{aligned}\vdash^{\ell} \lambda x : \Box_{\ell_1}(A \to B).&\mathbf{let\ box}_{\ell_2}\, u = x\ \mathbf{in}\\ \lambda y : \Box_{\ell_2} A.&\mathbf{let\ box}_{\ell_2}\, v = y\ \mathbf{in}\ \mathbf{box}_{\ell_3}\,(u\,v)\\ &: \Box_{\ell_1}(A \to B) \to \Box_{\ell_2} A \to \Box_{\ell_3} B\end{aligned}$$

is derivable. This type corresponds to a variant of the axiom K.

2.4 Operational Semantics

Operational semantics is given by the reduction relation of the form $M \longrightarrow M'$, read as "M reduces to M' in one step." The computation rules are given as follows.

$$\begin{aligned}(\lambda x : A.M_1)\, M_2 &\longrightarrow [M_2/x]M_1\\ \pi_i(\langle M_1, M_2\rangle) &\longrightarrow M_i\\ \mathbf{case}\ \iota_i(M)\ \mathbf{of}\ \iota_1(x_1) \Rightarrow N_1 \mid \iota_2(x_2) \Rightarrow N_2 &\longrightarrow [M/x_i]N_i\\ \mathbf{let\ box}_\ell\, u = \mathbf{box}_\ell\, M\ \mathbf{in}\ N &\longrightarrow [M/u]N\end{aligned}$$

They can be applied at any point in a term, so we also need the obvious congruence rules (if $M_1 \longrightarrow M_1'$, then $M_1\, M_2 \longrightarrow M_1'\, M_2$, and the like), which we omit here.

2.4.1 Example: Let $M = (\lambda x : \Box_\ell A.\mathbf{let\ box}_\ell\, u = x\ \mathbf{in}\ \pi_2(u))\,(\mathbf{box}_\ell\,\langle M_1, M_2\rangle)$. Then, M reduces to M_2 as follows:

$$\begin{aligned}M &\longrightarrow [\mathbf{box}_\ell\,\langle M_1, M_2\rangle/x]\mathbf{let\ box}_\ell\, u = x\ \mathbf{in}\ \pi_2(u)\\ &\quad (= \mathbf{let\ box}_\ell\, u = \mathbf{box}_\ell\,\langle M_1, M_2\rangle\ \mathbf{in}\ \pi_2(u))\\ &\longrightarrow [\langle M_1, M_2\rangle/u]\pi_2(u) \quad (= \pi_2(\langle M_1, M_2\rangle)) \longrightarrow M_2\end{aligned}$$

$$\frac{x : A \in \Gamma}{\Delta; \Gamma \vdash^{\ell} x : A} \text{ (T-OVAR)} \qquad \Delta; \Gamma \vdash^{\ell} () : \mathit{unit} \text{ (T-UNIT)}$$

$$\frac{u ::^{\ell'} A \in \Delta \qquad \ell' \sqsubseteq \ell}{\Delta; \Gamma \vdash^{\ell} u : A} \text{ (T-MVAR)} \qquad \frac{\Delta; \Gamma, x : A \vdash^{\ell} M : B}{\Delta; \Gamma \vdash^{\ell} \lambda x : A.M : A \to B} \text{ (T-ABS)}$$

$$\frac{\Delta; \Gamma \vdash^{\ell} M : A \to B \qquad \Delta; \Gamma \vdash^{\ell} N : A}{\Delta; \Gamma \vdash^{\ell} M\, N : B} \text{ (T-APP)}$$

$$\frac{\Delta; \Gamma \vdash^{\ell} M : A \qquad \Delta; \Gamma \vdash^{\ell} N : B}{\Delta; \Gamma \vdash^{\ell} \langle M, N \rangle : A \times B} \text{ (T-PAIR)}$$

$$\frac{\Delta; \Gamma \vdash^{\ell} M : A_1 \times A_2 \qquad i \in \{1, 2\}}{\Delta; \Gamma \vdash^{\ell} \pi_i(M) : A_i} \text{ (T-PROJ)}$$

$$\frac{\Delta; \Gamma \vdash^{\ell} M : A_i \qquad i \in \{1, 2\}}{\Delta; \Gamma \vdash^{\ell} \iota_i(M) : A_1 + A_2} \text{ (T-INJ)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash^{\ell} M : A_1 + A_2 \\ \Delta; \Gamma, x_1 : A_1 \vdash^{\ell} N_1 : B \\ \Delta; \Gamma, x_2 : A_2 \vdash^{\ell} N_2 : B\end{array}}{\Delta; \Gamma \vdash^{\ell} \textbf{case}\ M\ \textbf{of}\ \iota_1(x_1) \Rightarrow N_1 \mid \iota_2(x_2) \Rightarrow N_2 : B} \text{ (T-CASE)}$$

$$\frac{\Delta; \cdot \vdash^{\ell'} M : A}{\Delta; \Gamma \vdash^{\ell} \textbf{box}_{\ell'}\ M : \Box_{\ell'} A} \text{ (T-BOX)}$$

$$\frac{\Delta; \Gamma \vdash^{\ell} M : \Box_{\ell'} A \qquad \Delta, u ::^{\ell'} A; \Gamma \vdash^{\ell} N : B}{\Delta; \Gamma \vdash^{\ell} \textbf{let box}_{\ell'}\ u = M\ \textbf{in}\ N : B} \text{ (T-LETBOX)}$$

Figure 1: Typing Rules of $\lambda_s^{\Box}$

3 Properties of $\lambda_s^{\Box}$

In this section, we show that $\lambda_s^{\Box}$ satisfies basic properties including subject reduction, Church-Rosser and strong normalization. Then, we show that it also satisfies the noninterference property, as expected.

3.1 Basic Properties

All the statements of the properties mentioned above are standard. Note that, in Subject Reduction, not only is the type of a term preserved during reduction but also is the level at which the type judgments are derived. They are proved by standard techniques.

3.1.1 Theorem [Subject Reduction]: If $\Delta; \Gamma \vdash^{\ell} M : A$ and $M \longrightarrow N$ then $\Delta; \Gamma \vdash^{\ell} N : A$.

3.1.2 Theorem [Church-Rosser]: If $M \xrightarrow{*} M_1$ and $M \xrightarrow{*} M_2$, then there exists a term N such that $M_1 \xrightarrow{*} N$ and $M_2 \xrightarrow{*} N$.

3.1.3 Theorem [Strong normalization]: If $\Delta; \Gamma \vdash M : A$, then M is strongly normalizing.

3.2 Noninterference

One of the most important correctness properties is *noninterference*, which intuitively means that a program input at a high security level does not affect the program output at a lower level. To state this property more formally, we require the following technical definition.

3.2.1 Definition [Transparent ground types]: A type A is *transparent ground type at level* ℓ if and only if:

1. $A = \mathit{unit}$,
2. $A = A_1 \times A_2$ and both A_1 and A_2 are transparent ground types at ℓ,
3. $A = A_1 + A_2$ and both A_1 and A_2 are transparent ground types at ℓ, or
4. $A = \Box_{\ell'} A_0$ and $\ell' \sqsubseteq \ell$ and A_0 is transparent ground type at ℓ'.

Intuitively, a transparent ground type at ℓ represents values of which it is effectively possible to inspect equality.

Now, the noninterference property is stated as follows:

3.2.2 Theorem [Noninterference]: If $u ::^{\ell} A; \cdot \vdash^{\ell'} M : B$ and B is a transparent ground type at ℓ' and $\ell \not\sqsubseteq \ell'$, then, there exists a unique normal form M' (modulo α-conversion) such that, for any N, if $\vdash^{\ell} N : A$, then $[N/u]M \xrightarrow{*} M'$.

In this statement, u serves as a high level input, whose information is not allowed to flow into the level ℓ' (hence $\ell \not\sqsubseteq \ell'$). The condition on B expresses the fact that the value of B can be inspected (or observed) at level ℓ'. Thus, it cannot include function types or modal types at a level irrelevant to ℓ'.

This theorem can be proved by a very simple manner: it turns out that the modal variable that stands for a high level input will disappear during reduction—this is shown simply by inspecting the structure of normal forms (Theorem 3.2.4). Then, Theorem 3.2.2 is obtained as an easy corollary. We sketch the proof below.

First, we introduce neutral terms which correspond to applications of the elimination rules.

3.2.3 Definition [Neutral terms]: A term is *neutral* if it is of the form x, u, $M\,N$, $\pi_i(M)$, **case** M **of** $\iota_1(x_1) \Rightarrow N_1 \mid \iota_2(x_2) \Rightarrow N_2$, or **let box**$_\ell\, u = M$ **in** N.

3.2.4 Theorem: If $u ::^{\ell} A; \cdot \vdash^{\ell'} M : B$ and M is a normal form and B is a transparent ground type at ℓ' and $u \in \mathit{FMV}(M)$, then $\ell \sqsubseteq \ell'$.

Proof: By induction on the derivation of $u ::^{\ell} A; \cdot \vdash^{\ell'} M : B$ with case analysis on the last rule used. □

Finally, we can prove Theorem 3.2.2.

Proof of Theorem 3.2.2: Let M' be a normal form such that $M \xrightarrow{*} M'$. By Theorem 3.1.1, $u ::^{\ell'} A; \cdot \vdash^{\ell} M' : B$. Then, by the assumption $\ell \not\sqsubseteq \ell'$ and (a contraposition of) Theorem 3.2.4, $u \notin \mathit{FMV}(M')$, hence $[N/u]M' = M'$. By using the fact that $M \longrightarrow M'$ implies $[N/u]M \longrightarrow [N/u]M'$, we have $[N/u]M \xrightarrow{*} M'$, which is a normal form. By Theorems 3.1.2 and 3.1.3, any reduction sequence from $[N/u]M$ will end with M'. □

4 Encoding the SLam Calculus

In this section, we show how (a pure fragment of) the SLam calculus [10] can be encoded into $\lambda_s^{\Box}$. Recursive functions have been dropped because the target language $\lambda_s^{\Box}$ is not equipped with them. Also, we have dropped `protect` in the SLam calculus for the following reasons: (1) the static semantics of `protect`, which raises the security level of the type of the operand, can be simulated by application of a coercion function; (2) the dynamic semantics of `protect`, which dynamically raises the security level of a value, is not relevant to ensure noninterference—even if it was "nop," noninterference could be proved. (In fact, coercion functions used in our encoding are essentially identity functions.) We first briefly review the definition of the SLam calculus and then show the translation with its correctness theorems.

4.1 Review of the SLam Calculus

Let $\mathcal{L}$ be a join semilattice of security levels, ranged over by ℓ. The elements of $\mathcal{L}$ are ordered by $\sqsubseteq$ and the binary join of ℓ_1 and ℓ_2 is written $\ell_1 \vee \ell_2$. A type, more precisely a secure type, in the SLam calculus is a simple type where every type constructor is annotated with a security level, which signifies at which level the information on a value of the type may be available.

4.1.1 Definition [SLam types]: The set of *SLam types*, ranged over by t, and the set of *SLam secure types*, ranged over by s are defined as follows:

$$\begin{array}{lcl} t & ::= & \mathit{unit} \mid s \times s \mid s + s \mid s \to s \\ s & ::= & (t, \ell) \end{array}$$

When $s = (t, \ell)$, we often write $s \bullet \ell'$ for $(t, \ell \vee \ell')$ and $\sharp s$ for ℓ.

4.1.2 Definition [SLam expressions]: The set of expressions, ranged over by the metavariable e, are formed by the typing rules in Figure 2.

An operational semantics of the SLam calculus is given by the following computation rules together with congruence rules, omitted for brevity.

$$\begin{array}{lcl} (\lambda x : s.e_0)_\ell \, e_1 & \rightsquigarrow & [e_1/x]e_0 \\ \pi_i(\langle e_1, e_2 \rangle_\ell) & \rightsquigarrow & e_i \\ \textbf{case}\ \iota_i(e_0)_\ell\ \textbf{of}\ \iota_1(x_1) \Rightarrow e_1 \mid \iota_2(x_2) \Rightarrow e_2 & \rightsquigarrow & [e_0/x_i]e_i \end{array}$$

4.2 Translation from SLam to $\lambda_s^\Box$

The translation from the SLam calculus to $\lambda_s^\Box$ is rather straightforward. We take $\mathcal{L}$ as the partially ordered set of levels. Secure types are translated to ($\lambda_s^\Box$) types by the function $|s|$:

$$|\mathit{unit}| \;=\; \mathit{unit}, \quad |s_1 \mathbf{op} s_2| \;=\; |s_1| \mathbf{op} |s_2|, \quad |(t, \ell)| \;=\; \Box_\ell |t|$$

where **op** is $\times$, $+$, or $\to$. For the sake of simplicity, we assume that all bound variable names are different and there is a bijection from the set of SLam variables to the set of modal variables. The modal variable corresponding to x is written u_x below. Then, type-directed translation rules for expressions are given in Figures 3 and 4.

As will be formally stated in Theorem 4.2.1, a SLam expression of type (t, ℓ) will be translated to a term typed at ℓ. The translation follows the following patterns: (1) when a subexpression is consumed by a given SLam operation (for example, $e_1\, e_2$ consumes e_1 but not e_2), it is translated to the corresponding operation and the result is immediately unsealed; and (2) when a subexpression is not really consumed by an

$$\frac{\Gamma(x) = s}{\Gamma \vdash x : s} \quad \frac{\Gamma \vdash e : s \qquad s \leq s'}{\Gamma \vdash e : s'} \quad \Gamma \vdash ()_\ell : (\mathit{unit}, \ell) \quad \frac{\Gamma, x : s_1 \vdash e_0 : s_2}{\Gamma \vdash (\lambda x : s_1.e_0)_\ell : (s_1 \to s_2, \ell)}$$

$$\frac{\Gamma \vdash e_1 : (s_2 \to s_0, \ell) \quad \Gamma \vdash e_2 : s_2}{\Gamma \vdash e_1\, e_2 : s_0 \bullet \ell} \qquad \frac{\Gamma \vdash e_1 : s_1 \qquad \Gamma \vdash e_2 : s_2}{\Gamma \vdash \langle e_1, e_2 \rangle_\ell : (s_1 \times s_2, \ell)} \qquad \frac{\Gamma \vdash e : (s_1 \times s_2, \ell)}{\Gamma \vdash \pi_i(e) : s_i \bullet \ell}$$

$$\frac{\Gamma \vdash e : s_i}{\Gamma \vdash \iota_i(e)_\ell : (s_1 + s_2, \ell)}$$

$$\frac{\Gamma \vdash e_0 : (s_1 + s_2, \ell) \quad \Gamma, x_1 : s_1 \vdash e_1 : s \qquad \Gamma, x_2 : s_2 \vdash e_2 : s}{\Gamma \vdash \textbf{case}\ e_0\ \textbf{of}\ \iota_1(x_1) \Rightarrow e_1 \mid \iota_2(x_2) \Rightarrow e_2 : s \bullet \ell} \qquad \frac{\ell \sqsubseteq \ell'}{(\mathit{unit}, \ell) \leq (\mathit{unit}, \ell')}$$

$$\frac{\ell \sqsubseteq \ell' \qquad s_1' \leq s_1 \qquad s_2 \leq s_2'}{(s_1 \to s_2, \ell) \leq (s_1' \to s_2', \ell')} \qquad \frac{\ell \sqsubseteq \ell' \qquad s_1 \leq s_1' \qquad s_2 \leq s_2'}{(s_1 \times s_2, \ell) \leq (s_1' \times s_2', \ell')}$$

$$\frac{\ell \sqsubseteq \ell' \qquad s_1 \leq s_1' \qquad s_2 \leq s_2'}{(s_1 + s_2, \ell) \leq (s_1' + s_2', \ell')}$$

Figure 2: Typing rules of a core of the SLam calculus

Translation from subtyping to coercion: $s_1 \leq s_2 \searrow M$

$$\frac{\ell \sqsubseteq \ell'}{(\mathit{unit}, \ell) \leq (\mathit{unit}, \ell') \searrow \quad \lambda x : \Box_\ell \mathit{unit}.\textbf{let box}_\ell\, u_x = x\ \textbf{in}\ u_x} \qquad \frac{\ell \sqsubseteq \ell' \qquad s_1' \leq s_1 \searrow M_1 \qquad s_2 \leq s_2' \searrow M_2}{(s_1 \to s_2, \ell) \leq (s_1' \to s_2', \ell') \searrow \quad \lambda x : |(s_1 \to s_2, \ell)|.\textbf{let box}_\ell\, u_x = x\ \textbf{in}\ \lambda y : |s_1'|.M_2\,(\textbf{box}_{\sharp s_2}\,(u_x\,(\textbf{box}_{\sharp s_1}\,(M_1\, y))))}$$

$$\frac{\ell \sqsubseteq \ell' \qquad s_1 \leq s_1' \searrow M_1 \qquad s_2 \leq s_2' \searrow M_2}{\begin{array}{c}(s_1 \times s_2, \ell) \leq (s_1' \times s_2', \ell') \searrow \\ \lambda x : |(s_1 \times s_2, \ell)|.\textbf{let box}_\ell\, u_x = x\ \textbf{in} \\ \langle \textbf{box}_{\sharp s_1'}\,(M_1\,(\pi_1(u_x))), \textbf{box}_{\sharp s_2'}\,(M_2\,(\pi_2(u_x)))\rangle\end{array}}$$

$$\frac{\ell \sqsubseteq \ell' \qquad s_1 \leq s_1' \searrow M_1 \qquad s_2 \leq s_2' \searrow M_2}{\begin{array}{c}(s_1 + s_2, \ell) \leq (s_1' + s_2', \ell') \searrow \\ \lambda x : |(s_1 + s_2, \ell)|.\textbf{let box}_\ell\, u_x = x\ \textbf{in} \\ \textbf{case}\ u_x\ \textbf{of}\ \iota_1(y) \Rightarrow \iota_1(\textbf{box}_{\sharp s_1}\,(M_1\, y)) \mid \iota_2(z) \Rightarrow \iota_2(\textbf{box}_{\sharp s_2}\,(M_2\, z))\end{array}}$$

Figure 3: Translation from SLam to $\lambda_s^\Box$

Translation of expressions: $\Gamma \vdash e : s \searrow M$

$$\frac{\Gamma(x) = s}{\Gamma \vdash x : s \searrow u_x} \qquad \frac{\Gamma \vdash e : s \searrow N \qquad s \leq s' \searrow M}{\Gamma \vdash e : s' \searrow M\,(\mathbf{box}_{\sharp s}\, N)} \qquad \Gamma \vdash ()_\ell : (\mathit{unit}, \ell) \searrow ()$$

$$\frac{\Gamma, x : s \vdash e_0 : s_0 \searrow M}{\begin{array}{c}\Gamma \vdash (\lambda x : s.e_0)_\ell : (s \to s_0, \ell) \searrow \\ \lambda x : |s|.\mathbf{let\ box}_\ell\, u_x = x\ \mathbf{in}\ \mathbf{box}_{\sharp s_0}\, M\end{array}}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : (s_2 \to s_0, \ell) \searrow M_1 \qquad \Gamma \vdash e_2 : s_2 \searrow M_2 \\ u \notin \mathit{FMV}(M_1) \cup \mathit{FMV}(M_2)\end{array}}{\Gamma \vdash e_1\, e_2 : s_0 \bullet \ell \searrow \mathbf{let\ box}_{\sharp s_0}\, u = M_1\,(\mathbf{box}_{\sharp s_2}\, M_2)\ \mathbf{in}\ u}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : s_1 \searrow M_1 \\ \Gamma \vdash e_2 : s_2 \searrow M_2\end{array}}{\begin{array}{c}\Gamma \vdash \langle e_1, e_2\rangle_\ell : (s_1 \times s_2, \ell) \searrow \\ \langle \mathbf{box}_{\sharp s_1}\, M_1, \mathbf{box}_{\sharp s_2}\, M_2\rangle\end{array}} \qquad \frac{\begin{array}{c}\Gamma \vdash e : (s_1 \times s_2, \ell) \searrow M \\ i \in \{1,2\} \qquad u \notin \mathit{FMV}(M)\end{array}}{\begin{array}{c}\Gamma \vdash \pi_i(e) : s_i \bullet \ell \searrow \\ \mathbf{let\ box}_{\sharp s_i}\, u = \pi_i(M)\ \mathbf{in}\ u\end{array}}$$

$$\frac{\Gamma \vdash e : s_i \searrow M \qquad i \in \{1,2\}}{\Gamma \vdash \iota_i(e)_\ell : (s_1 + s_2, \ell) \searrow \iota_i(\mathbf{box}_{\sharp s_i}\, M)}$$

$$\frac{\Gamma \vdash e_0 : (s_1 + s_2, \ell) \searrow M_0 \qquad \Gamma, x : s_1 \vdash e_1 : s \searrow M_1 \qquad \Gamma, y : s_2 \vdash e_2 : s \searrow M_2}{\begin{array}{c}\Gamma \vdash \mathbf{case}\ e_0\ \mathbf{of}\ \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2 : s \bullet \ell \searrow \\ \mathbf{case}\ M_0\ \mathbf{of}\ \iota_1(x) \Rightarrow \mathbf{let\ box}_{\sharp s_1}\, u_x = x\ \mathbf{in}\ M_1 \mid \iota_2(y) \Rightarrow \mathbf{let\ box}_{\sharp s_2}\, u_y = y\ \mathbf{in}\ M_2\end{array}}$$

Figure 4: Translation from SLam to $\lambda_s^\Box$

operation (for example, none of function application, pairing, and injection consume their arguments), the subexpression is first sealed and passed to the operation. This straightforward translation can introduce a lot of unsealing followed by sealing; developing an optimized translation may be of interesting.

As mentioned in Section 2, a term containing a (modal) variable of high security as a free variable is regarded as confidential computation even if the variable is just discarded. The translation patterns also show how to avoid such undesirable increase of security levels. Unless a modal variable is really consumed, it can be passed to elsewhere by putting in a **box**, making the security level of the whole expression unrelated to that of the variable.

Correctness of the translation is given by Theorem 4.2.3. It requires auxiliary theorems stating that translation preserves typing and semantics with the following definitions.

Translation of SLam contexts to modal contexts is given as follows:

$$|x_1 : (t_1, \ell_1), \ldots, x_n : (t_n, \ell_n)| = u_{x_1} ::^{\ell_1} |t_1|, \ldots, u_{x_n} ::^{\ell_n} |t_n|$$

We write $e \Downarrow e'$ when $e \overset{*}{\rightsquigarrow} e'$ and there is no e'' such that $e' \longrightarrow e''$.

4.2.1 Theorem [Translation Preserves Typing]: If $\Gamma \vdash e : (t, \ell)$, then there exists M such that $\Gamma \vdash e : (t, \ell) \searrow M$ and $|\Gamma|; \cdot \vdash^{\ell} M : |t|$.

4.2.2 Theorem [Adequacy]: If $\vdash e : (t, \ell) \searrow M$ and t is ground, then $e \Downarrow e'$ iff $M \xrightarrow{*} M'$ and $\vdash e' : s \searrow M'$ for some $s \leq (t, \ell)$ and normal form M'.

4.2.3 Theorem [SLam Noninterference]: Let ℓ_1 and ℓ_2 be any two elements of $\mathcal{L}$. If $\ell_1 \not\sqsubseteq \ell_2$ and $x : (t, \ell_1) \vdash e : ((\mathit{unit}, \ell_2) + (\mathit{unit}, \ell_2), \ell_2)$, then for any e_1 and e_2 such that $\vdash e_i : (t, \ell_1)$, $[e_1/x]e \Downarrow v$ iff $[e_2/x]e \Downarrow v$.

Proof sketch: By Theorem 4.2.1, $x : (t, \ell_1) \vdash e : ((\mathit{unit}, \ell_2) + (\mathit{unit}, \ell_2), \ell_2) \searrow M$ and $u_x ::^{\ell_1}; \cdot \vdash M : \Box_{\ell_2}(\Box_{\ell_2}\mathit{unit} + \Box_{\ell_2}\mathit{unit})$. Then, the conclusion is immediate from Theorem 3.2.2. □

5 Related Work

Type-based Information Flow Analysis. As mentioned before, there have been a lot of type-based techniques of information flow analysis for various kinds of languages [10, 1, 23, 19, 3, 25]. (See also Sabelfeld and Myers [24] for an excellent survey of this area.) Among them, close to ours are of course ones for functional languages [10, 1, 23]. It is interesting to see that even their core type systems and semantics are slightly different from each other and that the proofs of noninterference are also significantly different, accordingly. For example, Heintze and Riecke [10]

and Abadi et al. [1] proved noninterference for variants of the SLam calculus, by using denotational techniques, while Pottier and Simonet [23] did it, by developing a customized operational semantics that can express two executions with different high security inputs at once. On the other hand, our noninterference proof is very simple, and essentially based on the observation that high security inputs simply disappear during reduction. Of course, this argument was possible as we have (1) full β-reduction, where any subterms—even terms under lambda abstractions—can be reduced and (2) nondeterministic reduction that allows to delay computation at a higher level.[1] So, we think it doesn't directly extend to a language with side-effects. Nevertheless, we believe it is worth noting that noninterference for a purely functional language *is* easy to prove.

To perform precise analysis for the while language with (first-order) procedures, Volpano and Smith [26] introduced procedures polymorphic with respect to security levels, that is, procedures parameterized by variables ranging over security levels. Although our calculus is not equipped with such a notion, it would be in principle possible to introduce the universal quantifier for possible worlds to our language. For example, the type of a function that takes two integers of the same security level and yields their sum might be written something like $\forall n \in \mathcal{L}.(\Box_n int \times \Box_n int \to \Box_n int)$.

Barthe and Serpette [4] developed a type system for information flow analysis (and binding time analysis) for $FOb_{1\leq:}$ [2], an object calculus with a first-order type system and subtyping, and proved noninterference. Their approach to proving noninterference is very similar to ours: both proofs are entirely syntactic and use the fact that a normal form at some level cannot contain higher-level variables. Our proof may appear slightly more involved since we use Church-Rosser and Strong Normalization properties, which are required only because we adopt full β-reduction and do not fix the evaluation strategy. On the other hand, Barthe and Serpette assumed the normal order for reduction; so, it always leads to a normal form (if any), making the proof look slightly simpler.

Monadic Type Systems and Lax Logic. One of the closest related work is Abadi et al.'s dependency core calculus (DCC) [1]. Its purpose is to give a unified account for more general program analysis—dependency analysis, of which information flow analysis is one instance. DCC, an extension of Moggi's computational lambda calculus [17], is equipped with monadic types, $T_\ell A$, indexed by a predetermined lattice element ℓ.

We believe that similarity to our modal types $\Box_\ell A$ is not superficial. In fact, the computational lambda calculus has been found to correspond to a modal logic called lax logic [5, 9]. One standard interpretation of lax modality, usually written $\bigcirc A$, is "A is true *under some constraint*," and the elimination rule of lax modality is given as:

[1] Strong normalization and Church-Rosser guarantee that noninterference holds under any reduction strategy, though.

$$\frac{\Gamma \mid\!\!\vdash \bigcirc A \qquad \Gamma, A \mid\!\!\vdash \bigcirc B}{\Gamma \mid\!\!\vdash \bigcirc B}$$

It corresponds to the typing rule for monadic binding **bind** $x = M$ **in** N by interpreting $\bigcirc$ as the monadic type constructor. Later, Pfenning and Davies [21] pointed out that the lax modality $\bigcirc$ can be decomposed as "possibly necessary" $\Diamond\Box$. On the other hand, our modal types $\Box_\ell A$ could be decomposed as $@_\ell \Box A$, where $@_\ell A$ would mean "A is true at the world ℓ." In some sense, in $\lambda_s^\Box$, it is made explicit at which world $\Box A$ holds, while, in lax logic and the original computational lambda calculus, it is abstracted out by the possibility modality $\Diamond$. However, the typing rules of $\lambda_s^\Box$ and DCC are rather different. It is left for future work to figure out how they (do or do not) correspond to each other.

Type Systems Based On Modal Logic. Recently, several typed calculi based on proof systems of modal logic have been proposed for various purposes: staged computation [7], binding-time analysis in partial evaluation [6], a formal account for the notion of meta-variables [20], and distributed computation [18, 13]. Each calculus (including $\lambda_s^\Box$) has slightly different modality, specialized to its purpose. To our knowledge, our work is the first to point out the relevance of modal logic to security or dependency analysis.

6 Conclusion

We have developed a typed lambda-calculus $\lambda_s^\Box$ to give a foundational account for type-based information flow analysis. The calculus corresponds, by (a natural extension of) the Curry-Howard isomorphism, to a proof system of an intuitionistic modal logic of local validity. The correspondence is based on the observation that security levels can be interpreted as possible worlds, legal directions of information flow can be as the reachability relation on possible worlds, and security types can be as propositions of validity. The calculus is shown to satisfy desirable properties including subject reduction, Church-Rosser, strong normalization. Moreover, we have found the noninterference property, a correctness property essential for information flow analysis, can be proved in a very simple syntactic manner, without involving denotational semantics or non-standard operational semantics. We also show that a purely functional core of the SLam calculus can be encoded into $\lambda_s^\Box$ and that its noninterference can be proved in terms of $\lambda_s^\Box$.

We briefly discuss possible future work below.

As mentioned in the last section, we conjecture that DCC's monadic types have strong connection with our modal types, although the type system is rather different. It is interesting work to investigate Curry-Howard isomorphism for DCC and study its logical interpretation.

We have studied a modal logic with validity as the only modality. It remains as an open question whether there is any sensible interpretation of modal possibility [21] in our context.

Recently, type-based information flow analysis for low-level languages such as the Java Virtual Machine Language (JVML) has been studied [16]. Since a type system for JVML can be interpreted as a variant of Gentzen's sequent calculus [14], we think it would be possible to apply our idea and develop a correspondence for low-level languages.

It may be an interesting question to answer how general our overall approach to foundations for type-based program analyses is. There have been a lot of type-based program analyses that use non-standard type systems—non-standard in the sense that types are decorated with information peculiar to the purpose of the analysis. It may be possible to find yet another correspondence between type-based program analyses and (modal) logic. Such work will be useful to deepen understanding of the essence of program analyses, as our work have been so for information flow analysis.

Acknowledgements. The authors are grateful to Masahiko Sato and anonymous reviewers for their useful comments. This work is supported in part by Grant-in-Aid for Scientific Research on Priority Areas Research No. 12133202 and Grant-in-Aid for Young Scientists (B) No. 15700011 from MEXT of Japan.

References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Proc. of POPL '99*, pages 147–160, 1999.

[2] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[3] Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in Java-like language. In *Proc. of 15th CSFW*, pages 253–267, 2002.

[4] Gilles Barthe and Bernard P. Serpette. Partial evaluation and non-interference for object calculi. In *Proc. of 4th Fuji International Symposium on Functional and Logic Programming (FLOPS'99)*, volume 1722 of *LNCS*, pages 53–67, Tsukuba, Japan, 1999.

[5] P. N. Benton, Gavin Bierman, and Valeria de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, 1998.

[6] Rowan Davies. A temporal-logic approach to binding-time analysis. In *Proc. of LICS'96*, pages 184–195, 1996.

[7] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[8] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, July 1977.

[9] Matt Fairtlough and Michael Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, 1997.

[10] Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In *Proc. of POPL '98*, pages 365–377, 1998.

[11] Kohei Honda, Vasco Vasconcelos, and Nobuko Yoshida. Secure information flows as typed process behaviour. In *Proc. of ESOP (ESOP)*, Springer LNCS 1782, pages 180–199, 2000.

[12] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *Proc. of 29th POPL (POPL'02)*, pages 81–92, Portland, OR, January 2002.

[13] Limin Jia and David Walker. Modal proofs as distributed programs. In *Proc. of ESOP*, volume 2986 of *LNCS*, pages 219–233, 2004.

[14] Shin-ya Katsumata and Atsushi Ohori. Proof-directed de-compilation of low-level code. In *Proc. of ESOP*, volume 2028 of *LNCS*, pages 352–366, 2001.

[15] Naoki Kobayashi. Type-based information flow analysis for the pi-calculus. Technical Report TR03-0007, Dept. of Computer Science, Tokyo Institute of Technology, October 2003.

[16] Naoki Kobayashi and Keita Shirane. Type-based information flow analysis for JVML. In *Informal Proc. of APLAS'02*, 2002.

[17] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[18] Jonathan Moody. Modal logic as a basis for distributed computation. Technical Report CMU-CS-03-194, School of Computer Science, Carnegie Mellon University, 2003.

[19] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. of POPL'99*, pages 228–241, 1999.

[20] Aleksander Nanevski, Brigitte Pientka, and Frank Pfenning. A modal foundation for meta-variables. In *Proc. of Merλin*, 2003.

[21] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[22] François Pottier. A simple view of type-secure information flow in the pi-calculus. In *Proc. of CSFW*, pages 320–330, 2002.

[23] François Pottier and Vincent Simonet. Information flow inference in ML. *ACM TOPLAS*, 25(1):117–158, 2003.

[24] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal On Selected Areas In Communications*, 21(1):5–19, 2003.

[25] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-thereaded imperative language. In *Proc. of POPL*, pages 355–364, 1998.

[26] Dennis Volpano and Geoffrey Smith. A type-based approach to program security. In *Proc. of TAPSOFT*, volume 1214 of *LNCS*, pages 607–621, 1997.

[27] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

Typing Noninterference for Reactive Programs *

Ana Almeida Matos, Gérard Boudol and Ilaria Castellani

Abstract

We propose a type system to enforce the security property of *noninterference* in a core reactive language, obtained by extending the imperative language of Volpano, Smith and Irvine with reactive primitives manipulating broadcast signals and with a form of "scheduled" parallelism. Due to the particular nature of reactive computations, the definition of noninterference has to be adapted. We give a formulation of noninterference based on bisimulation. Our type system is inspired by that introduced by Boudol and Castellani, and independently by Smith, to cope with nontermination and time leaks in a language for parallel programs with scheduling. We establish the soundness of this type system with respect to our notion of noninterference.

1 Introduction

To be widely accepted and deployed, the mobile code technology has to provide formal guarantees regarding the various security issues that it raises. For instance, foreign code should not be allowed to corrupt, or even simply to get knowledge of secret data owned by its execution context. Similarly, a supposedly trusted code should be checked for not disclosing private data to public knowledge. In [3] we have introduced a core programming model for mobile code called ULM, advocating the use of a *locally synchronous* programming style [1, 10] in a *globally asynchronous* computing context. It is therefore natural to examine the security issues from the point of view of this programming model. In this paper, we address some of these issues, and more specifically the non-disclosure property, for a simplified version of the ULM language. We recall the main features of the synchronous programming style, in its control-oriented incarnation:

Broadcast signals Program components react according to the presence or absence of signals, by computing and emitting signals that are broadcast to all components of a given "synchronous area".

*Research partially funded by the EU IST FET Project MIKADO, by the french ACI Project CRISS, and by the PhD scholarship POSI/SFRH/BD/7100/2001.

Suspension Program components may be in a suspended state, because they are waiting for a signal which is absent at the moment where they get the control.

Preemption There are means to abort the execution of a program component, depending on the presence or absence of a signal.

Instants Instants are successive periods of the execution of a program, where the signals are consistently seen as present or absent by all components.

The so-called *reactive* variant of the synchronous programming style, designed by Boussinot, has been implemented in a number of languages and used for various applications, see [8, 7]. This differs from the synchronous language ESTEREL [2], for instance in the way absence of signals is dealt with: in reactive programming, the absence of a signal can only be determined at the end of the current instant, and reaction is postponed to the next instant. In this way, the causal paradoxes that arise in some ESTEREL programs can be avoided, making reactive programming well suited for systems where concurrent components may be dynamically added or removed, as it is the case with mobile code.

We consider here a core reactive language, which is a subset of ULM that extends the sequential language of [18] with reactive primitives and with an operator of alternating parallel composition (incorporating a fixed form of scheduling). As expected, these new constructs add expressive power to the language and induce new forms of security leaks. Moreover, the two-level nature of reactive computations, which evolve both within instants and across instants, introduces new subtleties in the definition of noninterference. We give a formulation of noninterference based on bisimulation, as is now standard [15, 14, 16, 4]. We define a type system to enforce this property of noninterference, along the lines of that proposed by Boudol and Castellani [5], and independently by Smith [16], for a language for parallel programs with scheduling. In this approach, types impose constraints on the relation between the security levels of tested and written variables and of received and emitted signals.

Let us briefly recall the intuition about noninterference: in a system with multiple security levels, information should only be allowed to flow from lower to higher (more secure) levels [9]. As usual, we assume security levels to form a lattice. However, in most of our examples, we shall use only two security levels, *low* (public, L) and *high* (secret, H). Security levels are attributed to variables and signals, using subscripts to specify them (eg. x_H is a variable of high level). In a sequential imperative language, an *insecure flow* of information, or *interference*, occurs when the initial values of high variables influence the final value of low variables. The simplest case of insecure flow is that of an assignment of the value of a high variable to a low variable, as in $y_L := x_H$. It is called *explicit (insecure) flow*. More subtle kinds of flow, called *implicit flows*, may be induced by the flow of control. An example is the program $\texttt{if } x_H = 0 \texttt{ then } y_L := 0 \texttt{ else nil}$, where at the end of execution the value of y_L may give information about x_H.

Other programs may be considered as secure or not depending on the context in which they might appear. For instance, the program

$$(\texttt{while}\ x_H \neq 0\ \texttt{do}\ \texttt{nil})\ ;\ y_L := 0 \tag{1}$$

may be considered safe in a sequential setting (since whenever it terminates it produces the same value 0 for y_L), whereas it becomes critical in the presence of parallelism or scheduling (as explained for instance in [4, 5]). When moving to a reactive setting we must reconsider the security of such programs in the new contexts.

In the ULM model we consider two kinds of parallel composition: first, there is the globally asynchronous composition of "reactive machines". This is similar to the parallel composition usually considered in the literature (see for instance [15, 5, 16]), except that it is quite natural to assume that there is no specific scheduling at this level. We do not consider this global composition here, and we expect it could be dealt with in a standard compositional manner. Then, in a locally synchronous area, that is within a reactive machine, parallel composition is quite different: like in the implementation of reactive programming [7], we assume a deterministic cooperative scheduling discipline on threads. It is well-known that scheduling induces new possibilities of flow (see [15, 14] for instance), and this is indeed the case with the sequentialisation of threads that we adopt. Consequently, programs such as (1) can be dangerous in a reactive setting. Similarly, we should question whether the reactive counterparts of the above programs pose problems. In fact they do, as we shall see in Section 3.1.

Another problem we are faced with when addressing the security of reactive programs is their ability to suspend while waiting for an absent signal, thus giving rise to a special event called *instant change*. One of the effects of an instant change is to reset all signals to "absent". With the constructs of the language we may write (for any security level) a program `pause`, whose behaviour is to suspend for the current instant, and terminate at the beginning of the next instant (see Section 2.2). This allows us to write the following program:

$$\texttt{emit}\ a_L\ ;\ \texttt{if}\ x_H = 0\ \texttt{then}\ \texttt{nil}\ \texttt{else}\ \texttt{pause} \tag{2}$$

Depending on the value of x_H, this program may either terminate within an instant, in which case a_L remains present, or suspend and change instant, in which case a_L is erased. However, since instant changes are not statically predictable in general, we do not consider as observable the change in the status of low signals that occurs in the transition from one instant to the next. Consequently, we consider (2) as safe. These considerations will lead us to adapt the definition of noninterference. We will then be able to prove that our type system is sound for this notion of noninterference.

The rest of the paper is organized as follows. In Section 2 we introduce the language and its operational semantics. Section 3 presents the type system and some properties of typed programs, including subject reduction. We then define noninterference as a bisimulation relation and prove the soundness of our type system with

respect to it. Most proofs are omitted in this extended abstract. They may be found in the full paper [6].

2 The language

2.1 Syntax

We consider two infinite and disjoint sets of *variables* and *signals*, Var and Sig, ranged over by x, y, z and a, b, c respectively. We then let $Names$ be the union $Var \cup Sig$, ranged over by n, m. The set Exp of expressions includes booleans and naturals with the usual operations, but no signals. For convenience we have chosen to present the type system only in Section 3.1. However types, or more precisely *security levels*, ranged over by δ, θ, σ, already appear in the syntax of the language. Security levels constitute what we call *simple types*, and are used to type expressions and declared signals. In Section 3 we will see how more complex types for variables, signals and programs may be built from simple types.

The language of *processes* $P, Q \in Proc$ is defined by:

$$
\begin{array}{rcccccc}
P & ::= & x := e & | & \texttt{let}\ x : \delta = e\ \texttt{in}\ P & | & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \quad | \\
 & & \texttt{while}\ e\ \texttt{do}\ P & | & P ; Q & | & \texttt{nil} \quad | \\
 & & \texttt{emit}\ a & | & \texttt{local}\ a : \delta\ \texttt{in}\ P & | & \texttt{do}\ P\ \texttt{watching}\ a \quad | \\
 & & \texttt{when}\ a\ \texttt{do}\ P & | & (P \curvearrowleft Q) & | &
\end{array}
$$

Note the use of brackets to make the precedence of $\curvearrowleft$ unambiguous. The construct $\texttt{let}\ x : \delta = e\ \texttt{in}\ Q$ binds free occurrences of variable x in Q, whereas the construct $\texttt{local}\ a : \delta\ \texttt{in}\ Q$ binds free occurrences of signal a in Q. The free variables and signals of a program P, noted $\texttt{fv}(P)$ and $\texttt{fs}(P)$ respectively, are defined in the usual way.

2.2 Operational Semantics

Configurations C are quadruples $\langle \Gamma, S, E, P \rangle$ composed of a type-environment Γ, a variable-store S, a signal-environment E and a program P. The type-environment is a mapping from names to the appropriate types. We denote its update by $\{x : \delta\ \textit{var}\}\Gamma$ or $\{a : \delta\ \textit{sig}\}\Gamma$, where δ *var* and δ *sig* denote types for variables and signals respectively (formally introduced in Section 3.1). A variable-store is a mapping from variables to values. By abuse of language we denote by $S(e)$ the atomic evaluation of the expression e under S, which we assume to always terminate and to produce no side effects. We denote by $\{x \mapsto S(e)\}S$ the update or extension of S with the value of e for the variable x, depending on whether the variable is present or not in the domain of S. The signal-environment is the set of signals which are considered to be present. We restrict our attention to well-formed configurations, satisfying $\texttt{fv}(P) \subseteq \texttt{dom}(S)$

$$\text{(WHEN-SUS}_1\text{)}\ \frac{a \notin E}{(E, \texttt{when}\ a\ \texttt{do}\ P)\ddagger} \qquad \text{(WHEN-SUS}_2\text{)}\ \frac{(E,P)\ddagger}{(E, \texttt{when}\ a\ \texttt{do}\ P)\ddagger}$$

$$\text{(WATCH-SUS)}\ \frac{(E,P)\ddagger}{(E, \texttt{do}\ P\ \texttt{watching}\ a)\ddagger}$$

$$\text{(SEQ-SUS)}\ \frac{(E,P)\ddagger}{(E, P\,;\,Q)\ddagger} \qquad \text{(PAR-SUS)}\ \frac{(E,P)\ddagger \quad (E,Q)\ddagger}{(E, P \upharpoonleft Q)\ddagger}$$

Figure 1: Suspension predicate

and $\texttt{dom}(S) \cup E \subseteq \texttt{dom}(\Gamma)$. We will generally use the word *memory* to refer to the pair $\langle$variable-store, signal-environment$\rangle$.

A distinguishing feature of reactive programs is their ability to *suspend* while waiting for a signal. The suspension predicate, which applies to pairs of programs and signal-environments, is defined inductively by the rules in Figure 1. Suspension is introduced by the construct $\texttt{when}\ a\ \texttt{do}\ P$, in case signal a is absent. The suspension of a program P is propagated to certain contexts, namely processes of the form $P\,;\,Q$, $\texttt{do}\ P\ \texttt{watching}\ a$, $\texttt{when}\ a\ \texttt{do}\ P$ and $P \upharpoonleft Q$, which we call *suspendable processes*. We extend suspension to configurations by letting $\langle \Gamma, S, E, P\rangle\ddagger$ if $\langle E, P\rangle\ddagger$.

There are two forms of transitions between configurations: simple *moves*, denoted by the arrow $C \to C'$, and *instant changes*, denoted by $C \hookrightarrow C'$. These are collectively referred to as *steps*, and is denoted by $C \longmapsto C'$. The reflexive and transitive closure of these transition relations are denoted with a '$*$' as usual. An *instant* is a sequence of moves leading to termination or suspension.

2.2.1 Moves

The operational rules for imperative and reactive constructs are given in Figure 2. The functions $\texttt{newv}(N)$ and $\texttt{news}(N)$ are injective functions on finite sets of names which return a fresh name not in N, respectively a variable and a signal. They are used in order to guarantee determinism in the language. The notation $\{n/m\}P$ stands for substitution (name-capture avoiding) of m by n in P.

The imperative rules are as usual, where termination is dealt with by reduction to '`nil`'. Some comments on the reactive rules are in order. Signal emission adds a signal to the signal-environment. The local signal declaration is standard. The `watching` construct allows the execution of its body until an instant change occurs; the execution will then resume or not at the next instant depending on the presence of the signal (as explained below). As for the `when` construct, execution of its body depends on the presence of the signal; the body suspends if the signal is absent. Alternating parallel composition implements a co-routine mechanism. It executes its left

(ASSIGN-OP)
$$\langle \Gamma, S, E, x := e \rangle \to \langle \Gamma, \{x \mapsto S(e)\}S, E, \texttt{nil} \rangle$$

(SEQ-OP$_1$)
$$\langle \Gamma, S, E, \texttt{nil} \,;\, Q \rangle \to \langle \Gamma, S, E, Q \rangle$$

(SEQ-OP$_2$)
$$\frac{\langle \Gamma, S, E, P \rangle \to \langle \Gamma', S', E', P' \rangle}{\langle \Gamma, S, E, P \,;\, Q \rangle \to \langle \Gamma', S', E', P' \,;\, Q \rangle}$$

(LET-OP)
$$\frac{y = \texttt{newv}(\texttt{dom}(\Gamma))}{\langle \Gamma, S, E, \texttt{let}\ x : \delta = e\ \texttt{in}\ P \rangle \to \langle \{y : \delta\ \textit{var}\}\Gamma, \{y \mapsto S(e)\}S, E, \{y/x\}P \rangle}$$

(COND-OP$_1$)
$$\frac{S(e) = \texttt{true}}{\langle \Gamma, S, E, \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \rangle \to \langle \Gamma, S, E, P \rangle}$$

(COND-OP$_2$)
$$\frac{S(e) = \texttt{false}}{\langle \Gamma, S, E, \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \rangle \to \langle \Gamma, S, E, Q \rangle}$$

(WHILE-OP$_1$)
$$\frac{S(e) = \texttt{true}}{\langle \Gamma, S, E, \texttt{while}\ e\ \texttt{do}\ P \rangle \to \langle \Gamma, S, E, P \,;\, \texttt{while}\ e\ \texttt{do}\ P \rangle}$$

(WHILE-OP$_2$)
$$\frac{S(e) = \texttt{false}}{\langle \Gamma, S, E, \texttt{while}\ e\ \texttt{do}\ P \rangle \to \langle \Gamma, S, E, \texttt{nil} \rangle}$$

(EMIT-OP)
$$\langle \Gamma, S, E, \texttt{emit}\ a \rangle \to \langle \Gamma, S, \{a\} \cup E, \texttt{nil} \rangle$$

(LOCAL-OP)
$$\frac{b = \texttt{news}(\texttt{dom}(\Gamma))}{\langle \Gamma, S, E, \texttt{local}\ a : \delta\ \texttt{in}\ P \rangle \to \langle \{b : \delta\ \textit{sig}\}\Gamma, S, E, \{b/a\}P \rangle}$$

(WATCH-OP$_1$)
$$\langle \Gamma, S, E, \texttt{do nil watching}\ a \rangle \to \langle \Gamma, S, E, \texttt{nil} \rangle$$

(WATCH-OP$_2$)
$$\frac{\langle \Gamma, S, E, P \rangle \to \langle \Gamma', S', E', P' \rangle}{\langle \Gamma, S, E, \texttt{do}\ P\ \texttt{watching}\ a \rangle \to \langle \Gamma', S', E', \texttt{do}\ P'\ \texttt{watching}\ a \rangle}$$

(WHEN-OP$_1$)
$$\frac{a \in E}{\langle \Gamma, S, E, \texttt{when}\ a\ \texttt{do}\ \texttt{nil} \rangle \rightarrow \langle \Gamma, S, E, \texttt{nil} \rangle}$$

(WHEN-OP$_2$)
$$\frac{a \in E \quad \langle \Gamma, S, E, P \rangle \rightarrow \langle \Gamma', S', E', P' \rangle}{\langle \Gamma, S, E, \texttt{when}\ a\ \texttt{do}\ P \rangle \rightarrow \langle \Gamma', S', E', \texttt{when}\ a\ \texttt{do}\ P' \rangle}$$

(PAR-OP$_1$)
$$\langle \Gamma, S, E, \texttt{nil} \lhd Q \rangle \rightarrow \langle \Gamma, S, E, Q \rangle$$

(PAR-OP$_2$)
$$\frac{\langle \Gamma, S, E, P \rangle \rightarrow \langle \Gamma', S', E', P' \rangle}{\langle \Gamma, S, E, P \lhd Q \rangle \rightarrow \langle \Gamma', S', E', P' \lhd Q \rangle}$$

(PAR-OP$_3$)
$$\frac{\langle E, P \rangle \ddagger \quad \neg \langle E, Q \rangle \ddagger}{\langle \Gamma, S, E, P \lhd Q \rangle \rightarrow \langle \Gamma, S, E, Q \lhd P \rangle}$$

Figure 2: Operational semantics of moves

component until termination or suspension, and then gives control to its right component, provided this is not already suspended.

Example 1 (Alternating parallel composition) *In this example three threads are queuing for execution in an empty signal-environment (left column). Underbraces indicate suspension. The emission of signal a by the third process unblocks the first of the suspended processes, enabling them to execute one by one and then reach termination.*

	$\{\}$	$\underbrace{(\texttt{when}\ a\ \texttt{do}\ \texttt{emit}\ b) \lhd (\texttt{when}\ b\ \texttt{do}\ \texttt{emit}\ c)}) \lhd \texttt{emit}\ a$
$\rightarrow$	$\{\}$	$\texttt{emit}\ a \lhd ((\texttt{when}\ a\ \texttt{do}\ \texttt{emit}\ b) \lhd (\texttt{when}\ b\ \texttt{do}\ \texttt{emit}\ c))$
$\rightarrow^\star$	$\{a\}$	$(\texttt{when}\ a\ \texttt{do}\ \texttt{emit}\ b) \lhd (\texttt{when}\ b\ \texttt{do}\ \texttt{emit}\ c)$
$\rightarrow^\star$	$\{a, b\}$	$\texttt{when}\ b\ \texttt{do}\ \texttt{emit}\ c$
$\rightarrow^\star$	$\{a, b, c\}$	$\texttt{nil}$

We have seen that suspension of a thread may be lifted during an instant upon emission of the signal by another thread in the pool. This is no longer possible in a program in which all threads are suspended. When this situation is reached, an instant change occurs.

2.2.2 Instant changes

Suspension of a configuration marks the end of an instant. At this point, all signals are reset to absent (the new signal-environment is the empty set) and all the suspended

$$\text{(INSTANT-OP)}\quad \frac{\langle E, P\rangle\ddagger}{\langle \Gamma, S, E, P\rangle \hookrightarrow \langle \Gamma, S, \emptyset, \lfloor P \rfloor_E\rangle}$$

$$\lfloor \texttt{do}\ P\ \texttt{watching}\ a \rfloor_E \stackrel{\text{def}}{=} \begin{cases} \texttt{nil} & \text{if } a \in E \\ \texttt{do}\ \lfloor P \rfloor_E\ \texttt{watching}\ a & \text{otherwise} \end{cases}$$

$$\lfloor P\,;\, Q \rfloor_E \stackrel{\text{def}}{=} \lfloor P \rfloor_E\,;\, Q$$

$$\lfloor \texttt{when}\ a\ \texttt{do}\ P \rfloor_E \stackrel{\text{def}}{=} \begin{cases} \texttt{when}\ a\ \texttt{do}\ \lfloor P \rfloor_E & \text{if } a \in E \\ \texttt{when}\ a\ \texttt{do}\ P & \text{otherwise} \end{cases}$$

$$\lfloor P \curvearrowleft Q \rfloor_E \stackrel{\text{def}}{=} \lfloor P \rfloor_E \curvearrowleft \lfloor Q \rfloor_E$$

Figure 3: Operational semantics of instant changes

subprocesses of the form $\texttt{do}\ P\ \texttt{watching}\ a$ whose watched signal is present are killed. Indeed, the `watching` construct provides a way to recover from the deadlock situation. The semantics of *instant changes* is defined in Figure 3. The function $\lfloor P \rfloor_E$ is meant to be applied to suspended processes (see Figure 1) and therefore is only defined for them.

Instant changes are programmable; we hinted in the Introduction the possibility of encoding a primitive that enforces suspension of a thread until instant change. This primitive, which we call `pause`, is defined as follows.

Example 2 (`pause`) *Here the local declaration of signal a ensures that the signal cannot be emitted outside the scope of its declaration, and therefore that the program will suspend. At this point, the presence of b is checked, and since it has been emitted, the subprogram* $(\texttt{when}\ a\ \texttt{do}\ \texttt{nil})$*, where a is replaced by a fresh variable d, is aborted (i.e. turned into* `nil`*) at the beginning of the next instant.*

$\{\} \mid \texttt{local}\ a : \delta\ \texttt{in}\ (\texttt{local}\ b : \theta\ \texttt{in}\ (\texttt{emit}\ b\,;\, \texttt{do}\ (\texttt{when}\ a\ \texttt{do}\ \texttt{nil})\ \texttt{watching}\ b))$

$$\begin{array}{lll|l} \rightarrow^\star & \{\} & & \texttt{emit}\ b\,;\, \texttt{do}\ (\texttt{when}\ a'\ \texttt{do}\ \texttt{nil})\ \texttt{watching}\ b' \\ \rightarrow^\star & \{b\} & & \underbrace{\texttt{do}\ \underbrace{(\texttt{when}\ a'\ \texttt{do}\ \texttt{nil})}\ \texttt{watching}\ b'}} \\ \hookrightarrow & \{\} & & \texttt{nil} \end{array}$$

The following is a program where an instant change breaks a causality cycle:

$$\texttt{emit}\ a\,;((\texttt{when}\ b\ \texttt{do}\ \texttt{emit}\ c) \curvearrowleft (\texttt{do}\ (\texttt{when}\ c\ \texttt{do}\ \texttt{emit}\ b)\ \texttt{watching}\ a)\,;\ \texttt{emit}\ b)$$

Here the whole program suspends after the emission of a. Then, since a is present, the body of the `watching` construct is killed and a new instant starts, during which b is emitted, thus unblocking the other thread and allowing c to be emitted.

2.2.3 Execution paths

A computation of a configuration has the form:

$$\langle \Gamma, S, E, P\rangle \to^\star \langle \Gamma_1, S_1, E_1, P_1\rangle \hookrightarrow \langle \Gamma_1, S_1, \emptyset, \lfloor P_1 \rfloor_{E_1}\rangle \to^\star \langle \Gamma_2, S_2, E_2, P_2\rangle \hookrightarrow \dots$$

One can check that every configuration is able to perform a step if and only if its program is different from `nil`. The form of this step, simple move or instant change, depends on whether P is suspended or not. Moreover, the following result establishes the uniqueness of this step.

Theorem 2.1 (Determinism) *Any configuration $C = \langle \Gamma, S, E, P\rangle$ is in exactly one of three states:* terminated*, if $P = \texttt{nil}$;* suspended*, if $\langle E, P\rangle\ddagger$, in which case $\exists! C' : C \hookrightarrow C'$;* active*, in which case $\exists! C' : C \to C'$.*

3 Noninterference

3.1 Type System

We now introduce our type system, whose role is to rule out insecure programs. In the Introduction we illustrated the notion of implicit flow in sequential programs. Reactive constructs should also forbid "low writes" after "high tests", as in
$\texttt{when}\ a_H\ \texttt{do}\ \texttt{emit}\ b_L$. To see this, consider the program

$$\texttt{emit}\ c_L\,;\ (\texttt{do}\ (\texttt{when}\ a_H\ \texttt{do}\ \texttt{emit}\ b_L)\ \texttt{watching}\ c_L) \tag{3}$$

Whether a_H is present or not this program always terminates (in one or two instants respectively), emitting b_L only if a_H is present. Also the more subtle program (1) has its reactive counterpart $(\texttt{when}\ a_H\ \texttt{do}\ \texttt{nil})\,;\ \texttt{emit}\ b_L$. Again, this could be viewed as safe when run in isolation. However, when composed with other threads, this program can be source of interferences as suspension can be lifted by the emission of signal a_H. Consider for instance the program $\gamma \triangleleft (\alpha \triangleleft \beta)$ (which is the reactive analogue of the `PIN` example of [15, 5]), where

$$\begin{aligned}\gamma &: \texttt{if}\ \texttt{PIN}_H = 0\ \texttt{then}\ \texttt{emit}\ a_H\ \texttt{else}\ \texttt{emit}\ b_H\\ \alpha &: \texttt{when}\ a_H\ \texttt{do}\ \texttt{nil}\,;\ \texttt{emit}\ c_L\,;\ \texttt{emit}\ b_H\\ \beta &: \texttt{when}\ b_H\ \texttt{do}\ \texttt{nil}\,;\ \texttt{emit}\ d_L\,;\ \texttt{emit}\ a_H\end{aligned}$$

If $\texttt{PIN}_H = 0$, no suspension occurs: α is executed before β, and c_L is emitted before d_L. If $\texttt{PIN}_H \neq 0$, α is initially suspended and β is executed first, emitting d_L and then unblocking α. In this case c_L is emitted after d_L.

Note that imperative constructs with high tests, followed by "low writes" (as in the imperative program (1) of the Introduction) remain problematic in a reactive setting, for we can write the program $\gamma' \curlywedge (\alpha' \curlywedge \beta')$:

$$\gamma' : \texttt{if PIN}_H = 0 \texttt{ then } x_H := 0 \texttt{ else } x_H := 1$$
$$\alpha' : (\texttt{while } x_H = 0 \texttt{ do pause}\,;\, r_L := 0\,;\, x_H := 0)$$
$$\beta' : (\texttt{while } x_H = 1 \texttt{ do pause}\,;\, r_L := 1\,;\, x_H := 1)$$

Reactive concurrency introduces new leaks that are not exhibited with the usual asynchronous concurrency.
Consider the program $(\gamma'' \curlywedge \alpha'') \curlywedge \beta''$, running in two different environments, $E_1 = \{a_H, z_H\}$ and $E_2 = \{z_H\}$:

$$\gamma'' : (\texttt{pause}\,;\, x_L := 1)$$
$$\alpha'' : (\texttt{do } (\texttt{when } a_H \texttt{ do nil}) \texttt{ watching } z_H \curlywedge \texttt{when } b_L \texttt{ do } x_L := 0)$$
$$\beta'' : (\texttt{nil}\,;\, \texttt{pause}\,;\, \texttt{emit } b_L)$$

The threads γ'' and α'' are running in parallel, containing different low assignments on x_L. Notice that while γ'' starts by suspending itself, the thread α'' suspends if and only if signal a_H is absent. Therefore, only in an environment where a_H is present, will γ'' and α'' start by switching positions. The `nil` component in β'' (although technically redundant) stresses that this thread gains control before suspension, thus ensuring that the first thing that happens after the change of instant is the emission of signal b_L. What remains to be done is the two assignments to the low variable x_L, but the order in which they will happen depends in the order of appearance of the continuations of γ'' and α'' (which as we've seen depends on the the presence of the high signal a_H). This example suggests that it is possible to make the order of execution of neighboring threads depend on high signals. The tree of threads can then be seen as the body of a conditional, where any high test is its potential guard. This motivates the introduction of some extra conditions in the typing of reactive concurrency, similar to those used for conditionals in [5, 16].

Let us now present our type system. As we mentioned in Section 2, expressions will be typed with *simple types*, which are just security levels δ, θ, σ. As usual, these are assumed to form a lattice $(\mathcal{T}, \leq)$, where the order relation $\leq$ stands for "less secret than" and $\wedge, \vee$ denote meet and join. Starting from simple types we build *variable types* of the form δ *var* and *signal types* of the form δ *sig*. Program types will have the form (θ, σ) *cmd*, as in [5, 16]. Here the first component θ represents a lower bound on the level of written variables and emitted signals, while the second component σ is an upper bound on the level of tested variables and signals.

Our type system is presented in Figure 4. It is analogous to the one in [5, 16], apart from the new restrictions on parallel composition. The types for the `when` and `watching` commands are similar to those for the `while` command, since their semantics also consists of the execution of a process under a guard. As regards reactive

(NIL) $\Gamma \vdash \texttt{nil} : (\theta, \sigma)\ \mathit{cmd}$

(ASSIGN) $$\frac{\Gamma \vdash e : \theta \quad \Gamma(x) = \theta\ \mathit{var}}{\Gamma \vdash x := e : (\theta, \sigma)\ \mathit{cmd}}$$

(LET) $$\frac{\Gamma \vdash e : \delta \qquad \{x : \delta\ \mathit{var}\}\Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd}}{\Gamma \vdash \texttt{let}\ x : \delta = e\ \texttt{in}\ P : (\theta, \sigma)\ \mathit{cmd}}$$

(SEQ) $$\frac{\Gamma \vdash Q_1 : (\theta_1, \sigma_1)\ \mathit{cmd} \quad \Gamma \vdash Q_2 : (\theta_2, \sigma_2)\ \mathit{cmd} \quad \sigma_1 \leq \theta_2}{\Gamma \vdash Q_1\,;\, Q_2 : (\theta_1 \wedge \theta_2, \sigma_1 \vee \sigma_2)\ \mathit{cmd}}$$

(COND) $$\frac{\Gamma \vdash e : \delta \quad \Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd} \quad \Gamma \vdash Q : (\theta, \sigma)\ \mathit{cmd} \quad \delta \leq \theta}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q : (\theta, \delta \vee \sigma)\ \mathit{cmd}}$$

(WHILE) $$\frac{\Gamma \vdash e : \delta \quad \Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd} \quad \delta \vee \sigma \leq \theta}{\Gamma \vdash \texttt{while}\ e\ \texttt{do}\ P : (\theta, \delta \vee \sigma)\ \mathit{cmd}}$$

(EMIT) $$\frac{\Gamma(a) = \theta\ \mathit{sig}}{\Gamma \vdash \texttt{emit}\ a : (\theta, \sigma)\ \mathit{cmd}}$$

(LOCAL) $$\frac{\{a : \delta\ \mathit{sig}\}\Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd}}{\Gamma \vdash \texttt{local}\ a : \delta\ \texttt{in}\ P : (\theta, \sigma)\ \mathit{cmd}}$$

(WATCH) $$\frac{\Gamma(a) = \delta\ \mathit{sig} \quad \Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd} \quad \delta \leq \theta}{\Gamma \vdash \texttt{do}\ P\ \texttt{watching}\ a : (\theta, \delta \vee \sigma)\ \mathit{cmd}}$$

(WHEN) $$\frac{\Gamma(a) = \delta\ \mathit{sig} \quad \Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd} \quad \delta \leq \theta}{\Gamma \vdash \texttt{when}\ a\ \texttt{do}\ P : (\theta, \delta \vee \sigma)\ \mathit{cmd}}$$

(PAR) $$\frac{\Gamma \vdash P : (\theta_1, \sigma_1)\ \mathit{cmd} \quad \Gamma \vdash Q : (\theta_2, \sigma_2)\ \mathit{cmd} \quad \sigma_1 \leq \theta_2 \quad \sigma_2 \leq \theta_1}{\Gamma \vdash P \curlywedge Q : (\theta_1 \wedge \theta_2, \sigma_1 \vee \sigma_2)\ \mathit{cmd}}$$

(SUB) $$\frac{\Gamma \vdash P : (\theta, \sigma)\ \mathit{cmd} \quad \theta \geq \theta' \quad \sigma \leq \sigma'}{\Gamma \vdash P : (\theta', \sigma')\ \mathit{cmd}}$$

(EXPR) $$\frac{\forall x_i \in \texttt{fv}(e).\delta \geq \theta_i \text{ where } \Gamma(x_i) = \theta_i\ \mathit{var}}{\Gamma \vdash e : \delta}$$

Figure 4: Typing Rules

parallel composition, the side conditions express the fact that any high test is a potential guard to the order of execution of concurrent threads. It forbids a thread from performing assignments and emissions at levels that are not higher than or equal to those of the tested variables and signals of parallel threads.

One may notice that these side conditions restrict the compositionality of the type system and introduce some overhead (two comparisons of security levels) when adding new threads in the system. This is the price we pay for allowing loops with high guards such as $\texttt{while}\ x_H = 0\ \texttt{do}\ \texttt{nil}$ (which are rejected by previous type systems, as [15, 14]) in the context of a co-routine mechanism. However, it might be worth examining if this restriction could be lifted to some extent by means of techniques proposed for other concurrent languages ([11, 12]).

3.2 Properties of typed programs

It is easy to see from the semantic rules that computation may affect a type-environment only by extending it with fresh names. Our first result states that types are preserved along execution.

Theorem 3.1 (Subject Reduction) *If* $\Gamma \vdash P : (\theta, \sigma)$ *cmd and* $\langle \Gamma, S, E, P\rangle \longmapsto \langle \Gamma', S', E', P'\rangle$ *then* $\Gamma' \vdash P' : (\theta, \sigma)$ *cmd.*

Our next result ensures that program types have the intended properties. We use the generic term "guard" for either a tested variable or a tested signal.

Proposition 3.2 (Guard Safety and Confinement)

1. *If* $\Gamma \vdash P : (\theta, \sigma)$ *cmd then every guard in* P *has type* $\delta \leq \sigma$.
2. *If* $\Gamma \vdash P : (\theta, \sigma)$ *cmd then every variable assigned to in* P *and every signal emitted in* P *has security level* δ *(that is type* δ *var or* δ *sig, respectively) with* $\theta \leq \delta$.

We now introduce some terminology that will be useful to define our notion of indistinguishability. We use $\mathcal{L}$ to designate a *downward-closed set of security levels*, that is a set $\mathcal{L} \subseteq \mathcal{T}$ satisfying $\theta \in \mathcal{L}\ \&\ \sigma \leq \theta \Rightarrow \sigma \in \mathcal{L}$. The *low memory* is the portion of the variable-store and signal-environment to which the type-environment associates "low security levels" (i.e. security levels in $\mathcal{L}$). Two memories are said to be low-equal if their low parts coincide:

Definition 3.1 ($\mathcal{L}, \Gamma$-equality of Memories and Configurations) $\langle S, E\rangle =^{\Gamma}_{\mathcal{L}} \langle R, F\rangle \stackrel{def}{\Leftrightarrow} \forall x.\Gamma(x) = \theta\ \textit{var}\ \&\ \theta \in \mathcal{L} \Rightarrow S(x) = R(x)$ *and* $\forall a.\Gamma(a) = \theta\ \textit{sig}\ \&\ \theta \in \mathcal{L} \Rightarrow a \in E \Leftrightarrow a \in F$. *Two configurations are said to be low-equal when they have low-equal memories.*

There is a class of programs for which the security property is trivial to establish because of their inability to change low memory. As usual, we will refer to these as *high programs*. Here we distinguish two classes of high programs based on a syntactic, respectively semantic, analysis:

Definition 3.2 (High Reactive Programs)

1. Syntactically high programs $\mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{syn}}$ *is inductively defined by:* $P \in \mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{syn}}$ *if*

- $P = (x := e)$, *and* $\Gamma(x) = \theta$ *var* $\Rightarrow \theta \notin \mathcal{L}$.
- $P = (\mathtt{emit}\ a)$, *and* $\Gamma(x) = \theta$ *sig* $\Rightarrow \theta \notin \mathcal{L}$.
- $P = \mathtt{let}\ x : \delta = e\ \mathtt{in}\ Q$, *and* $Q \in \mathcal{H}^{\Gamma\cup\{x:\delta\ var\},\mathcal{L}}_{\mathtt{syn}}$, *and*
- $P = \mathtt{local}\ a : \delta\ \mathtt{in}\ Q$, *and* $Q \in \mathcal{H}^{\Gamma\cup\{a:\delta\ sig\},\mathcal{L}}_{\mathtt{syn}}$, *and*
- $P = (Q_1\,;\,Q_2)$, $P = (\mathtt{if}\ e\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2)$, $P = (\mathtt{while}\ e\ \mathtt{do}\ Q_1)$, $P = (\mathtt{when}\ a\ \mathtt{do}\ Q_1)$, $P = (\mathtt{do}\ Q_1\ \mathtt{watching}\ a)$, *or* $P = (Q_1 \leftcurvearrow Q_2)$, *where* $Q_i \in \mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{syn}}$ *for* $i = 1, 2$.

2. Semantically high programs $\mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{sem}}$ *is coinductively defined by* $P \in \mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{sem}}$ *implies:*

- $\forall S, E, \langle \Gamma, S, E, P\rangle \rightarrow \langle \Gamma', S', E', P'\rangle$ *implies* $\langle S, E\rangle =^{\Gamma}_{\mathcal{L}} \langle S', E'\rangle$ *and* $P' \in \mathcal{H}^{\Gamma',\mathcal{L}}_{\mathtt{sem}}$, *and*
- $\forall S, E, \langle \Gamma, S, E, P\rangle \hookrightarrow \langle \Gamma', S', E', P'\rangle$ *implies* $P' \in \mathcal{H}^{\Gamma',\mathcal{L}}_{\mathtt{sem}}$

Note that $P = \mathtt{let}\ x : \delta = e\ \mathtt{in}\ Q$ (as well as $P = \mathtt{local}\ a : \delta\ \mathtt{in}\ Q$) is considered syntactically high even if $\delta \notin \mathcal{L}$, provided Q is syntactically high in the extended typing environment. It may be shown that both properties are preserved by execution. As argued in the Introduction, we do not consider as relevant the initialization of the low signal environment that is induced by instant changes. This is reflected by the absence of the low equality condition after instant changes in Definition 3.2.2 (recall that the variable-store S is not modified during an instant change). Since instant changes are not statically predictable, this assumption allows us to deduce, from the syntactic property of being a high program, the corresponding behavioural property. In other words, the set of syntactically high programs is a subset of the semantically high ones, that is $\mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{syn}} \subseteq \mathcal{H}^{\Gamma,\mathcal{L}}_{\mathtt{sem}}$. An example of a semantically high program that is not syntactically high is $\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{nil}\ \mathtt{else}\ y_L := 0$.

The key result for proving noninterference is the following theorem, whose proof is quite elaborate and therefore omitted here (it uses the notions of $\mathcal{L}$-boundedness and $\mathcal{L}$-guardedness as in [5] and some intermediate results). This result states that typed programs which immediately fork because of a high test, are syntactically high. Therefore, we are sure that the only changes in low memory occurring beyond such a fork occur at instant changes, and do not involve variables.

Theorem 3.3 (Forking programs) *Let* P *be typable in* Γ, $\langle S, E\rangle =^{\Gamma}_{\mathcal{L}} \langle R, F\rangle$, $\langle \Gamma, S, E, P\rangle \longmapsto \langle \Gamma_1, S_1, E_1, P_1\rangle$, $\langle \Gamma, R, F, P\rangle \longmapsto \langle \Gamma_2, R_2, F_2, P_2\rangle$,

and $\langle S_1, E_1\rangle \neq_{\mathcal{L}}^{\Gamma_1\cap\Gamma_2} \langle R_2, F_2\rangle$ *or* $\langle \Gamma_1, P_1\rangle \neq \langle \Gamma_2, P_2\rangle$ *or* $\langle E, P\rangle\ddagger$ *iff* $\neg\langle F, P\rangle\ddagger$. *Then* $P \in \mathcal{H}_{\mathtt{syn}}^{\Gamma,\mathcal{L}}$. *(*[1]*)*

This is quite a strong result, for it says that if there is any difference in the first step of two computations of a typable program under low-equal memories, then the program code contains no low-assignments or emissions. The intuition is that such a difference reflects the passage of a high test, and our type system guarantees that no changes in low memory will follow. Since the type system is based on a syntactical analysis, the result uses the syntactic notion of high program. For an example of a forking program which becomes syntactically high (after one step of computation) see (2) in the Introduction.

3.3 Security notion and soundness of the type system

In this section we define reactive bisimulation, and prove our noninterference result with respect to it. The two-level nature of reactive computations, together with the asymmetry in signal removal and emission, poses some challenges in the definition of noninterference. However, these subtleties can be factored out through the notion of high program.

Definition 3.3 ($\mathcal{L}$-Bisimulation equivalence ($\approx_{\mathcal{L}}$)) *The equivalence* $\approx_{\mathcal{L}}$ *is the largest symmetric relation* $\mathcal{R}$ *such that* $C_1 \mathcal{R} C_2$, *where* $C_1 = \langle \Gamma_1, S_1, E_1, P_1\rangle$ *and* $C_2 = \langle \Gamma_2, S_2, E_2, P_2\rangle$, *imply:*

- $C_1 =_{\mathcal{L}}^{\Gamma_1\cap\Gamma_2} C_2$, *and*
- *either*
  - *a.* $P_i \in \mathcal{H}_{\mathtt{sem}}^{\Gamma_i,\mathcal{L}}$ *for* $i = 1, 2$, *or*
  - *b.* $C_1 \longmapsto C_1'$ *implies* $\exists C_2'$ *such that* $C_2 \longmapsto^{\star} C_2'$ *and* $C_1' \mathcal{R} C_2'$

We can now formalize the notion of *secure program* in the usual way:

Definition 3.4 (Γ-Secure Programs) P *is secure in the typing context* Γ *if for any downward-closed set* $\mathcal{L}$ *of security levels and for any* S_1, E_1, S_2, E_2 *such that* $\langle S_1, E_1\rangle =_{\mathcal{L}}^{\Gamma} \langle S_2, E_2\rangle$ *then* $\langle \Gamma, S_1, E_1, P\rangle \approx_{\mathcal{L}} \langle \Gamma, S_2, E_2, P\rangle$.

Finally, we are in position to prove that every typable program is secure:

Theorem 3.4 (Noninterference) *If* P *is typable in* Γ *then* P *is* Γ*-secure.*

Proof 3.4 *For any* $\mathcal{L}$, *define the relation* $\mathcal{S}_{\mathcal{L}}$ *on configurations* $C_1 = \langle \Gamma_1, S_1, E_1, P_1\rangle$ *and* $C_2 = \langle \Gamma_2, S_2, E_2, P_2\rangle$, *such that* $C_1\ \mathcal{S}_{\mathcal{L}}\ C_2$ *if and only if:*

[1] Note that the case where $\langle S_1, E_1\rangle \neq_{\mathcal{L}}^{\Gamma} \langle R_2, F_2\rangle$ attends the case where only one of the computations performs an instant change and $\exists a_L \in E \cup F$.

- P_i *is typable in* Γ_i *for* $i = 1, 2$*, and*
- $C_1 =_{\mathcal{L}}^{\Gamma_1 \cap \Gamma_2} C_2$*, and*
- *either*
  - *i.* $P_i \in \mathcal{H}_{\mathtt{syn}}^{\Gamma_i, \mathcal{L}}$ *for* $i = 1, 2$*, or*
  - *ii.* $\langle \Gamma_1, P_1 \rangle = \langle \Gamma_2, P_2 \rangle$.

Note first that if P *is typable in* Γ *and* $\langle S_1, E_1 \rangle =_{\mathcal{L}}^{\Gamma} \langle S_2, E_2 \rangle$*, then* $\langle \Gamma, S_1, E_1, P \rangle \; \mathcal{S}_{\mathcal{L}} \; \langle \Gamma, S_2, E_2, P \rangle$ *by clause ii. Next we prove that* $\mathcal{S}_{\mathcal{L}} \subseteq \approx_{\mathcal{L}}$*. Suppose that* $C_1 \; \mathcal{S}_{\mathcal{L}} \; C_2$*. Then we have* $C_1 =_{\mathcal{L}}^{\Gamma_1 \cap \Gamma_2} C_2$ *by hypothesis, and either:*

1. *Both* $P_i \in \mathcal{H}_{\mathtt{syn}}^{\Gamma_i, \mathcal{L}}$ *for* $i = 1, 2$ *by clause i. Since* $\mathcal{H}_{\mathtt{syn}}^{\Gamma_i, \mathcal{L}} \subseteq \mathcal{H}_{\mathtt{sem}}^{\Gamma_i, \mathcal{L}}$ *then* $P_i \in \mathcal{H}_{\mathtt{sem}}^{\Gamma_i, \mathcal{L}}$.
2. $\langle \Gamma_1, P_1 \rangle = \langle \Gamma_2, P_2 \rangle$ *by clause ii. We may assume that* $P_i \notin \mathcal{H}_{\mathtt{syn}}^{\Gamma_i, \mathcal{L}}$*, since otherwise we would fall into the previous point. If* $C_1 \longmapsto C_1' = \langle \Gamma_1', S_1', E_1', P_1' \rangle$*, then* $P_1 \neq \mathtt{nil}$*. Since* $P_1 = P_2$*, also* $P_2 \neq \mathtt{nil}$ *and by Theorem 2.1* $\exists! C_2'$ *such that* $C_2 \longmapsto C_2' = \langle \Gamma_2', S_2', E_2', P_2' \rangle$*. By Theorem 3.3, we conclude that* $C_1' =_{\mathcal{L}}^{\Gamma_1' \cap \Gamma_2'} C_2'$ *and* $\langle \Gamma_1', P_1' \rangle = \langle \Gamma_2', P_2' \rangle$*. Hence,* $C_1' \; \mathcal{S}_{\mathcal{L}} \; C_2'$ *by clause ii.*

Let us now return to the program $P = \mathtt{if}\ x_H = 0\ \mathtt{then}\ \mathtt{nil}\ \mathtt{else}\ \mathtt{pause}$, so that we can understand the subtleties in this apparently straightforward definition of security. As can be seen from the definition of bisimulation, this program is accepted as secure for the reason that it is high. Let us argue why such programs do not pose security problems. Although we have not pursued this question formally here, our motivation stems from the following reasoning. Consider the program $P \triangleleft Q$, where Q would be an observer which we assume to be confined to the lowest level (and therefore typable, although the composition need not be). When control is given to Q, he is not able to know whether this is due to the suspension or to the termination of P. In particular, if Q starts by suspending itself, an instant change will occur in both cases. Unable to distinguish the two possible execution paths that P might have taken, Q cannot perform different actions accordingly. For similar reasons we do not consider necessary to require the synchronization of instant changes for matching two computations of a program.

4 Conclusion and related work

In this paper we have addressed the question of noninterference for reactive programs. We have presented a type system guaranteeing noninterference in a core imperative reactive language. We are currently studying a call-by-value language for mobility built around a reactive core, called ULM [3]. We intend to adapt to this language the techniques we have developed here.

As has been observed, reactive programs obey a fixed scheduling policy, which is enforced syntactically using the parallel construct $\curlywedge$. Other approaches to noninterference in the presence of scheduling include the probabilistic one, proposed for instance in [17] and [14]. In these papers scheduling is introduced at the semantic level (adding probabilities to the transitions), and security is formalized through a notion of probabilistic noninterference. It should be noted that, unlike [5], which allows to express different scheduling policies, and [14] which accounts for an arbitrary scheduler (satisfying some reasonable properties), here a fixed deterministic scheduling is in use. Indeed, the novelty of our work resides mainly in addressing the question of noninterference in a reactive scenario. The work [13] examines the impact of synchronization on information flow, and uses it as a means to study time leaks without explicitly introducing a scheduler. However the analogy cannot be pushed very far since [13] has no notion of instant (and thus no way of recovering from deadlocks) and uses asynchronous parallel composition.

References

[1] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *IEEE*, 91(1):64–83, 2003.

[2] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Sci. of Comput. Programming*, 19:87–152, 1992.

[3] G. Boudol. ULM, a core programming model for global computing. In *ESOP'04*, 2004.

[4] G. Boudol and I. Castellani. Noninterference for concurrent programs. In *ICALP'01*, number 2076 in Lecture Notes in Computer Science, pages 382–395, 2001.

[5] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109–130, 2002.

[6] G. Boudol, I. Castellani, and A. Matos. Typing noninterference for reactive programs. Full paper available at "http://www-sop.inria.fr/mimosa/personnel/ana.matos/tec-rep2004.ps", 2004.

[7] F. Boussinot. Reactive Programming, Software available at "http://www-sop.inria.fr/mimosa/rp/", 2003.

[8] J.-F. Susini F. Boussinot. The sugarcubes tool box: a reactive JAVA framework. *Software Practice and Experience*, 28(14):1531–1550, 1998.

[9] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings 1982 IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[10] N. Halbwachs. Synchronous programming of reactive systems, 1993.

[11] N. Yoshida K. Honda. A uniform type structure for secure information flow. In *Proceedings of the The 29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, 2002.

[12] A. Myers S. Zdancewic. Observational determinism for concurrent program security. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, 2003.

[13] A. Sabelfeld. The impact of synchronization on secure information flow in concurrent programs. In *Proceedings of Andrei Ershov 4th International Conference on Perspectives of System Informatics*, 2001.

[14] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *13th Computer Security Foundations Workshop*. IEEE, 2000.

[15] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings POPL '98*, pages 355–364. ACM Press, 1998.

[16] Geoffrey Smith. A new type system for secure information flow. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 115–125. ACM Press, June 2001.

[17] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7(2-3), 1999.

[18] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

Principals, Policies and Keys in a Secure Distributed Programming Language

(Extended Abstract)

Tom Chothia
Department of Computer Science
Stevens Institute of Technology
Hoboken, New Jersey 07030, USA.
tomc@cs.stevens-tech.edu

Dominic Duggan
Department of Computer Science
Stevens Institute of Technology
Hoboken, New Jersey 07030, USA.
dduggan@cs.stevens-tech.edu

Jan Vitek
Dept of Computer Science
Purdue University
West Lafayette, IN 47907, USA.
jv@cs.purdue.edu

Abstract

KDLM is a model of information flow control for distributed programming languages, introducing a distributed notion of selective declassification. KDLM uses type-level notions of both principals and policies to model security policies incorporating access control and information flow control, with run-time representatives based on cryptographic keys. This article addresses the question of how this control is enforced when programs cross address spaces and networks. An object calculus is introduced that extends KDLM. This uses a notion of locally opaque principal and key names in objects to transmit and enforce restrictions across address spaces, with key equality used as a form of runtime type discrimination to reveal "opaque" principal names at the receiver.

1 Introduction

Language-based security is becoming an increasingly important aspect of computer security. Large amounts of data are being processed by computer programs and there is a need for control and accountability in the way these programs handle the information they glean from the data. The computer security community has focussed on access control mechanisms for databases of sensitive data. The programming language community has focussed on information flow control: making sure that sharing of data is controlled in some manner to ensure that it does not leak in undesirable ways. The

programming language community has focussed on information flow control because language technology such as type systems can be fruitfully applied to this problem [25].

The Decentralized Label Model (DLM) is a model of information flow control that was introduced by Myers and Liskov [20]. This model avoids one undesirable aspect of classical information flow control, the need for some centrally defined lattice of information levels, by implicitly defining a lattice based on access control. It also introduced *selective declassification*, allowing a principal to relax its own access control and in this way relax information flow control. This relies critically on the notion of the principal for the currently executing code.

More recently the authors proposed the Key-based Decentralized Label Model (KDLM) as a model that combines information flow control and application-based cryptography for secure network programming [6]. The argument for this approach is the usual end-to-end argument in system design: it is ultimately unrealistic to expect a single one-size-fits-all solution to network security in the runtime. The application must be able to build its own network security stack for any approach to scale, yet the type system must prevent the application from violating the information flow guarantees in the type system while establishing the network security. KDLM combined ideas from DLM and type-based cryptographic APIs [10]. In particular KDLM introduced the notion of key names, type level entities representing encryption and signing keys, as well as principals as in DLM. A particular motivation for KDLM was *distributed declassification*.

A question that is posed by KDLM is, what is the nature of principal names and key names in programs that may cross address spaces? All modelling for security purposes is done in the type system, with any runtime checks (beyond the static type-based checking) based on keys. We can certainly formulate examples with principals such as `Alice` and `Bob`, but these merely refer to compile-time type names within an address space. How do two separate programs come to agreement on the identities of their principals?

In this article we extend KDLM with an object calculus, motivated by its ultimate application in an object-oriented programming language. We call this language Jeddak, it extends Java. Since principals (and key names) must be modelled as type-level entities, objects carry type-level fields via an adaptation of the form of generics that have been added to Java and are part of JDK 1.5 [4, 5, 27]. In this approach, the names of keys and principals are treated as abstract data types for transmission of data, with key equality used as a form of runtime type discrimination to reveal “opaque” principal names at the receiver.

In Sect. 2 we explain with some examples how our approach can be used to support key distribution while enforcing access control across address spaces. In Sect. 3 we provide a formal type system. Finally Sect. 4 considers related work and conclusions. Due to space considerations, several details are omitted from this extended abstract. They are available as a technical report in a longer version of the paper [7].

2 Key Distribution

The Jeddak type system models notions of key-based access control. So it includes notions of *principal names* and *policy names* that are analogous to types. Whereas the latter may be loosely interpreted as sets of values, the former represent abstract modelling entities whose runtime representatives are actual keys. The motivation for this approach is that much access control can be checked at compile-time. This in turn enables lightweight access control checking that facilitates a fine-grained audit trail for checking accountability.

However this compile-time checking cannot be extended across a distributed system, where values cross process address spaces. An analogy may be made here with RMI programming. Code executing within an address space is fully type-checked. Dynamic typing is used to ensure the type-safety of values that are transmitted between address spaces. Unmarshalling ensures that values are well-formed, and checked downcasting ensures that values have the application-specific types required at receiving sites.

Jeddak includes analogous mechanisms for "hiding" access information at a sending site and "exposing" that access information at the receiving site. The exposure operation requires runtime checking, but based on key identity rather than checked downcasting. We illustrate this by showing how keys with limited distribution may be shared between processes in a safe manner.

Types for values include *labelled types*, of the form

$$\{K_1,\ldots,K_n\}T.$$

The type T is a type in the normal sense. The set $\{K_1,\ldots,K_n\}$ is a *label*. In DLM, a label is a set of security policies. In KDLM, a label is a set of *policy names*, i.e., names for security policies. This extra level of indirection is motivated by mechanisms for distributed declassification, as explained in [6]. A policy is essentially an access control list, of the form

$$[P:\{P_1,\ldots,P_k\}]$$

where $P,P_1,\ldots,P_k$ are *principal names*. This policy is controlled by the principal P. This notion of control of a policy by a principal is useful for reclassification purposes: a principal can allow information controlled by one policy to "flow" (via assignment) to a variable controlled by a different policy, provided that principal controls both policies. The policy itself restricts access to a variable to the principals $\{P_1,\ldots,P_k\}$. A policy may be either to restrict "read" rights (to control secrecy) or to restrict "write" rights (to control integrity) on a variable. Since access to a variable is controlled by a set of such policies, access to that variable is restricted to the principals in the intersection of the policies making up the label.

For example if we have

`K1` specifies policy `[Joe:{Joe,Mary,Sam}]`

```
K2 specifies policy [Mary:{Mary,Sue,Sam}]
{K1,K2}int x;
```

Then the variable `x` is only accessible to the principals `Mary` and `Sam`.

Policy names only exist at compile-time, for modelling the security policies of the program. At run-time policy names may be represented by run-time witnesses, given by public-private key pairs. For a secrecy policy name, such a pair corresponds to a public encryption key and its corresponding private decryption key. For an integrity policy name, such a pair corresponds to a public authentication key and its corresponding signing key. A policy name that has an associated key pair representing it at run-time is declared as an "actual" policy name (as opposed to a "virtual" policy name that is used only for modelling purposes at compile-time). Consider the following example:

```
spolicy MyPolicy [ThisPrin:{Mary,Sue}];
pubkey⟨MyPolicy⟩ myPublicKey;
{MyPolicy}privkey⟨MyPolicy⟩ myPrivateKey;
```

This example declares a policy names, `MyPolicy`, and two keys, `myPublicKey` and `myPrivateKey`. The keyword `spolicy` identifies this policy name as a secrecy policy name; dually the language has integrity policy names, identified by the keyword `ipolicy`. The policy `MyPolicy` is controlled by the principal for the currently executing code, identified by the keyword `ThisPrin`. We assume as in the Java security model that code is signed by a principal. The policy `MyPolicy` has a (unique) associated public-private key pair. Two variables are declared as part of an actual policy name declaration, and these variables are initialized with the public and private parts of the associated key pair. The parameterized types `pubkey`$\langle\_\rangle$ and `privkey`$\langle\_\rangle$ denote the types of encryption and decryption keys, respectively (where the policy name specifies a secrecy policy). These types are parameterized by the associated policy name. So the public encryption key for the policy name `MyPolicy` has type `pubkey`$\langle$`MyPolicy`$\rangle$.

The decryption key has type `privkey`$\langle$`MyPolicy`$\rangle$. It must also have a label that is consistent with the policy of the associated policy name. The policy of the policy name `MyPolicy` restricts read access to variables to the principals `Mary` and `Sue` (and implicitly `ThisPrin`). If data with this access restriction is subsequently encrypted for transmission over a network, we must ensure that the private decryption key is not distributed to principals other than `Mary`, `Sue` and `ThisPrin`.

Now we have an apparent problem with actual policy names. Since such policy names can only be declared locally in a program, never at top level, how can we ever return a result whose type involves actual policy names? Consider for example returning a newly generated public-private key pair as the result of a method. To do this, we must return not only the key pair, but also the key name that they are associated with. We have so far talked of methods that take policy names as inputs, similarly to type parameters. Now we must allow methods to return policy names as outputs[1].

[1] For the reader cognizant with type theory, input policy names are universally quantified while output policy names are existentially quantified [17].

The following example defines a class for objects that contain public-private key pairs. Each such object has a `publicKey` field and a `privateKey` field. The class abstracts over the policy name that these fields share, and the principal and principal set in the policy for the key name, and the private policy name that specifies access restrictions on the private key:

```
class KeyPair ⟨prin Owner,
               prinSet Objects,
               spolicy K [Owner:Objects] ⟩ {
   pubkey⟨K⟩ publicKey;
   {K}privkey⟨K⟩ privateKey;
   KeyPair (pubkey⟨K⟩ pub, {K}privkey⟨K⟩ priv) auth Owner {
               this.publicKey = pub;
               this.privateKey = priv;
   }
}
```

An instantiation of this class must include instantiations for all of the type-level parameters. We extend the positional Java syntax for instantiation with a field-based syntax that is useful for what follows, so an example instantiation is:

```
new KeyPair⟨Owner=O,Objects=Os,K=K⟩
           (publicKey,privateKey)
```

for some `O`, `Os` and `K`, and some public and private keys `publicKey` and `privateKey`, respectively. The type of the resulting object is:

```
KeyPair⟨Owner=O,Objects=Os,K=K⟩
```

Now there is a problem with this type. Suppose a new policy name is generated as part of the generation of a new pair, in a local context, and this object is to be returned from this context. But this return is not possible because the surrounding context must already know the name of the newly generated policy name `K`, which is impossible. Therefore Jeddak adopts a refinement of the JDK 1.5 notion of raw types: we allow some of the type parameters in a generic instantiation to be hidden. This partial hiding is the motivation for the field-based syntax for generic instantiation. We have the subtyping

$$\texttt{KeyPair⟨Owner=O,Objects=Os,K=K⟩} \leq \texttt{KeyPair⟨Owner=O,Objects=Os⟩}$$

So the identity of the locally generated policy name is hidden, but the access restrictions are still visible in the type of the resulting object. In the following example, a nullary method `newPair` generates a new key pair, as part of declaring a new local (actual) policy name, and returns the result in a `KeyPair` object:

```
KeyPair⟨Owner,Objects⟩ newPair
     ⟨prin Owner, prinSet Objects⟩ () auth Owner {
   spolicy K [Owner:{Objects}];
   pubkey⟨K⟩ publicKey;
   {K}decKey⟨K⟩ privateKey;
```

```
   return new KeyPair ⟨Owner=Owner,Objects=Objects,K=K⟩
                      (publicKey, privateKey);
}
```

Now we can say:

```
KeyPair⟨ThisPrin,{Mary,Sue}⟩ x =
   newPair ⟨ThisPrin,{Mary,Sue}⟩();
KeyPair⟨ThisPrin,{Mary,Sue}⟩ y =
   newPair ⟨ThisPrin,{Mary,Sue}⟩();
```

Now we have the danger that the keys in x and y may be considered interchangeable, since x and y have the same types. This is because we have "hidden" the policy names for the keys of x and y. In Jeddak the hidden types are considered to be *local abstract types*, that can actually be referred to as fields of the object. So we have the typings:

$$\texttt{x.publicKey} \in \texttt{pubkey}\langle\texttt{x.K}\rangle$$

$$\texttt{y.publicKey} \in \texttt{pubkey}\langle\texttt{y.K}\rangle$$

$$\texttt{x.privateKey} \in \{\texttt{x.K}\}\texttt{privkey}\langle\texttt{x.K}\rangle$$

So the policy names for x and y are distinguished (x.K and y.K, respectively). The encryption and decryption keys cannot be referred to independently of the policy name.

If a public key must be transmitted, it must be bundled up in another object that has a copy of the policy name:

```
class PubKey ⟨prin Owner,
              prinSet Objects,
              spolicy K [Owner:Objects] ⟩
   pubkey⟨K⟩ publicKey;
   PubKey (pubkey⟨K⟩ pub) auth Owner {
      this.publicKey = pub;
   }
}
```

Then we can make a copy of the public key for distribution as follows:

```
PubKey⟨ThisPrin,{Mary,Sue}⟩ z =
   new PubKey ⟨Owner=ThisPrin,Objects={Mary,Sue},K=x.K⟩
              (x.publicKey);
```

The typing of this is subtle. We have the following subtyping rule:

$$\texttt{PubKey}\langle\texttt{Owner=ThisPrin,Objects=}\{\texttt{Mary,Sue}\}\texttt{,K=x.K}\rangle \leq \texttt{PubKey}\langle\texttt{Owner=ThisPrin,Objects=}\{\texttt{Mary,Sue}\}\rangle.$$

The first type reveals that the K type field of the PubKey object is actually an alias for x.K. This is reflected by the type equality K=x.K in the type of the object. Such a type cannot escape the scope of the variable x. On the other hand, the second type does not reveal this alias. This type is less informative, but an object with this type can escape the scope of x. So the general strategy for initializing an object with type fields is to first give it the very precise type resulting from its instantiation, then use subtyping to "hide" the bindings of the type fields.

For example, assuming a default nullary constructor for the above clause, the above code can be elaborated (say for bytecode generation) as:

```
PubKey ⟨Owner=ThisPrin,Objects={Mary,Sue},K=x.K⟩ temp =
   new PubKey ⟨Owner=ThisPrin,Objects={Mary,Sue},K=x.K⟩ ();
temp.init (x.publicKey);
PubKey⟨Owner=ThisPrin,Objects={Mary,Sue}⟩ z = temp;
```

The argument type of temp.init is pubkey⟨temp.K⟩, but the type of temp also exports the type equality K=x.K, so we have temp.K=x.K. It is only because of this type equality that the constructor application is able to type check, since the argument has type pubkey⟨x.K⟩. So equality constraints between policy names are a crucial component of ensuring that copy constructors type-check.

An object of type

$$\texttt{KeyPair}\langle\texttt{Owner=ThisPrin,Objects=}\{\texttt{Mary,Sue}\}\rangle$$

still cannot be transmitted to another address space, since the principal names ThisPrin, Mary and Sue only make sense within an address space. We can certainly elide this type to

$$\texttt{KeyPair}\langle\rangle$$

This type has no free variables in it, and therefore it can be transmitted across address spaces. However there is then the problem that the resulting object is useless at the receiving site. Because the principals in the policy are locally abstract, there is no way to relate them to principals at the receiving site, and therefore no way to allow principals at the receiving site to access the private parts of the object according to the policy. Because the owner of the policy and the principals allowed read access by the policy, are completely abstract, no principal can ever access the key.

We need a way of hiding some of the principal identities in an object type, transmitting that object to another address space, then safely re-exposing the principal identities at the receiving site. To make this exposure safe, it must be based on checking the identity of authentication keys that identify principals. The following class accomplishes this:

```
class DistKeyPair ⟨prin Owner,
                   prinSet Objects,
                   prin Reader ∈ Objects,
```

```
                    spolicy K [Owner:Objects]⟩ {
    final pubkey⟨Reader⟩ rdrKey;
    pubkey⟨K⟩ publicKey;
    KeyPair (pubkey⟨Reader⟩ rdr,
             pubkey⟨K⟩ pub,
             {K}privkey⟨K⟩ priv) {
        this.rdrKey = rdr;
        this.publicKey = pub;
        this.privateKey = priv;
    }
}
```

The generator of the key pair creates an object of this type and elides its type to `DistKeyPair⟨⟩`. This key is then transmitted to another process. The receiving process uses the `rdrKey` field to justify exposing the identity of the reader principal name. If the `rdrKey` field is equal to `ThisPrin`, then the local principal name `Reader` can be exposed to `ThisPrin` at the receiving site. Although the set of allowed readers `Objects` is still opaque, the set constraint `Objects`$\supseteq\{$`Reader`$\}$ justifies allowing the receiving process to access the key. This checked exposure requires a new construct in Jeddak, the `keyassert` statement that asserts an equality between keys to justify an equality between policy names. If the key assertion succeeds, then the executing process is allowed to access the private key in the received object, using the reflected equality between key names and the inclusion constraint in the object type. This is summarized in Fig. 1.

3 Type System

We now provide a more detailed description of the type system. The syntax of types and kinds is provided in Fig. 2. The system of arities or kinds organizes the well-formed types, principal names and policy names. Besides the kind of types (Type) and principal names (Prin), there are two kinds for policy names: encryption (secrecy) policy names $\mathsf{SPolicy}(P:\overline{P})$ and signing (integrity) policy names $\mathsf{IPolicy}(P:\overline{P})$. This kind identifies the creator of the policy (P) and the principals who may have access to the private part of the public-private key pair for that policy ($\{\overline{P}\}$).

The types then consist of key types, indexed by policy names, classes, and opaque types resulting from accessing type fields in objects. The latter type has the form $x.t$ or $a.t$ where x is a program variable and a is an object identity. This restriction on the form of such opaque types only permits a limited dependency of types on values, maintaining the phase distinction between values and types. A class type has the form $C\langle\overline{t} = \overline{OT}\rangle$ where each OT is either null $\top$ or a type definition T. The former denotes a type in the instantiation of a class that has been made opaque, the latter a type whose instantiation is revealed in the object type.

Kinds and types depend on each other, resulting in a dependent kind system. The

```
            // Server side:
            Object req = reqCh.readObject();
            PubIKey kx = (PubIKey)req;

            spolicy [ThisPrin:kx.Owner] Kshare;
            pubkey⟨Kshare⟩ ksPub;
            privkey⟨Kshare⟩{Kshare} ksPriv;

            DistKeyPair reply =
              new DistKeyPair⟨...,Reader=kx.K⟩
                (kx.publicKey,ksPub,ksPriv);
            replyCh.writeObject(reply);
        // Client side:
        PubIKey req = new PubIKey ⟨...,K=ThisPrin⟩ (thisPrin);
        reqCh.writeObject(req);

        Object reply = replyCh.readObject();
        DistKeyPair ky = (DistKeyPair)reply;
        keyassert ky.rdrKey==thisPrin;
        // Now we know:  ky.Reader=ThisPrin
        // So we know:  ThisPrin∈ky.Objects
        ...  ky.privateKey ...
```

Figure 1: Enforcing Access Restrictions Across Address Spaces

$A \in$ Arity, Kind	::=	Prin	Principal
	|	SPolicy$(P : \overline{P})$	Secrecy policy name
	|	IPolicy$(P : \overline{P})$	Integrity policy name
	|	Type	Type
$K, P, T \in$ Policy, Prin, Type	::=	k, p, t	Variable, name
	|	$x.t$, $a.t$	Type field
	|	PrivKey(K)	Private key type
	|	PubKey(K)	Public key type
	|	$C\langle \overline{t} = \overline{OT} \rangle$	Class
$OT \in$ Optional Type	::=	$T \mid \top$	
$L \in$ Label	::=	$\{K_1, \ldots, K_m\}$	
$LT \in$ Labelled type	::=	$[T]^{L_1, L_2}$	

Figure 2: Syntax of Kinds and Types

formation rules for kinds and types must therefore be formulated in a mutually recursive fashion, and these formation rules are intertwined with the rules for environment formation in the type system. An environment is a sequence of pairs, binding (variables or names) to (kinds or types). We have several forms of environments:

$$\begin{array}{rcl} TE \in \text{Type Env} & ::= & \varepsilon \mid (t : A) \mid TE_1, TE_2 \\ VE \in \text{Value Env} & ::= & \varepsilon \mid (x : LT) \mid (a : LT) \mid VE_1, VE_2 \\ CE \in \text{Class Env} & ::= & \emptyset \mid \{C \mapsto Cl\} \mid CE_1 \cup CE_2 \\ EE \in \text{Constraint Env} & ::= & \emptyset \mid \{K_1 = K_2\} \mid EE_1 \cup EE_2 \end{array}$$

The sequence concatenation operation (,) is assumed to be associative with ε (empty sequence) as its unit. The type environment *TE* binds type, principal and policy names to their kinds. The value environment *VE* binds variables, keys and object identifiers to their types. The class environment maps from a class name to the definition of that class. Since this environment is invariant throughout both the static and dynamic semantics, we leave it implicit in the semantic rules. Finally the constraint environment *EE* records hypothetical equalities between policy names, based on runtime key equality checks. This latter environment is only necessary because of the keyassert construct.

The type environment *TE* is used in the definition of a metafunction that, given a label (set of policy names), computes the set of principals that is defined by that label

in that type environment:

$$\begin{array}{rcl} PRINS_{TE;VE}(\emptyset) & = & \emptyset \\ PRINS_{TE;VE}(L_1 \cup L_2) & = & PRINS_{TE;VE}(L_1) \cap PRINS_{TE;VE}(L_2) \\ PRINS_{TE;VE}(\{k\}) & = & \{\overline{P}\} \\ & & \text{if } \exists TE_1, TE_2.\ TE = (TE_1, k : \mathsf{SPolicy}(P : \overline{P}), TE_2) \end{array}$$

An equivalent definition can be given for the set of principals allowed "write" access according to integrity labels.

The formation rules for environments and kinds are omitted for lack of space. The formation rules for (labelled) types are also omitted. These rules check the obvious well-formedness conditions, e.g., that in an encryption key $\mathsf{PubKey}(K)$, the argument K denotes a policy name with an encryption policy name kind. These rules also check label constraints that are placed on public and private key types by the kinds of the corresponding key names.

The syntax of values, expressions and processes is provided in Fig. 3. The type system for expressions in general uses judgements of the form

$$TE; VE; EE \vdash e \in^P [T]^{L_1, L_2}$$

to check that the expression (code) e is well-formed with annotated type $[T]^{L_1,L_2}$ (secrecy label L_1 and integrity label L_2), under the assumption that it will be evaluated (executed) under the authority of the principal P, in the corresponding type, value and constraint environments TE, VE and EE, respectively.

Aliasing of type fields in objects, reflected by type sharing constraints in object interfaces, is a fundamentally important part of our mechanism for sharing keys, since policy names are locally opaque in objects. Consider for example:

```
PubKey⟨K = ⊤⟩ x;      // k is local opaque key name
PubKey⟨K = x.K⟩ x;    // by (Val Obj Self)
PubKey⟨K = x.K⟩ y = x;
pubkey⟨x.K⟩ z = e ? x.publicKey : y.publicKey;
```

Allowing either x or y to be assigned to z is allowed because of the type sharing constraint in the types of z and y.

Because of this treatment of type sharing, parameterized types are handled differently from GJ [4]. Whereas the latter allows class extensions of the form:

$$\mathsf{class}\ C_1\langle \overline{t} \rangle\ \mathsf{extends}\ C_2\langle \overline{T} \rangle\ \{\ldots\},$$

we restrict the forms of such extensions to

$$\mathsf{class}\ C_1\langle \overline{t_2}, \overline{t_1} : \overline{A_2}, \overline{A_1} \rangle\ \mathsf{extends}\ C_2\langle \overline{t_2} = \overline{t_2} \rangle\ \{\ldots\}.$$

In other words, the extending class can only add type parameters to the extended class, and the first $\overline{t_2}$ type parameters are aliases for the type parameters to the extended class.

This is necessary because of our treatment of type equality, and particularly the (VAL OBJ SELF) rule for allowing an object to refer to its own identity in the definition of its type exports. To see why the restriction on class extensions is necessary, consider the following example:

$\mathsf{class}\ C_1\langle t_1 : \mathsf{Type}, t_2 : \mathsf{Type}\rangle\ \mathsf{extends}\ C_2\langle t_1 = t_2, t_2 = t_1\rangle\ \{\ldots\}$
$C_1\langle t_1 = T_1, t_2 = T_2\rangle\ x;$
$C_1\langle t_1 = x.t_1, t_2 = x.t_2\rangle\ x;$ // by (Val Obj Self)
$C_2\langle t_1 = x.t_2, t_2 = x.t_1\rangle\ x;$ // by (Val Subsumption)
$C_2\langle t_1 = x.t_1, t_2 = x.t_2\rangle\ x;$ // by (Val Obj Self)

From this it is necessary to conclude that $x.t_1 = x.t_2$ and $x.t_2 = x.t_1$, which in general will not be true.

The type rules for expressions are provided in Fig. 4. In the (VAL ACCESS) and (VAL INVOKE) rules for field access and method invocation, respectively, the definitions of the type parameters $\overline{t}$ are relativized to the object itself ($\overline{x.t}$ if the object is referenced through the variable x, for example). This gives the field or method the most general type possible, with other equivalent types derivable using type equality and the type definitions $\overline{t} = \overline{T}$ exported in the object interface.

The (VAL ACCESS), (VAL INVOKE) and (VAL KEY ASSERT) rules all involve accessing a value, so any result from the ensuing computation carries with it the restrictions on the label for the value. This is the point of the merge that is done on the labelled type of the result, incorporating the restrictions from the value's labels into the result.

The (VAL OBJ SELF) rule allows any object's type to be redefined to relativize the identity of its type fields, some of which may be opaque, to the reference to the object itself. For example the t field of an object referenced by variable x can be relativized to have the definition $x.t$, reflecting the identity of the type that is relative to x.

The (VAL SUBSUMPTION) rule in Fig. 4 allows any value to have its type to be subsumed to some other type. This includes strengthening the secrecy restrictions and weakening the integrity restrictions in the outermost labels, as well as various forms of type equality and subtyping. There are two sources of type equality hypotheses. One is equations in the constraint environment *EE*, added by keyassert. The other source of type equality hypotheses is in type definitions exported in object interfaces.

4 Conclusions

The motivation for this work has been the need for proper programming abstractions for developing applications that must manage some or all of the task of securing their communication in a network environment. Abadi [1] considers a type system for ensuring that secrecy is preserved in security protocols implemented in that type system. Gordon and Jeffrey [11, 12] have developed a type-based approach to verifying authentication protocols. Abadi and Blanchet [2, 3] pursue an approach to analyzing

security protocols, initially for secrecy properties but later generalizing it to integrity properties. All of these works are focused on verifying secrecy and integrity properties of security protocols. As such the type systems that they use are far more sophisticated than the average programmer will use, while at the same time they give very strong guarantees of secrecy and integrity. The focus of our work is not protocol verification, but building accountable systems: engineering a system where accesses can be logged, but doing it in such a way that the performance of the system is not killed by the demands of credentials checking. So for example we make no attempt to cope with replay attacks.

Other work on security in programming languages has focused on ensuring safety properties of untrusted code [22, 21, 18] and preventing unwanted security flows in programs [8, 19, 28, 24]. Sabelfeld and Myers [25] provide an excellent overview of work in language-based information-flow security. Pottier and Conchon [24] have developed an interesting approach to encoding information flows in the lambda-calculus, allowing non-interference to be checked in an operational manner, and Pottier [23] has extended this to the pi-calculus. That approach can be applied to the object calculus introduced in the current article. In the presence of declassification, it is a well-known problem to define what safety guarantees are provided by information flow control [25, 29].

We have not had space to consider declassification in this account, particularly the notion of distributed declassification introduced by KDLM. It appears that the notion of *robust declassification* can be adapted to the framework of KDLM; we intend to report on this in a subsequent paper. A fuller report on Jeddak, including applications and an implementation, are also a topic of ongoing research.

The current article borrows heavily from work on type systems for module languages [14, 13, 15, 16, 26, 9]. In effect objects in our object calculus are first-class modules, with interface inclusion based on the inheritance hierarchy. However module languages do not normally consider re-exposing a type field that has been made opaque, as we do with keyassert. Also of course module languages do not normally enforce access control and information flow control. The semantics of module languages are normally functional; our object-calculus based semantics appears to be novel.

References

[1] Martin Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Science*, pages 611–638, 1997.

[2] Martin Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 33–44, 2002.

[3] Bruno Blanchet. From secrecy to authenticity in security protocols. In *9th International Static Analysis Symposium (SAS'02)*, pages 242–259, 2002.

[4] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the Java programming language. In Craig Chambers, editor, *Proceedings of ACM Symposium on Object-Oriented Programming: Systems, Languages and Applications*, pages 183–200, Vancouver, British Columbia, 1998. ACM Press.

[5] Robert Cartwright and Guy Steele. Compatible genericity with run-time types for the Java programming language. In Craig Chambers, editor, *Proceedings of ACM Symposium on Object-Oriented Programming: Systems, Languages and Applications*, pages 201–215, Vancouver, British Columbia, 1998. ACM Press.

[6] Tom Chothia, Dominic Duggan, and Jan Vitek. Type-based distributed access control. In *Computer Security Foundations Workshop*, Asilomar, California, June 2003. IEEE.

[7] Tom Chothia, Dominic Duggan, and Jan Vitek. Principals, policies and keys in a secure distributed programming language. Technical report, Stevens Institute of Technology, 2004.

[8] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 1977.

[9] Dominic Duggan. Type-safe dynamic linking with recursive DLLs and shared libraries. *ACM Transactions on Programming Languages and Systems*, November 2002.

[10] Dominic Duggan. Type-based cryptographic operations. *Journal of Computer Security*, 2003.

[11] Andrew D. Gordon and Alan Jeffrey. Authenticity by typing for security protocols. In *IEEE Computer Security Foundations Workshop (CSFW)*, June 2001.

[12] Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *IEEE Computer Security Foundations Workshop (CSFW)*, June 2002.

[13] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 123–137, Portland, Oregon, January 1994. ACM Press.

[14] Robert Harper, John Mitchell, and Eugenio Moggi. Higher-order modules and the phase distinction. In Paul Hudak, editor, *Proceedings of ACM Symposium on*

Principles of Programming Languages, pages 341–354. Association for Computing Machinery, 1990.

[15] Xavier Leroy. Manifest types, modules, and separate compilation. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 109–122, Portland, Oregon, January 1994. acmp.

[16] Xavier Leroy. Applicative functors and fully transparent higher-order modules. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 154–163, San Francisco, California, January 1995. ACM Press.

[17] John Mitchell and Gordon Plotkin. Abstract types have existential types. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, 1988.

[18] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, 1999.

[19] A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *IEEE Symposium on Security and Privacy*, 1998.

[20] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4), 2000.

[21] George Necula. Proof-carrying code. In *ACM Symposium on Principles of Programming Languages*, pages 106–119, 1997.

[22] George Necula and Peter Lee. Safe kernel extensions without run-time checking. In *Operating Systems Design and Implementation*, 1996.

[23] Francois Pottier. A simple view of type-secure information flow in the pi-calculus. In *Computer Security Foundations Workshop*, 2002.

[24] Francois Pottier and Sylvain Conchon. Information flow inference for free. In *Proceedings of ACM International Conference on Functional Programming*, 2000.

[25] Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 2002.

[26] Zhong Shao. Transparent modules with fully syntactic signatures. In *Proceedings of ACM International Conference on Functional Programming*, Paris, France, September 1999.

[27] Jose H. Solorzano and Suad Alagic. Parametric polymorphism for Java: A reflective solution. In Craig Chambers, editor, *Proceedings of ACM Symposium on Object-Oriented Programming: Systems, Languages and Applications*, Vancouver, British Columbia, 1998.

[28] D. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development*. Springer-Verlag, 1997.

[29] Steve Zdancewic and Andrew C. Myers. Robust declassification. In *Computer Security Foundations Workshop*, 2001.

$e \in \text{Expression}$	$::=$	w, x, y, z	Variable
	$\mid$	a, b, c, n	Name(Object id, Key)
	$\mid$	$\mathsf{newKey}\langle k : A\rangle(a^+ : LT_1, a^- : LT_2)\{e\}$	New policy name
	$\mid$	$\mathsf{new}\ [C\langle\overline{t = T}\rangle]^{L_1,L_2}(\overline{e})$	Create object
	$\mid$	$v.x$	Variable access
	$\mid$	$v.x(\overline{e})$	Method invocation
	$\mid$	$(LT\ x = e_1;\ e_2)$	Local variable
	$\mid$	$\mathsf{upcast}_{L_1,L_2} e$	Upcast on labels
	$\mid$	$(\mathsf{keyassert}\ e_1 == e_2\ \mathsf{in}\ e;\ \mathsf{else}\ e')$	Assert key equality
	$\mid$	v	Value
$v \in \text{Value}$	$::=$	$x \mid \langle\langle -59\langle a\rangle\rangle - 59\rangle_{P,()}^{L_1,L_2}$	
$K \in \text{Constructor}$	$::=$	$C(\overline{LT_2\ x_2}, \overline{LT_1\ x_1})\{\mathsf{super}(\overline{x_2});\ \overline{\mathsf{this}.x_1 = x_1};\}$	
$M \in \text{Method}$	$::=$	$LT\ f(\overline{LT\ x})\{\mathsf{return}\ e;\}$	
$Cl \in \text{Class}$	$::=$	$\mathsf{class}\ C\langle\overline{t_2}, \overline{t_1} : \overline{A_2}, \overline{A_1}\rangle\ \mathsf{extends}\ C_2\langle\overline{t_2 = t_2}\rangle\ \{\overline{LT\ x};\ \overline{M}\ K\}$	

Figure 3: Syntax of Expressions and Classes

Figure 4: Formation Rules for Expressions

Session VII

Invited Talk

Weak Secrets and Computational Soundness

Martín Abadi
Computer Science Department
University of California, Santa Cruz
abadi@cs.ucsc.edu

Passwords and other weak secrets sometimes serve as cryptographic keys in security protocols and elsewhere (e.g., [3, 9, 10, 12]). The use of weak secrets is particularly delicate because of the possibility of off-line guessing attacks. (In such an attack, data that depends on a weak secret may help in checking many guesses of the value of the weak secret.)

Addressing this difficulty, much recent research explores the design and analysis of systems that rely on weak secrets. Some of the methods developed are based on computational approaches to cryptography where the attacks are characterized in terms of their run-time complexity and probability of success (e.g., [2, 7]). Other methods are symbolic; they include abstract, formal accounts of off-line attacks and often benefit from tool support (e.g., [4–6, 8, 11]). This talk will report on some recent progress in this direction.

In parallel, another line of recent research aims to establish relations between computational and symbolic views of cryptography (e.g., [1, 13]). Intuitively, the symbolic approaches are typically based on the useful fiction that strong cryptographic keys are essentially unguessable; as long as the keys are strong (and under some non-trivial hypotheses) this fiction can be justified on computational grounds. This talk will discuss how to justify the symbolic treatment of weak secrets.

The talk is partly based on published work with Phil Rogaway and on current work with Bruno Blanchet, Cédric Fournet, and Bogdan Warinschi.

References

[1] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in*

Cryptology – EUROCRYPT ' 2000, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

[3] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 72–84, 1992.

[4] R. Corin, J. M. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. Technical report TR-CTIT-03-52, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, December 2003.

[5] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks (extended abstract). In R. Gorrieri and R. Lucchi, editors, *IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, Warsaw, Poland, April 2003. Dipartamento di Scienze dell'Informazione Universita di Bologna, Italy.

[6] Stéphanie Delaune and Florent Jacquemard. A theory of dictionary attacks and its complexity. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW 2004)*, 2004. To appear.

[7] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *Advances in Cryptology – EUROCRYPT ' 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 524–543. Springer-Verlag, 2003.

[8] Li Gong. Verifiable-text attacks in cryptographic protocols. In *INFOCOM '90*, pages 686–693, 1990.

[9] Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[10] J. Kohl and C. Neuman. RFC 1510: The Kerberos network authentication service (V5). Web page at `ftp://ftp.isi.edu/in-notes/rfc1510.txt`, September 1993.

[11] Gavin Lowe. Analysing protocol subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

[12] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer, editors, *VLDB 2003: Proceedings of 29th International Conference on Very Large Data Bases*, pages 898–909. Morgan Kaufmann Publishers, 2003.

[13] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems (extended abstract). *Electronic Notes in Theoretical Computer Science*, 32, April 2000.

Session VIII

Workshop on Logical Foundations of an Adaptive Security Infrastructure

Preface to WOLFASI

The Workshop on Logical Foundations of an Adaptive Security Infrastructure was intended to address basic logical questions (formalization, specification, verification) deriving from futuristic scenarios such as the following:

A distributed computer system, for which information security is a high priority, operates in a semi-autonomous mode. During a period of critical operation, the system detects an intrusion attempt in some nodes, along with a power glitch at other nodes, and a report about increase in a certain type of threat. This information is collected, analyzed, and various responses are executed: dealing with the perceived intrusion, rerouting network traffic around suspect nodes, adjusting the power allocation, adjusting the cryptographic strength of certain message authentication functions, etc. This set of executed responses was chosen to best achieve the desired result, within the confines of the security policy, as currently re-evaluated, at the appropriate time.

It was felt that the field of adaptive security is sufficiently well defined, important, and of current interest to warrant a special session of its own in the framework of FCS.

I want to thank Philip Scott (LICS Workshops Chair) and Mika Hirvensalo (ICALP Workshops Chair) for their help. I especially want to thank Andrei Sabelfeld (FCS Workshop Chair) for his very generous and gracious assistance. Finally, I want to thank the members of the WOLFASI Program Committee, listed below, for their interest and hard work.

Leo Marcus
WOLFASI Program Chair

WOLFASI Program Committee

John Baldwin, Chicago
Elisa Bertino, Milan
David Chess, IBM
Grit Denker, SRI
David Evans, Virginia
Wei Fan, IBM
Elena Ferrari, Insubria
Christopher Geib, Honeywell
Joe Halpern, Cornell
Sushil Jajodia, George Mason
Alan Jeffrey, De Paul
Angelos Keromytis, Columbia
Wenke Lee, Georgia Tech
Janos Makowsky, Technion
Tal Malkin, Columbia
Fabio Massacci, Trento
John McLean, NRL
Stephan Merz, INRIA Lorraine
Jonathan Millen, SRI
Carlo Montangero, Pisa
Alan Mycroft, Cambridge
Dusko Pavlovic, Kestrel
Paolo Perlasca, Milan
S. Raj Rajagopalan, Telecordia
Peter Reiher, UCLA
Michel de Rougemont, LRI
Vitaly Shmatikov, SRI
Alexander Shnitko, Novosibirsk
Luca Viganò, ETH
Ron Watro, BBN
Duminda Wijesekera, George Mason

Introduction to Logical Foundations of an Adaptive Security Infrastructure*

Leo Marcus
The Aerospace Corporation
Los Angeles
marcus@aero.org

Abstract

We give an introduction to questions relating to the logical underpinnings of an adaptive security infrastructure.

1 Introduction

The goals of this paper are to introduce the Adaptive Security Infrastructure concept, discuss issues of assurance and logical formalization, and state some tentative definitions and theorems.

The term "adaptive security" is intended to indicate that security policies and mechanisms can change in some automated or semi-automated fashion in response to events. Of course, adaptation is a matter of degree; all security architectures and devices are adaptive to *some* degree.

The need (or "use"; of course, as in many such technological "advances", sometimes it is a case of "invention is the mother of necessity" instead of the other way round) for (more adaptive) adaptive security stems from two considerations: short term and long term:

1. standard "static" security architectures do not cope well with rapidly changing security environments, including physical parameters, threats, attacks, policies, and mission goals.

2. At the other end of the spectrum, systems designed for extended many-decade life cannot predict and handle all future threats and attacks by *ab initio* built-in non-flexible mechanisms.

*This paper is intended as an introduction to WOLFASI, the Workshop on Logical Foundations of an Adaptive Security Infrastructure, held in conjunction with FCS'04, LICS'04, and ICALP'04, July 12-13, 2004, in Turku, Finland. As such, it is hereby declared to be exempt from some traditional requirements of professional papers, for example, to present actual results.

Appropriate adaptive architectures and mechanisms should be chosen according to which aspects of the short-term or long-term need are being addressed.

The term "infrastructure" was added on to "adaptive security", obtaining Adaptive Security Infrastructure (ASI), in order to indicate the approach that sees adaptive security as an integral, fundamental, functional component underlying any system, rather than an ill- (or nil-) structured collection of security devices.

While this need is become increasingly recognized – one could even say that over the last few years there has been a paradigm shift toward adaptivity – systems are still being specified, designed, and built without a good method for architecting system-wide adaptive security mechanisms.

Much work is currently being focused on detailed aspects of the related fields of intrusion detection, sensor networks, architectures, and security policies. Much less work is devoted towards putting together those pieces[1]. In particular, there does not appear to be a currently accepted good method for gaining confidence that the mechanisms to be employed will work together to deliver what, and only what, is needed. The hard part is "only" to decide what is wrong (security-wise) with the current state of affairs, what to do about it, and how to do that, with the resources available. Without a system-wide perspective, mechanisms can interfere with each other, be counterproductive, and create new vulnerabilities. Indeed, without the assurance that comes from rigorous specification leading to an enhanced likelihood of real verification, the cure may be worse than the disease.

Perhaps reflecting the author's personal bias, the first step toward true assurance requires some formalization of an ASI that *could*, *eventually* lead to the verification that proposed adaptive security mechanisms will perform as hoped (specified).

Enough about the need for adaptive security and formalization. In any case, we hope to show that there are some interesting logical questions relating to ASIs that have not really been addressed until now[2]. It is a hope of this workshop to help remedy that.

2 Components of an ASI

In order to be able to satisfy the stated goals, i.e., to coordinate detection of security-relevant input, security policy, user input, analysis, and then be able to formulate and execute a response, if needed, a natural approach is to isolate the three conceptual components of sensor, analysis, and response.

Taking this approach to the extreme, one can imagine a system which is constantly monitoring, analyzing, and responding, in order to maintain security invariants or to

[1] Over the past few years I have accumulated approximately 600 research papers on relevant topics. The bibliography section lists some of them.

[2] There are certainly connections with the related fields of "collaborative enterprises", "self-healing systems", "auto-adaptive systems", "reconfigurable systems," and the like.

evolve the system to satisfy new security properties, taking into account current security policy, severity of environmental effects, temporal and geographic aspects of attacks and responses[3].

The skeptical reader may be wondering how we can hope to prove anything about such a complicated system, when we can barely prove the most rudimentary security properties of the most rudimentary devices and mechanisms[4]? The answer is hierarchy! In other words, *assuming* the building blocks (protocols, algorithms, devices, interfaces) work as advertised, how do they function together? What properties need to be defined in order to even formulate theorems? What properties must components and interfaces have in order that their cooperative effect satisfies some desired property?

3 Formalization: Principles and Issues

What kind of "formalization" are we interested in? Some vague basic principles:

1. Use a mathematical logical framework
2. Abstract from realistic scenarios
3. Don't be concerned with usability or current technology (of course, at a deeper level, we recognize that current technology has an undeniable, if unmeasurable, influence on our imagination)
4. Long term goal should be a common, uniform, inter-interpretable semantics to allow rigorous specifications and verifications of architectures, properties, and capabilities that can connect policy, detection, analysis, and response.

The basic assumption:

- ASI exists in a temporal and spatial world. If we accept the temporal and distributed nature of the whole system in its full generality we get arbitrary architectural structures (patterns of connectivity, e.g. generalized networks) existing within the system and the ASI, and these structures may be dynamically changing. Any aspect of policy, specification, detection, analysis, or response can be considered in a version relativized to any definable structure. We call this the Pervasive Hierarchy Assumption (PHA).

The following research issues may appear to be rather grandiose in scope. Of course, they are, but part of the fun is to break them up into smaller bite-size, or at least meal-size, chunks.

[3] It is tempting at this point to jump straight to modeling biological defenses, immune systems, etc. But we prefer a more logical framework.

[4] If the skeptical reader would really say that, he or she is obviously not aware of some very good "historical" work in proving security properties of "evaluated products," or more recently security properties of protocols. Nevertheless, the point that we are far away from proving properties of an ASI is correct.

1. What are the appropriate semantics of a dynamic, adaptive security policy, and how should that be specified?
2. How should we take into account the global-local nature of all components of an ASI according to the PHA?
3. How should we specify the ”security-relevant resources” available so that at any time the analyzer can choose an appropriate response?
4. How do we specify the capabilities of responses (including trade-offs?)
5. How should we unify the temporal-spatial reasoning aspects?
6. What are the decidability or complexity issues in such a system?
7. What is the role of ”approximate security”?

3.1 Research Issues: Spatial

1. Specification of hierarchical architectures
2. Central (local) and distributed (global) detection, analysis, and response coordination
3. Smooth transition between hierarchies
4. Testability of policy satisfaction
5. Enforceability of response

3.2 Research Issues: Temporal

1. Duration of response
2. Synchronization
3. Relative speeds of changing environment, detection, analysis, communication, response
4. Incorporation of time in policy
5. Acknowledgments, success reports

4 Adaptive Security Policy

The goal for specifying adaptive security is twofold: to provide an umbrella guide for deciding if future events, actions, or responses are to be permitted under current policy; and to allow new security goals to be stated, in order to initiate system responses to enforce that policy, if necessary.

For example, we want to be able to reason about policy change within the context of larger policy or policy hierarchy[5]. We want to be able to test, prove, and implement security policies. We also want to be able to analyze combinations of security policies, for example, if the union of two security contains a contradiction.

We have used the term "security policy" without definition until now, which is dangerous since it might mean a lot of different things to different people, or to the same person at different times (as in the case of the author.) But what we mean here and now can be stated intuitively as follows:

- a security policy is (a specification of) what is allowed.

More precisely, in purely semantic terms, a security policy is a set of computer systems, namely those computer systems that satisfy that policy. Thus, if a computer system is identified with a set of computation sequences (the set of its permitted computation sequences), then a security policy is a family of sets of computation sequences. It is hard to get more general than that[6]. The general definition can be refined a bit by defining a *primitive* security policy to be a set of computations (so , e.g. "non-interference" or "non-deducibility" are not primitive), and an *enforceable* security policy to be a primitive policy that can be monitored.

Exactly which of these security policies are "static" and which are adaptive (or dynamic, if you prefer), is not a question with an objective answer.

However, as an example of a simple adaptive policy consider the following:

- System initially satisfies policy P_1
- At the first occurrence of condition C, system switches to policy P_2.

So this immediately raises the issue: what does satisfying a policy P in an interval (from one time/event t_1 to another time/event t_2) mean?

Answer?: *non-contradicting* the policy, i.e., that there is some continuation of the computation, or in the case of non-primitive policies, some enlargement of the set of computations (within some larger context of admissible computations), that explicitly satisfies the policy.

[5] As a simple example, a structured policy hierarchy can specify the system policy with regard to the security/performance tradeoff. At certain times confidentiality may be more important, and at others, availability.

[6] This generality appears to dilute totally the use of the word "security" in ASI. Security is certainly the motivation, and source of examples, but there may not be a technical (logical) reason to limit these considerations to the conventional security concerns of confidentiality, integrity, availability, etc.

If we represent the above situation by $\langle P_1; C \rightarrow P_2 \rangle$ then we can easily generalize the notation to, for example:

1. $\langle P; C_1 \rightarrow P_1, C_2 \rightarrow P_2, \ldots, C_n \rightarrow P_n \rangle$ Branching Policies
2. $\langle P; C_1 \rightarrow \langle P_1; C_2 \rightarrow P_2 \rangle \rangle$ Compound Policies

with the obvious intended meanings.

4.1 Incremental Policy

An incremental policy change is when we know what aspect we want to change, but don't know or don't care about the rest of the policy as expressed in its complete system-wide specification. For example, changing one user's access rights could/ should be expressible as an increment affecting only that user. This raises the question of dependencies among policies that may appear to be local: perhaps the change to one user's access rights, via some admissible interaction with other users, changes those other users' rights as well.

An increment can be a "weakening" (allowing more computations) represented by set union of the previous policy with the new policy, or a "strengthening" (allowing fewer computations) represented by set intersection of the previous policy with the new policy.

A policy increment can be indicated by: $\langle P; C \rightarrow (+P_1 - P_2) \rangle$, where P_1, P_2 are themselves policies, meaning: strengthen by P_1 and then weaken by P_2. Such an increment could be complex combination of strengthenings and weakenings.

4.2 Local Policy

Let H be a hierarchy description, A an ASI specification (as opposed to an individual instantiation), and P a policy.

Intuitively, we want

- *P is local with respect to H in A*

to mean something like

- the satisfaction of P in A is dependent only on the satisfaction of some (perhaps other, "test") policy in all subsystems satisfying H.

In certain situations we may want to define locality differently, by playing with the quantifiers and saying

1. "*For all* instantiations of A *there is* a test policy for P such that ..." or
2. "*There is* a test policy for P such that *for all* instantiations of A ..." or
3. "... in *some* subsystems satisfying H"

5 Specification, Derivation, and Verification of Response

One of the more challenging questions is how to specify and reason about responses, their relation to resources, and their capabilities. As examples, in current 2004 technology, some kinds of (defensive) responses that would be appropriate for certain security-relevant tasks include, in random order:

1. allocate resources (e.g. power; turning devices on or off)
2. adjust routing (including or excluding nodes)
3. change access rights
4. change crypto algorithms, keys, protocols
5. change sensor networks
6. change auditing
7. change strength of authentication
8. adjust intrusion detection system settings (altering the false positive/negative ratio)
9. install patches
10. destroy data or devices
11. install new hardware or software

In the general formal context of an ASI we can define a "response" to be simply a distributed program/algorithm running concurrently with the ongoing ASI and system operation. Of course, intuitively, common responses have more specific properties, like changing the state and terminating.

In order to incorporate responses into a formal framework, we need to

1. Specify and evaluate responsive resources
   - including communication channels, if needed
   - and including current (and projected) strength and location
2. Coordinate response with analysis
3. Plan appropriate action in time and space; consider temporary and local "fixes" while long-term global solution-response is being worked on

6 Detection and Analysis

The detection and analysis components are very closely related.

Typical detection data and mechanisms currently employed include:

1. intrusion detection methods of various kinds (e.g. signature and anomaly)
2. network statistics
3. system usage statistics
4. insider threat statistics
5. electronic background data

Who knows what other kinds of environmental information may be useful in the future? In coordinating this information lessons from the field of sensor networks are very relevant here. Obviously, the possible connection between the nature of data collected, the nature of the policy implemented, and the nature of the analysis engine, and how these connections themselves can be made adaptive, is a wide open question.

7 Other Topics

Other issues that could easily be relevant to the formalization of an ASI are

1. Approximate security, that is:
   - How to specify *achievable* security goals
   - Allow statistical properties in security policies
2. Game-theoretic view, that is:
   - Consider adaptive security to be a game between the environment and the ASI
   - The goal is to (assume minimal restriction on the environment and) design the ASI so the adversary (environment) does not have a winning strategy

8 Future Theorem

A typical theorem to be proved in some distant future verification of an ASI could look like:

Theorem:

1. For any system *S* implementing the specification S

2. for any ASI *A* implementing the specification A
3. for any adaptive security policy *P* of type P
4. for any environment *E* satisfying conditions E:

S + *A* satisfies *P* in *E*.

The ASI architect's problem: Given E, P, and S, find A, as above. As E gets more "realistic", P has to get weaker in order for there to be any hope of finding an appropriate A. This weakening can be in the temporal axis (allow for longer "lapse" of security) or the approximation axis (allow for less rigorous security conditions.)

9 Bibliography

Here we present a brief, definitely not comprehensive, list of documents that may provide the interested reader with some good starting points, before beginning his or her own directed internet search.

1. N. Aguirre, T. Maibaum, A temporal logic approach to the specification of reconfigurable component-based systems, Proceedings of the 17th International Conference on Automated Software Engineering ASE 2002, Edinburgh, UK, September 2002.

2. M. Aksit, Z. Choukair, Dynamic, adaptive, and reconfigurable systems: overview and prospective vision, DARES - The International Workshop on Distributed Auto-Adaptive and Reconfigurable Systems, in 23rd International Conference on Distributed Computing Systems Workshops (ICDCS 2003 Workshops), 19-22 May 2003, Providence, RI, USA.

3. X. Ao, N. Minsky, T. Nguyen, A hierarchical policy specification language, and enforcement mechanism, for governing digital enterprises, Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)

4. I. Aron, S. Gupta, Analytical comparison of local and end-to-end error recovery in reactive routing protocols for mobile ad hoc networks, Proceedings of the 3rd ACM international Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Boston, Massachusetts, 2000, pp 69 - 76

5. A. Arora, S. Kulkarni, Detectors and correctors: a theory of fault-tolerance components, 18th International Conference on Distributed Computing Systems, pp: 436–443, IEEE Computer Society, Amsterdam, The Netherlands, 1998

6. L. Bauer, J. Ligatti, D. Walker, More enforceable security policies. In Foundations of Computer Security, Copenhagen, Denmark, July 2002.

7. L. Bauer, J. Ligatti, D. Walker, A calculus for composing security policies Technical Report TR-655-02, Princeton University, 2002.
http://citeseer.ist.psu.edu/bauer02calculus.html

8. M. Becker, L. Gilham, D. Smith, Planware II: Synthesis of schedulers for complex resource systems, Kestrel Institute, submitted for publication, 2003

9. A. Belokosztolszki, Ken Moody and David M. Eyers, A formal model for hierarchical policy contexts, Policy 2004: IEEE 5th International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, U.S.A. June 2004.

10. R. Bharadwaj, Secure middleware for situation-aware naval C2 and combat systems, 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), 28-30 May 2003, San Juan, Puerto Rico

11. C. Bidan, V. Issarny, Dealing with multi-policy security in large open distributed systems, Proceedings of 5th European Symposium on Research in Computer Security, pages 51–66, September 1998

12. J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. Surendran, D. Martin, Automatic management of network security policy DARPA Information Survivability Conference and Exposition (DISCEX II). Volume II. Pages 12-26. Anaheim, CA. June 2001. Pub. IEEE Computer Society Press, Los Alamitos, California. ISBN: 0769512127

13. Michael Carney and Brian Loe, A comparison of methods for implementing adaptive security policies (carney.pdf) Proceedings of the 7th USENIX Security Symposium San Antonio, Texas, January 26-29, 1998

14. A. Cass, B. Lerner, E. McCall, L. Osterweil, A. Wise, Logically central, physically distributed control in a process runtime environment, Technical Report 99-65, University of Massachusetts at Amherst, Nov. 1999.

15. B. Charron-Bost, C. Delporte-Gallet, H. Fauconnier, Local and temporal predicates in distributed systems, ACM Transactions on Programming Languages and Systems (TOPLAS) Volume 17 , Issue 1 (January 1995), pp: 157 - 179

16. Craig M. Chase, Vijay K. Garg, Detection of global predicates: techniques and their limitations, Workshop on Distributed Algorithms, France, September 1995

17. S. Cheng, D. Garlan, B. Schmerl, J. Sousa, B. Spitznagel, P. Steenkiste, Using architectural style as a basis for system self-repair, Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance, 2002, pp: 45-59

18. D. Chess, C. Palmer, S. White, Security in an autonomic computing environment, IBM Systems Journal, Vol 42, No. 1, 2003 107-118

19. K. Cheverst, C. Efstratiou, N. Davies, A. Friday, Architectural ideas for the support of adaptive context-aware applications, Workshop on Infrastructure for Smart Devices - How to Make Ubiquity an Actuality HUC 2k, Bristol September 27, 2000

20. J. Cobleigh, L. Osterweil, A. Wise, B. Lerner, Containment units: a hierarchically composable architecture for adaptive systems, ACM SIGSOFT 2002, Charleston, SC, November 2002

21. R. Corin, A. Durante, S. Etalle, P. Hartel, A trace logic for local security properties, Int. Workshop on Software Verification and Validation (SVV), Mumbai, India, Dec. 2003

22. J. Doyle, I. Kohane, W. Long, H. Shrobe, P. Szolovits, Event recognition beyond signature and anomaly, In Proceedings of the Second IEEE SMC Information Assurance Workshop. IEEE, IEEE Computer Society, June 2001

23. C. Efstratiou, A. Friday, N. Davies and K. Cheverst, Utilising the event calculus for policy driven adaptation in mobile systems, Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), Monterey, Ca., U.S., J. Lobo, B. J. Michael and N. Duray (eds.), IEEE Computer Society, pp. 13-24, June, 2002

24. D. Evans, A. Twyman, Flexible policy-directed code safety, in IEEE Symposium on Security and Privacy, May 1999

25. R. Feiertag, S. Rho, L. Benzinger, S. Wu, T. Redmond, K. Levitt, D. Peticolas, M. Heckman, Intrusion detection inter-component adaptive negotiation Proceedings of the RAID 99: Recent Advances in Intrusion Detection, West Lafayette, Indiana, USA, September 7-9, 1999

26. R. Filman, T. Linden, SafeBots: a paradigm for software security controls, 1996 ACM New Security Paradigms Workshop, Lake Arrowhead

27. M. Fitzi, U. Maurer, From partial consistency to global broadcast, In Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC), pages 494-503, 2000

28. D. Gabelaia, R. Kontchakov, A. Kurucz, F. Wolter, M. Zakharyaschev, On the computational complexity of spatio-temporal logics, To appear in the proceedings of FLAIRS, 2003

29. V. Garg, J. Mitchell, Distributed predicate detection in a faulty environment, Proc. IEEE International Conference on Distributed Computing Systems, Amsterdam, Netherlands, 1998

30. D. Garlan, B. Schmerl, J. Chang, Using gauges for architecture-based monitoring and adaptation, Working Conference on Complex and Dynamic Systems Architecture, December 2001, Brisbane, Australia

31. C. Geib, R. Goldman, Information modeling for intrusion report aggregation, Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX-II), 2001

32. C. Gill, D. Levine, Quality of service management for real-time embedded information systems, Technical Report, Center for Distributed Object Computing, Dept. of Computer Science, Washington University, St. Louis, MO, 2000

33. C. Gunter, T. Jim, Design of an application-level security infrastructure, DIMACS Workshop on Design and Formal Verification of Security Protocols, September, 1997

34. J. Guttman, Filtering postures: local enforcement for global policies, 1997 IEEE Symposium on Security and Privacy

35. U. Halfmann, W. Kuhnhauser, Embedding security policies into a distributed computing environment, Operating Systems Review, Vol. 33, No. 2 (April 1999)

36. B. Hashii, S. Malabarba, R. Pandey, M. Bishop, Supporting reconfigurable security policies for mobile programs Computer Networks 33 (2000), 77-93

37. D. Hollingworth, T. Redmond, Enhancing operating system resistance to information warfare, MILCOM 2000. 21st Century Military Communications Conference Proceedings

38. T. Jensen, D. Le Metayer, T. Thorn, Verification of control flow based security policies, IEEE Symposium on Security and Privacy, pp 89-103, 1999

39. S. Jiang, R. Kumar, Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications, IEEE Transactions on Automatic Control, 2001, submitted

40. Anita Jones, Cyber security in open systems, in Computer Systems: Theory, Technology and Applications edited by A. Herbert and K. S. Jones, Springer-Verlag, December 2003

41. Z. Kalbarczyk, S. Bagchi, K. Whisnant, R. Iyer, Chameleon: a software infrastructure for adaptive fault tolerance, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, NO. 6, June 1999

42. John Keeney, Vinny Cahill, Chisel: A policy-driven, context-aware, dynamic adaptation framework, Policy 2003, Como Italy, June 2003

43. A. Keromytis, J. Parekh, P. Gross, G. Kaiser, V. Misra, J. Nieh, D. Rubenstein, S. Stolfo, A holistic approach to secure survivability, ACM Workshop on Survivable and Self-Regenerative Systems (SRS), held in conjunction with the 10th ACM International Conference on Computer and Communications Security (CCS). October 2003, Fairfax, VA.

44. F. Kerschbaum, E. Spafford, D. Zamboni, Using embedded sensors for detecting network attacks, Journal of Computer Security Volume 10, Issue 1-2, 2002, pp: 23 - 70

45. J. Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbu, M. Gertz, The willow survivability architecture, Proceedings of the 4th Information Survivability Workshop, 2001

46. C. Ko, P. Brutch, J. Rowe, G. Tsafnat, K. Levitt, System health and intrusion monitoring (SHIM) using a hierarchy of constraints, Proceedings of 4th International Symposium, RAID, 2001

47. J. Kong, H. Luo, K. Xu, D. Gu, M. Gerla, S. Lu, Adaptive security for multilevel ad hoc networks, Wireless Communications and Mobile Computing, Special Issue on Mobile Ad Hoc Networking, 2002

48. C. Krishna, I. Koren, A. Ganz, C. Moritz, Security tradeoffs in NEST, DARPA presentation, Dec. 2003

49. O. Kupferman, M. Vardi, Synthesizing distributed systems, Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science, 2001

50. W. Lee, J. B. D. Cabrera, A. Thomas, N. Balwalli, Y. Zhang, Performance adaptation in real-time intrusion detection, RAID 2003

51. W. Lee, W. Fan, M. Miller, S. Stolfo, E. Zadok, Toward cost-sensitive modeling for intrusion detection and response, Journal of Computer Security Volume 10, Issue 1-2 2002

52. R. de Lemos, J. Fiadeiro, An architectural support for self-adaptive software for treating faults, Workshop on Self-healing Systems, November 2002

53. Y. Liu, S. Smith, A component security infrastructure Foundations of Computer Security Workshop (FCS'02), Copenhagen, Denmark, July 2002

54. J. Lobo, R. Bhatia, S. Naqvi, A policy description language, in Proc. of AAAI, Orlando, FL, July 1999

55. O. Maler, Control from computer science, IFAC Annual Reviews in Control, 2003

56. D. Malkhi, M. K. Reiter, An architecture for survivable coordination in large distributed systems, Knowledge and Data Engineering, vol 12(2), 2000

57. L. Marcus, Semantics of static, adaptive, and incremental security policies, First Symposium on Requirements Engineering for Information Security (SREIS) March 2001, Indianapolis, Technical Report ATM 2001(8104-05)-1, The Aerospace Corporation July 2001

58. L. Marcus, Local and global requirements in an adaptive security infrastructure, International Workshop on Requirements for High Assurance Systems (RHAS 2003), Sept. 2003

59. S. Merz, M. Wirsing, J. Zappe, A spatio-temporal logic for the specification and refinement of mobile systems, Fundamental Approaches to Software Engineering (FASE 2003), April 2003, Warsaw, Poland

60. J. Millen, Local reconfiguration policies, Proceedings of the 1999 IEEE Symposium on Security and Privacy

61. N. Mittal, V. Garg, On detecting global predicates in distributed computations, Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS), pp 3-10, April 2001

62. P. Nathan, A trajectory for the evolution of security infrastructure management systems (SIMS) architecture, Symbiot, Inc., Austin, TX, Dec. 2003

63. P. Pal, R. Schantz, J. Loyall, Timeliness in auto-adaptibve distributed systems, Proceedings. 24th International Conference on Distributed Computing Systems Workshops, March 2004.

64. D. Pavlovic, Towards semantics of self-adaptive software, In P. Robertson, R. L., and Shrobe, H., eds., Self-Adaptive Software. Springer-Verlag. 2000, pp50-64

65. X. Qie, R. Pang, and L. Peterson, Defensive programming: using an annotation toolkit to build DOS-resistant software, ACM SIGOPS Operating Systems Review Volume 36, Issue SI Winter 2002

66. D. Ragsdale, C. Carver, J. Humphries, U. Pooch, Adaptation techniques for intrusion detection and intrusion response systems, www.itoc.usma.edu/ragsdale/pubs/adapt.pdf

67. R. Rivest, B. Lampson, SDSI – a simple distributed security infrastructure, USENIX 96 and CRYPTO 96, April 30, 1996

68. R. Roshandel, Coupling static and dynamic semantics in an architecture description language, Working Conference on Complex and Dynamic Systems Architecture, December 2001, Brisbane, Australia

69. T. Ryutov, C. Neuman, The specification and enforcement of advanced security policies, Proceedings of the Conference on Policies for Distributed Systems and Networks (POLICY 2002), June 5-7, 2002, Monterey, California

70. J. Salasin, The DARPA ISO DASADA project: Dynamic assembly for systems' adaptability, dependability, and assurance, http://www.rl.af.mil/tech/programs/dasada/program-overview.html

71. R. Sandhu, Decentralized management of security in distributed systems, Proceedings IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, October 15-16 1991, Santa Barbara, CA

72. F. Schepers, A framework for adaptive security management systems, http://tmf.studentenweb.org/papers/infosec/msc-thesis/html/msc-thesis.html

73. D. Schnackenberg, K. Djahandari, D. Sterne, Infrastructure for intrusion detection and response, Proceedings of the DARPA Information Survivability Conference and Exposition 2000.

74. F. Schneider, Enforceable security policies, Cornell University Technical Report TR98-1664. Jan 1998; also ACM Transactions on Information and System Security Vol 3, No 1, pp 30-50, February 2000

75. D. Scott, A. Beresford, A. Mycroft, Spatial security policies for mobile agents in a sentient computing environment, IEEE POLICY 2003 (4th International Workshop on Policies for Distributed Systems and Networks).

76. L. Semini, C. Montangero, A logic with modalities for asynchronous distributed systems, tp.di.unipi.it/pub/techreports/TR-99-23.ps.Z

77. D. Stewart, P. Khosla, Real-time scheduling of sensor-based control systems, Proceedings of the IFAC/IFIP Workshop, May 15-17, 1991. pp. 139-144.

78. O. Strichman, R. Goldring, Testing, monitoring, and controlling real-time systems with temporal invariants, http://iew3.technion.ac.il/Home/Users/SECOND.php?ofers+Ofer+Strichman+4+4

79. B. Tung, Common intrusion detection framework, 1999, http://www.isi.edu/gost/cidf/

80. P. Venables, Security monitoring in heterogeneous globally distributed environments, Information Security Technical Report, Volume 3, Issue 4, 1998, pp: 15-31

81. R. Venkatesan, S. Bhattacharya, Threat-adaptive security policy, IEEE Journal, 1997, pp: 525-531

82. L. Wagner, Byzantine agreements in secure communication, 5th Operations Research Conference, ASOR 2003

83. D. Wijesekera, S. Jajodia, Policy algebras for access control – the predicate case, Proceedings of the 9th ACM conference on computer and communications security Washington, DC, USA, 2002 pp: 171 - 180

84. D. Wile, Towards a synthesis of dynamic architecture event languages, WOSS '02 (Workshop on Self-healing Systems, November 2002)

85. T. Wu, M. Malkin, D. Boneh, Building intrusion tolerant applications Proceedings of USENIX Security Symposium, August 1999

86. Q. Zhang, R. Janakiraman, Indra: a distributed approach to network intrusion detection and prevention, Washington University Tech. Report # WUCS-01-30, 2001

87. B. Zhao, A. Joseph, J. Kubiatowicz, Locality aware mechanisms for large-scale networks, Proceedings of Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, June, 2002

Practical and Theoretical Issues on Adaptive Security

Alexander Shnitko

Novosibirsk State Technical University
Karl Marks Avenue 20,
Novosibirsk, Russia, 630092
shnitko@newmail.ru

Abstract

In recent years emphasis in providing formal information security has moved from tasks oriented towards construction of "ideal" security model to tasks oriented towards integral solutions of special security problems of complex information systems. One of the axioms for this approach became the statement that "uncertainty (and insecurity) exists, cannot be eliminated and must be mitigated" in such kinds of systems. One of the possible ways to settle this matter is by using the concept of adaptation applied appropriately. This paper is intended to highlight general problems and methods of their solutions concerning the adaptive security concept in complex information systems.

Keywords: complex information system, adaptive security, security modeling

"Water shapes its course according to the nature of the ground over which it flows; the soldier works out his victory in relation to the foe whom he is facing"
Sun Tzu. "On the art of war"

1 Introduction

A wide range of modern information systems works in open dynamic environments which impede usage of traditional formal modeling (e.g. HRU, Bell-LaPadulla models) to reason about their security properties. However an approach to separate solutions of specific security problems (e.g. virus detection, intrusion detection, certificate management, etc.) is also shown to be inconsistent in providing adequate protection over time [6, 25].

One of the possible approaches to the problem of providing integral security in such information systems is to develop a common framework based on continuous

adaptation and evolution depending on changing external and internal conditions. Despite a number of research projects being initiated in recent years to address related problems, there is still no consistent approach to the subject.

This paper is intended to outline a possible vision on the place of adaptation in the broad field of information security, to propose a formalization of such concepts, and to discuss some important issues related to the subject. The concepts provided in this work are results of formal research rather than practical modeling experience.

As related to the subject defined in the WOLFASI call [26] the main goal of the paper presented is to discuss general issues connected to the formal definition of Adaptive Security Infrastructure (ASI). Also it partially addresses topics from the WOLFASI call such as composite and hierarchical nature of components of ASI, temporal reasoning about its dynamic behavior, and complexity issues of such systems.

The paper is organized in the following matter. Section 2 briefly outlines common practical and theoretical issues of providing integral security in complex information systems which contribute to the adaptive security conception. Section 3 discusses general aspects of modeling of adaptive security systems with some motivating samples included to illustrate its correspondence with specific security problems. In Section 4 we highlight some important issues connected with implementation of such models in practice.

2 Technical problems of providing integral security

In practice it is unlikely to reach certain level of protection of complex information systems using only local solutions. Modern information security standards [13, 14] explicitly address this issue requiring integral set of methods to be applied in real-world security systems (integral security approach). For example as stated in [14] this must include administrative, procedural and technical means. And in [13] there are in particular such obligatory requirements for secure information systems as self-testing, fault-tolerance and active audit.

In order to illustrate current demand for the adaptive approaches to the information security, it is required to identify general security problems in modern complex information systems. This section is devoted to the description of such problems, primarily connected with technical issues both in theory and practice. Although the list of issues highlighted is not claimed to be thorough, it allows us to make some conclusions about some general tasks for which the adaptive security concept is proposed.

2.1 Practical issues

Dynamical nature of specific security threats There are a lot of open questions in the field of information security (e.g. formal verification, computational aspects and others); yet, they differ significantly in priorities when considering real world implementations.

Though there are a lot of threats to the information security, still for the last decade on the top of the list of practical problems remains protection from computer viruses and hacker attacks [24, 28]. The typical solution to these problems is as follows: to acquire specialized hardware and software tools for prevention of viruses and hacker threats, and then begin the process of continuous update and upgrade of these systems. Unfortunately this approach implies the explicit presence of unavoidable side effects in its behavior and the lack of formalized and proved conclusions about efficiency and security of such systems (mostly because of uncertainty in internal and external condition, including appearing of new kinds of viruses and attacks, unpredictable system loading, etc.).

Another source of information security threats is internal infrastructure. Though there are a lot of achievements both in practical implementation of formally secure and reliable information systems, the dynamic and open architectures are still significantly vulnerable to malicious users, configuration mistakes, etc.

Obstacles, problems and demands Today there is a lot of work in progress in research and development of intellectual or automatic security tools and solutions intended to address problems stated above [1, 8-10, 15, 25]. However their real-world implementations still have no wide-spread usage, and most importantly there is no generally recognized vision on the subject of providing dynamic security in complex information systems across researchers and implementers.

Therefore it is very topical to develop common foundations for such issues, and one of the possible approaches there is an assumption for future complex secure information systems to be based on some meta-model or formal infrastructure intended to link and coordinate all the dynamic components within the unified environment with common set of goals and restrictions.

However, today there are many obstacles to the practical implementations of this approach in real-world systems. Some of such obstacles which contribute significantly to the adaptive security conception are listed below:

- Explicit complication of the overall information infrastructure which is already very complex in installation and maintenance;
- Typical priority of tactical tasks in the field of information security as compared with strategical tasks, absence of common goals for overall information security;
- Insufficient knowledge about possibilities and limitations of available security tools and methods, disregard to synergetic aspects of the subject;
- Incomplete or inadequate information about threats and their reasons and in particular about real goals, methods and tools used by trespassers.

These obstacles in turn lead to the following typical problems:

- Inefficient and inadequate usage of available security methods and tools;
- Scattering of resources when trying to solve a lot of special security problems at the same time;
- Practically guarantied vulnerability of security system with explicit aftereffect.

Listed issues require efficient solutions intended to keep and evolve current achievements of the real-world information security which results in high demand for the adaptive security tools and methods.

2.2 Theoretical issues

Related work In recent years a body of research was initiated intended to develop solutions for the problem of providing dynamic security in complex information systems (e.g. [4, 15, 25]). One of the important questions in this field is the development and analysis of adaptive functions for the wide range of information security tasks. Therefore, the problem of formal modeling and correct implementation of such adaptive systems arises.

First of all, this task is explicitly cross-disciplinary in nature which leads to the following complications: (1) a coordination of heterogeneous fields of knowledge (i.e. definitions, axioms, assumptions, methods, etc.) is required, (2) implicit dependences on the achievements of the "base" theories is present. Secondly, the rapid development of the subject impedes the activities oriented towards the common systematization of the basic information security theoretical matters (e.g. definition of atomic information security element, security level, etc.).

Though a lot of work has been done in related directions a solid foundation for further research and development in the field of adaptive security is still required. Currently, several possible frameworks and models are being proposed for this purpose (e.g. [18, 22, 26]).

Our approach The approach to the formal task of providing adaptive information security described in this work implies usage of Control Theory and Dynamical Systems Theory notions for the description of specific security functions (e.g. control object and subject, feedback, iterative adaptive algorithms, etc.). It is neither an alternative access control model, nor agent-based model, but rather a general formalization of the miscellaneous adaptive security processes in complex information systems (e.g. adaptive encryption, adaptive intrusion detection, adaptive self-testing and self-configuration, etc.).

3 Adaptive security modeling

3.1 Framework for adaptive security

This work is based on the ideas similar to those described in [2, 18, 27]. It uses notion "framework" as a set of common definitions and assumptions about adaptive security modeling. Concepts and definitions used for formalization follow the works related to Control Theory such as [7, 19].

3.1.1 Background and definitions

Common adaptive model Complex Information System (CIS) is defined as a set of intercommunicating objects with following properties:

- Every object performs some type of information processing;
- Comprehensive mathematical model of the whole information system in unknown;
- Internal and external influences on the objects cannot be fully determinated.

CIS objects are characterized by a set of control parameters and by their internal states.

According to [18, 22, 26] a common model of adaptive security is defined as follows (Figure 1): CIS include Control Objects (CO) and Control Device (CD, operating as controlling object) which consists of Detector Device (DD, collecting data from Control Object and environment), Analyzer Device (AD, processing general CIS requirements and data from Detector Device) and Responder Device (RD, providing resulting control action).

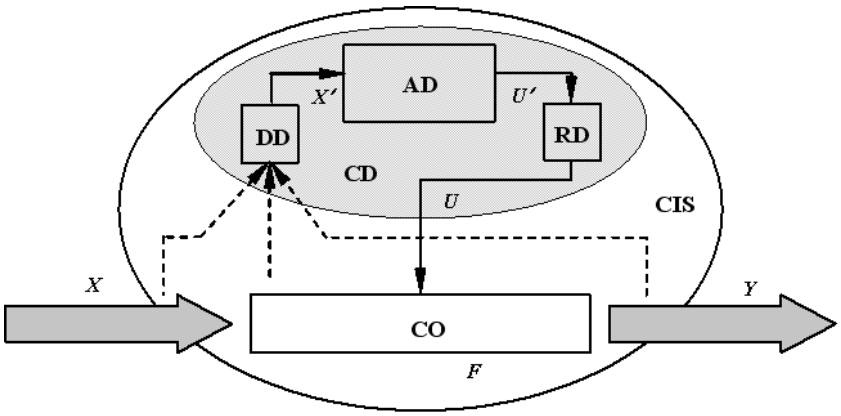


Figure 1: General model of adaptive CIS

In the general case the adaptive security model could include hierarchy of Control Devices and Control Objects with structured set of goals and restrictions. This means that specific CIS treated as adaptive model could appear as a Control Object itself at a higher level of decomposition.

Control Object in context of information security For the means of adaptive security framework we consider following decomposition of basic tools and methods used in information security technical processes (Figure 2):

- *Requirements to CIS security* – goals, rules, and restrictions which specify overall security limitations and originate from outside the system;
- *Theoretical methods of providing information security in CIS* – formal security models, cryptography, security protocols, etc. which operate with abstract definition of information;
- *Practical tools of providing information security in CIS* – firewalls, anti-virus systems, proxy servers, etc. which operate with practical definition of information.

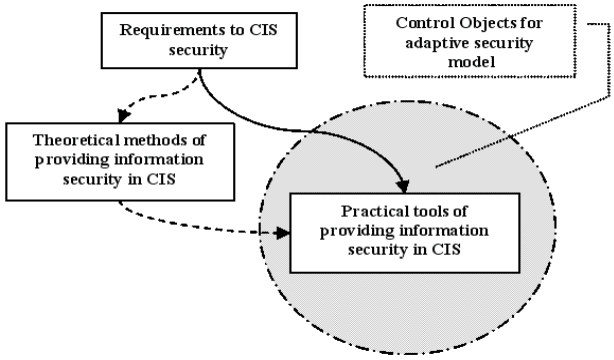


Figure 2: Control object for adaptive security model

Concerning this decomposition practical security tools are considered as Control Object in adaptive security model with control parameters defined respectively.

However the majority of security threats are caused by application components of CIS. Therefore in general case every individual CIS component should be considered as security-related and dynamically assigned specific level of importance concerning current security issues. Respectively any CIS component could be used in adaptive security processes.

Types of adaptive models Hence, adaptive security model could have large variety of properties two distinct types could be singled out as follows.

Optimal adaptive security model: has global goals with formal definition as calculated functions; full controllability; local self-organization is limited, global self-organization is extended; the model of the system is already identified; adaptation mostly preventive (stochastic, before environment influences); control is optimal.

Simplified adaptive security model: has only local goals expressed as constraints; partial controllability; local self-organization is extended, global self-organization is limited; the model of the system require continuous identification; adaptation mostly compensatory (algorithmic, after environment influences); redundant control.

Due to the practical and theoretical obstacles discussed in Paragraph 2 for the majority of present-day CIS only simplified model of adaptive security could be sufficiently implemented. However the optimal adaptive model should be considered as a desired state for further evolutions of such system.

Contribution to cyber-warfare models As stated in recently proposed cyber-warfare models (e.g. [1, 5, 11, 12, 20, 23]) the general task of providing information security for specific CIS implies the presence of defender and attacker. In the adaptive security framework the attacker could be interpreted as other CIS, its elements or common information environment as a whole. In order to contribute to the cyber-warfare models mentioned possible axiomatic premises about relations of the subjects in adaptive security framework in first approach could include as follows:

- Attacker and defender have some conflicting goals;
- Attacker and defender are limited in resources used to reach their goals;
- Attacker and defender in general case have compound structure;
- Attacker and defender have external and internal priorities concerning their goals, resources and structure;
- Attacker and defender have undetermined options in interactions.

3.2 Mathematical formalization of adaptive security

The process of adaptation is illustrated on Figure 1. According to [19] in general case a task of adaptation is considered as a problem of optimal control of specified object F. State S of the object and its influence on the environment Y depends on influences X of the environment and set of adaptable factors $U = (u_1, u_2, ..., u_r) : Y = S(X, U)$.

Goals Z of the adaptive control are defined by specific constraints on the state S of the object. In the general case such constraints can be represented as follows:

$$S(X,U) \in \begin{cases} H(U) = (h_1(U), h_2(U), ..., h_p(U)) \geq 0; \\ G(U) = (g_1(U), g_2(U), ..., g_s(U)) = 0; \\ Q(U) = (q_1(U), q_2(U), ..., q_l(U)) \to \min, \end{cases} \quad (*)$$

where

$$h_i(U) = M_x(h_i^*(Y)) = M_x(h_i^*(F(X,U))),\ i = 1, ..., p;$$
$$g_j(U) = M_x(g_j^*(Y)) = M_x(g_j^*(F(X,U))),\ j = 1, ..., s;$$
$$q_k(U) = M_x(q_k^*(Y)) = M_x(q_k^*(F(X,U))),\ k = 1, ..., l;$$

and $M_x(\cdot) = \int (\cdot)p(X)dX$– function for average-out by X, where $p(X)$ – frequency distribution of the states X of the environment.

The goal of adaptation is the solution of the problem $(*)$, and since from the complex systems perspective, there is not adequate information on the mathematical model of the object F and frequency distribution $p(X)$ of the states of the environment, it is necessary to develop and apply special adaptive algorithms. Such algorithms would be intended to solve the problem $(*)$ observing only values of the functions $h_i^*(\cdot),\ g_j^*(\cdot),\ q_k^*(\cdot)$ $(i = 1, ..., p;\ j = 1, ..., s;\ k = 1, ..., l)$ at the specific moments of time.

$$U_{N+1} = \varphi(\overrightarrow{U_{N,W}},\ \overrightarrow{H^*_{N,W}},\ \overrightarrow{G^*_{N,W}},\ \overrightarrow{Q^*_{N,W}}),$$

where φ – recurrent algorithm of the adaptation, W – size of memory of the algorithm, and $\overrightarrow{U_{N,W}}$, $\overrightarrow{H^*_{N,W}}$, $\overrightarrow{G^*_{N,W}}$, $\overrightarrow{Q^*_{N,W}}$ – adoptable parameters vector and vectors of the values of the criterion function measured from $N - W$ till Nmoments of time. Such algorithms could be developed both for discrete and continuous time.

An approach for more precise definition of the adaptation problem in the context of the information security using notion of security modes proposed as follows.

Let $M_1, M_2, ..., M_k$ be alternative security modes specifying security parameters for class $\Omega = \{T_j\}$ of the information tasks in CIS. Any mode M_i characterized by vector of security-related control parameters: $M_i = M_i(\vec{U})$. Let also $q = q(T, M)$ be a criterion function intended for the estimation of specific characteristics (e.g. efficiency, adequacy) of distinct security modes applied for specified information tasks $T_j \in \Omega : q_{i,j} = q(T_j, M_i)$.

The necessity of usage of adaptation arises when the specified criterion function can be calculated only a posteriori. Correspondingly the goal is to select an efficient security mode minimizing following integral criteria for the current flow of the information tasks using only observations of its values:

$$Q(M) = \int q(T, M)p(T)dT$$

Graphical interpretation of adaptive security processes

Let Z^* be admitted region of adaptable factors, Z– admitted region of system states, U – current value of adaptable factors, $\varphi(U)$ – adaptable function, X– current environment influence and $S(X,U)$ – current state of the system. In this notion common graphical interpretation of allowable adaptive security processes is illustrated on Figure 3 and Figure 4.

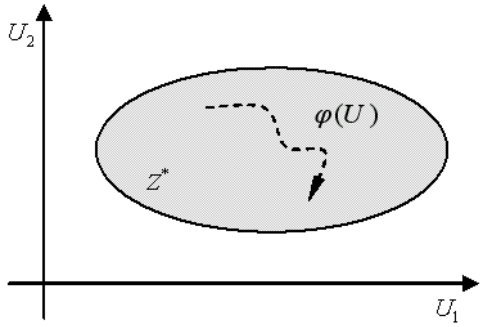


Figure 3: Process of adaptation in the adoptable factors space

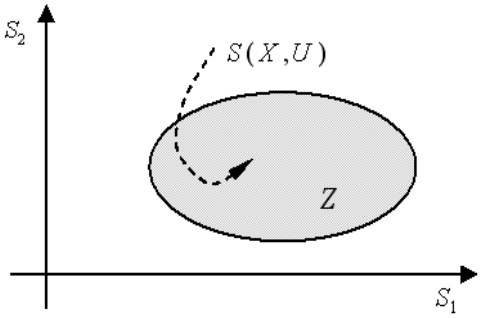


Figure 4: Process of adaptation in the system states space

3.3 Motivating samples of adaptive security modeling

The necessity of adaptation for specific information security functions has been recognized for a variety of tasks. Furthermore miscellaneous solutions are being proposed and implemented, e.g. for encryption adaptation in open architectures [21], for intrusion detection performance adaptation [16], for security policy adaptive transitions [3] and others. However these solutions primarily address the adaptation problem in connection to the specific security function, rather than considering it as a global property of complex information systems.

This paragraph includes two simplified examples intended to illustrate how the general adaptive model proposed above could be used for specific tasks of information security.

3.3.1 Adaptive self-scanning and searching vulnerabilities

Task description. CIS with high requirements on reliability should continuously perform self-scanning and searching vulnerabilities activities. However particular tests could require considerable time and computational power, decreasing availability of certain components of CIS and increasing risks of Denial of Service (DoS) attacks. The task of adaptation is to select optimal sequence of tests and components to be tested depending on specified priorities and current state of CIS.

General problems. (1) initial specification of test priorities and estimation of their complexity; (2) development of adaptive algorithm of selection of optimal sequence of tests at any period of time; (3) measurement of current components loading, computation of criterion function.

Task formalization. Let $CIS = \{O_1, O_2, ..., O_n\}$ be complex information system, where O_i – individual components. Let $T_1, T_2, ..., T_m$ be set of tests for self-scanning and searching vulnerabilities, $CP(T_j)$ – computational requirements of tests, $P(T_j)$ – specified priorities of tests, and P_{crit} – critical value for the test priority. Let $R(O_i)$ be value for maximum computational power of specific component and $W(O_i)$ – its current computational load. In general case $R(O_i)$ values should be periodically revised. It is required to determine optimal sequence of tests in order to minimize decrease of availability of CIS components.

The solution could be formulated as a task of searching the maximum of the following function:

$$\varphi(O_i, T_j) = \begin{cases} \frac{R(O_i) - W(O_i) - CP(T_j)}{CP(T_j)}, & when\ P(T_j) < P_{crit}; \\ 1, & when\ P(T_j) \geq P_{crit}. \end{cases}$$

at any given time, since an optimal sequence of tests could not be predetermined because of unknown dynamic load of CIS components.

Considering the significant number of CIS components and the complexity of $W(O_i)$ computations, optimized adaptive algorithms could be used for the solution of the problem (e.g. algorithms descibed in [7, 19]).

3.3.2 Adaptive security policies with intrusion detection

Task description. Components of CIS are controlled by several OS installed on different computers, the intrusion detection system is used on the security perimeter. In the event of an intrusion attempt, the detection sequence of security states transitions is developed depending on the current functional load and active security policy.

General problems. (1) initial estimation for the security level of different security policies (e.g. “basic”, “medium”, “strict”); (2) development of adaptive algorithm of changing security states (e.g. selection of optimal security policy and effective time of its usage).

Task formalization. Let $CIS = \{O_1, O_2, ..., O_n\}$ be complex information system, where O_i – individual components. Let $P_1, P_2, ..., P_m$ be different security policies and T – effective time of security policy usage. Let $EV = \{ev_1, ev_2, ..., ev_k\}$ be security events (e.g. “attack of type1”, “attack of type2”, “security policy expired”, etc.). It is required to determine adequate state of the system $S(t_i) = SE(ev(t_i), P(t_{i+1})) = \{P(t_i), T(P(t_i))\}$ at any moment of time t_i depending on security events and currently used security policy $P(t_{i-1})$

Considering adaptive nature of attacks on the CIS it is logical to apply adaptive algorithm for the selection of the security policy and time of its efficient usage correspondingly to current security situation. In particular case of simply ordered security policies the first control parameter could be adapted as follows:

$$P(t_{i+1}) = \begin{cases} Strengthen(P(t_i)), & when\ ev(t_i) \in \{\text{"attack registered"}\} \\ Weaken(P(t_i)), & when\ ev(t_i) \in \{\text{"time elapsed"}\} \end{cases}$$

where $Strengthen(P_j) = P_{j+1},\ Weaken(P_j) = P_{j-1}$ – operators of the security policy transitions.

The second control parameter could be adapted stochastically: $T(P(t_i)) = T_{\min} + \xi \cdot T_{\text{add}}$, where $T_{\min}$ – minimum acceptable time of new security policy usage (e.g. depending on the restricted response time of the system administrator, etc.), T_{add} – defined additional time of new security policy usage, ξ – uniformly distributed stochastic variable.

Such approach could be developed with specific adaptive algorithm chosen for particular tasks of prevention and decreasing risk of DoS attacks on CIS.

Though the samples described could be developed and used separately for the specific information security tasks, the most common purpose of the adaptive security model is to propose a common approach to the problem of adaptation of dynamic information security functions in the CIS.

4 Adaptive security implementation

Though the work presented is primarily research-oriented, the adaptive security concept itself is tightly linked with practice. This paragraph is intended to outline some basic questions connected to the implementation issues of the adaptive functions in information security systems which require further consideration. Additional information on the topic and some approaches to the stated problems could be found in related works (e.g. [3, 16, 21]).

4.1 Practical functions of adaptive security

Though the general idea of adaptive security promises a lot of advantages there is still a lack of reasonable implementations.

Concerning the model described the currently operating CIS could use the elements of adaptation to efficiently solve practical security tasks by implementing common adaptive infrastructure or by using specific methods and algorithms. Some possible purposes of adaptive security are listed below.

Functional tasks of adaptive security system:

- Internal audit and searching of potential vulnerabilities;
- Automatic maintenance and control for update and upgrade activities;
- Formal optimization of security functions;
- Preventive and reactive actions depending on the current state of system and environment.

General tasks of adaptive security system:

- Identification of the CIS model;
- Optimization of control functions.

4.2 Complexity of adaptive security

As stated above in a majority of present-day CIS adaptive security model could be implemented only in simplified version. The most common complications for the task are as follows:

- Complexity of correct definition of goals and restrictions on the security of CIS as a whole and/or its components;
- Problems of occasional or continuous identification of the object controlled (the model of CIS as a whole and/or its components);
- Task of development of adaptation algorithms with small response time (in order to provide adequate reaction on the external and internal influences).

These complications could be solved using following well-known approaches:

- Using initial redundancy in limitations on CIS and on security tools and methods used with following optimization of resource usage and information flows (which helps to simplify common formalization of goals and requirements for CIS, partially compensate lack of information about common model of CIS and disregard about synergetic aspects of components interactions);

- Processing only top-level analytical information, storing only selective historical data related to information security events (could be used for the task of identification, to accelerate decision-making process and to lower complexity of adaptive algorithms);

- Using stochastic and specially optimized adaptive algorithms designed for the search of local extremum (could be used to reduce response time, to compensate lack of information about common model of CIS, and also could emulate preventive effect by using stochastic behavior which reduces risk of abuse of adaptive system reactions by attacker).

Redundancy. The task of reliability control in CIS by using redundancy is well-known and has been thoroughly explored. This approach show itself from the good side in practical systems. Component duplicating, constraints of the kind "everything is restricted by default" could be examined in a certain way and defined for specific practical systems. However theoretical aspects about efficiency of such approach should be examined more precisely.

Analytical data. This task is more complicated and requires additional work in reasoning about importance of components of CIS and their adaptable factors. It apparently is not easily formalized in the general case, and requires significant theoretical research. The complexity of this task depends on application area and falls into stage of initial development of the adaptive model and stage of continuous improvement and adjustment of the model.

Special adaptive algorithms. This task has been thoroughly explored in Control Theory and number of efficient approaches to the problem had been developed. Supposing correct implementation of those results the efforts on this stage of the development of adaptive security model could be relatively small because of presence of optimal stochastic and gradient methods (e.g. [7, 19]), which could be efficiently used as the solution of the problem of local extremum search in the limited period of time.

4.3 Correctness of adaptive security

To solve the problem of adaptive security formal correctness verification in the specific CIS it is required to assess adequacy and realizability of the models and methods listed above.

One of the possible approaches to such assessment consists in statistical evaluating of the properties of chosen adaptive algorithms, including study of their deviation from specified goals of adaptation and their robustness.

This approach is being explored, but at the moment there is not enough information to make general conclusion and to formulate theoretical results on the usage of statistical methods in assessment of correctness of adaptive security systems.

5 Conclusions

Though it is very attractive to develop all-sufficient, reliable adaptive model of complex information security system, the practical and theoretical progress in this direction is still far from the desired goal. It also should be mentioned that adversaries are unconstrained with “the rules” of local theories and often use unexplored and unformalized power of human brain to reach their goals. Therefore today real security can not be built solely on the theoretical results or special solutions – the general control and management from the humans are still essential. Taking this into consideration the research and development activities in the field of adaptive security modeling could be basically regarded as an intention to create auxiliary but powerful tools for the creative process of providing integral security in real complex information systems.

Formal aspects of such process mentioned in the work presented require a continuous feedback from practical information security experts in order to improve and refine general approaches and models. Then, having basic conceptions defined, further research in the field of adaptive security could be primarily oriented on formal verification of such models and on issues connected to their efficient implementation.

6 Acknowledgements

I would like to thank Nikolay Dolozov for his patient and constant support of my research activities over time, Alexander Popov for constructive critics and discussions of the subject, and Leo Marcus for his initiative in organizing Workshop on Logical Foundations of an Adaptive Security Infrastructure (WOLFASI) this year which motivated me to define the vision of this research subject in more general way.

References

[1] Alexander D. S., Abraugh W., Keromytis A., Smith J., Security in Active Networks, In Secure Internet Programming, Lecture Notes in Computer Science, Springer-Verlag Inc., New York, NY, USA, 1999.

[2] Badrinath B., Fox A., Kleinrock L., Popek G., Reiher P., Satyanarayanan M., A Conceptual Framework for Network and Client Adaptation, Mobile Networks and Applications, v.5 n.4, p.221-231, December 2000.

[3] Carney M., Loe B., A Comparison of Methods for Implementing Adaptive Security Policies, In proceedings of the 7th USENIX Security Symposium, pp. 1-14, 1998.

[4] Cybersecurity today and tomorrow: pay now or pay later, Computer Science and Telecommunications Board, Division of Engineering and Physical Sciences, National Research Council. Washington, D.C.: National Academy Press, 2002.

[5] Denning D., Information Warfare and Security, Addison-Wesley, 1999.

[6] Denning D., The Limits of Formal Security Models, National Computer Systems Security Award Acceptance Speech, October 1999.

[7] Fradkov A., Adaptive Control in Complex Systems. Searchless Methods. (Moskow: Nauka), 1990. (In Russian)

[8] George S., Evans D., Marchette S., A Biological Programming Model for Self-Healing, First ACM Workshop on Survivable and Self-Regenerative Systems, October 2003.

[9] Getting Past The Cyberspace Hype. Adaptive Security. A Model Solution - A Solution Mode, Internet Security Systems, Inc., June 1997.

[10] Global Security Practice: Adaptive Security, Cap Gemini Ernst&Young. http://www.capgemini.com/technology/as/pdf/adaptivesecurity.pdf

[11] Gorodetski V., Kotenko I., Attacks against Computer Network: Formal Grammar-based Framework and Simulation Tool. Fifth International Symposium. RAID 2002. Zurich, Switzerland. October 2002. Proceedings. Lecture Notes in Computer Science, V.2516. Springer Verlag, 2002.

[12] Halpern J., O'Neill K., Secrecy in Multiagent Systems, In proceedings of the 15th IEEE Computer Security Foundations Workshop, pp. 32–46, June 2002.

[13] Information Technology - Security techniques - Evaluation criteria for IT security ISO/IEC 15408-1, 15408-2 and 15408-3, December 1999.

[14] ISO/IEC 17799 Information technology – Code of practices for information security management, First edition. ISO/IEC 17799:2000(E), December 2001.

[15] Kephart J., Chess D., The Vision of Autonomic Computing, Computer, IEEE Computer Society, January 2003.

[16] Lee W., Cabrera J., Thomas A., Balwalli N., Saluja S., Zhang Y., Performance Adaptation in Real-Time Intrusion Detection Systems, Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002), Zurich, Switzerland, October 2002.

[17] Li J., Yarvis M., Reiher P., Securing Distributed Adaptation, Computer Networks, Vol. 38, No. 3, January 2002.

[18] Marcus L., Local and Global Requirements in an Adaptive Security Infrastructure, In Proceedings of International Workshop on Requirements for High Assurance Systems, September 2003.

[19] Rastrigin L., Adaptation of Complex Systems (USSR, Riga: Izdatelstvo Zinatie), 1981. (In Russian)

[20] Rattray G., Strategic Warfare in Cyberspace, MIT Press, 2001.

[21] Reiher P., Eustice K., Sung K.-M., Adapting Encrypted Data Streams in Open Architectures, Proceedings of the Adaptive Middleware Services Workshop (AMSW) 2001, August 2001.

[22] Shnitko A., Adaptive security in complex information systems // KORUS-2003.The 7-th Korea-Russia International Symposium on Science and Technology: Proceedings / University of Ulsan, Republic of Korea. – Ulsan, 2003. Vol. 2. – P.206-210.

[23] Stytz M., Banks S., Requirements and Issues in Cyberwarfare Simulation. http://www.sisostds.org/doclib/doclib.cfm?SISO_RID_1001972

[24] The State of Information Security, 2003, CIO Magazine and PricewaterhouseCoopers, September 2003. http://www2.cio.com/research/surveyreport.cfm?id=64

[25] Trust in cyberspace / Fred B. Schneider, editor; Committee on Information Systems Trustworthiness, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications, National Research Council, 1999.

[26] Workshop on logical foundations of an adaptive security infrastructure, Turku, Finland, July 12-13 2004. http://www.aero.org/wolfasi

[27] Yarvis M., Reiher P., Popek G., Conductor: A Framework for Distributed Adaptation, In Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS VII), pp. 44-49, Rio Rico, Arizona, March 1999.

[28] Yearly Economic Damage Estimate - All Attacks - since 1995, SIPS Excerpt, MI2G Ltd., March 2004. http://www.mi2g.com/cgi/mi2g/press/020304graph.php

On Comparing the Expressing Power of Access Control Model Frameworks

E. BERTINO
Purdue University
bertino@cerias.purdue.edu

B. CATANIA
University of Genova
catania@disi.unige.it

E. FERRARI
University of Insubria at Como
elena.ferrari@uninsubria.it

P. PERLASCA
University of Milano
perlasca@dico.unimi.it

Abstract

One of the key requirements of an Adaptive Security Infrastructure (ASI) is the ability of formally reasoning about the specified security policies. In this paper, we contribute to this issue by focusing on one of the most relevant kind of security policies, i.e., access control policies. Several access control frameworks have been proposed so far in the literature, based on different formalisms. However, what has not yet so far extensively explored is the comparison of the expressive power of such frameworks. We believe that this is a key issue since such analysis can be the basis for choosing the framework that better fits the security needs of a given domain. This analysis is particular relevant for complex and distributed environments, like the ones in which the ASI paradigm may be usefully applied. The aim of this paper is thus to make a step in this direction by comparing the expressive power of three well known frameworks with respect to the set of access control models they are able to express.

Keywords: Access control framework, access control policies, access control model, analysis, logic programming

1 Introduction

Adaptive Security Infrastructures provide functionalities to collect information about the security environment, such as, for instance, security threats, and to analyze such data for proposing and executing specific compensating actions. Clearly, executed actions must agree with the security policies in place in the considered environment. Therefore, a key requirement in such a dynamic system is that of providing a framework for the representation, analysis, and usage of security policies. Security policies can be of many different types according to the different aspects they regulate in an

environment, and can be expressed by using different formalisms. Moreover, security policies can also be dynamic in that they may change according to the actions and the properties of the environment in which they are used. In this context, an important issue is the ability of formally representing the semantics of such policies in order to be able to formally answer specific questions regarding security. For instance, it is important to formally state which actions are in accordance with the specified policy, and which actions should be prevented because they violate the security policy. Additionally, in distributed environments, like the ones in which ASIs may be usefully applied, many advanced applications have articulated security requirements that cannot be adequately supported by a single-policy mechanism. Thus, what is required is a multi-policy system, that is, a system in which many different security policies can coexist. Such systems are very flexible in that they give the Security Administrator (SA) the possibility of selecting the policies the better fits the security requirements of the considered domain.

In this paper, we mainly focus on one of the most important class of security policies, i.e., access control policies. Access control policies are enforced by an access control mechanism, which is in charge of managing authorization rules and, based on these rules, regulating accesses by subjects to the protected objects. Different approaches can be adopted to realize a multi-policy access control mechanism. A first option is to use different access control models, each of which is able to represent a single policy. However, this solution has the drawback that different models must coexist within the same system and this can cause non trivial interoperability issues. An alternative promising solution is to base the system on a general access control framework allowing the specification and enforcement of different access control policies. Additionally, in such multi-policy scenarios, we believe that the availability of a framework able to represent in a uniform way heterogeneous access control policies is a useful instrument for the SA in performing security analysis, identifying strategies, and producing compensating actions. In particular, such a framework can be useful when a modification of the access control requirements determines an update of the enforced access control policies or when we need to represent in a uniform way the heterogeneity of the security policies and formalism used to express them. For instance, in order to response to a crisis involving different countries, such as military or humanitarian crisis, a fundamental key issue is the ability to collect and share information according to the different security policies enforced by each country. In this respect it is important to coordinate the access to the available information so that the different access control policies be not violated. Additionally, another relevant issue is the ability to dynamically change in a more restrictive way the specific access control policies.

Various general access control frameworks have been so far proposed [1, 6, 2]. Each framework provides a formalism, by which access control policies can be specified, and a semantics, by which authorizations can be computed. Different frameworks are usually based on different formalisms and different computational semantics, thus

supporting the representation of different sets of policies. An important point, which has not been extensively explored so far, is the comparison of the expressive power of the proposed frameworks. For database languages, the issue of expressive power has been widely explored and has resulted in several formal results. Most of them have been proposed in the context of deductive query languages and concern query containment and equivalence [7, 9]. On the other hand, no similar theory has been developed for access control languages. The development of such a theory is very relevant since it allows one to determine which is the most expressive framework and thus to select the framework that could become the core component of any multi-policy access control system. The aim of this paper is thus to give a contribution in this direction by comparing the expressive power of three well known frameworks.

The frameworks we consider for the analysis are the Jajodia et al. framework [6], the NIST framework [2], and the Bertino et al. framework [1], also called LAMP (LogicAl Multi-Policy) framework. We focus our comparison on these three frameworks because they are the most comprehensive among the ones proposed so far and have been shown to be able to model a variety of access control policies. The Jajodia et al. [6] framework is based on a logical formalism and has been conceived to express traditional discretionary and role-based access control models. It supports policy specification by means of stratified Datalog programs. Based on the operational semantics for stratified programs, each program generates a single set of authorizations. The NIST framework [2] is a general framework for modeling role-based access control (RBAC) policies. RBAC has been proved to be policy neutral, being able to express both discretionary and mandatory policies. Finally, the LAMP framework [1] is a logical framework designed with the aim of providing the representation of heterogeneous policies, either discretionary, mandatory, or role-based. Each policy is represented by means of a Datalog program with negation, and authorization bases are computed by using the stable model semantics of logic programs [4]. In particular, in this paper we formally prove than LAMP is more expressive than the other two frameworks.

The remainder of this paper is organized as follows. Sections 2, 3, and 4 briefly present the frameworks [1, 6, 2] we are going to compare, whereas the comparison, with respect to the expressive power is then proposed in Section 5. Finally, Section 6 presents some conclusions and outlines future work.

2 The LAMP (LogicAl MultiPolicy) framework

The LAMP framework [1] has been developed with the goal of being as general as possible. It is based on a set of primitive “building blocks” upon which all other necessary concepts can be constructed: based on the access control model to be represented, only the building blocks needed to represent the model are selected and composed together. LAMP can express discretionary, mandatory, and role-based policies. It supports both

positive and negative, direct and propagated authorizations,[1] as well as the possibility of expressing user-defined constraints over the set of access authorizations.

The framework supports the representation of four basic components - subjects, objects, privileges, and sessions - and the representation of arbitrary authorization rules. Additionally, the framework supports the specification of constraints on basic components. Each basic component is characterized by a variable number of *attributes* which model properties that are relevant in the specification of access control policies.

LAMP relies on a logical formalism, based on C-Datalog [5], an object-oriented extension of Datalog. C-Datalog supports both classical object-oriented concepts, such as classes, objects, inheritance, used to model basic components, as well as typical logic-based concepts, such as deductive rules, used to represent authorization and constraint rules.

A key feature of LAMP is the distinction between the notion of *Access Control Model Schema* (ACMS for short) and *Access Control Model Instance* (ACMI for short). Informally, an ACMS defines the structural components over which the model is based, whereas an ACMI provides information concerning the component instances, that is, the actual subjects, objects, privileges, sessions, and the authorization and constraint rules used to instantiate the model. The components supported by LAMP are the following:

- *Domain component (DC)*. Domain classes represent the structure of basic components of the LAMP framework (subjects, objects, privileges, and sessions), whereas domain instances, i.e., sets of facts, represent the actual subjects, objects, privileges, and sessions.
- *Domain structure component (DSC)*. Domain structure information represents relationships existing among the basic components.
- *Authorization component (AC)*. Authorization component contains authorization rules.
- *Propagation component (PC)*. Propagation component consists of rules, called *propagation rules*, by which additional authorizations can be derived, starting from authorization rules and domain information.
- *Constraint component (CC)*. The constraint component is composed of derived relation rules able to express static and dynamic constraints on the basic components.

LAMP provides a set of predicates for expressing authorization rules and constraints. Among them ErrorC is used to define constraint rules, Auth_d is used to define direct authorizations, Auth_p is used to define propagated authorizations, and Auth to refer indiscriminately to direct and propagated authorizations, when no distinctions are

[1] A positive authorization establishes that a subject is authorized to exercise a given privilege on a given object, whereas a negative authorization establishes that a subject is denied to access a given object under a given privilege. Direct authorizations represent authorizations explicitly specified by the Security Administrator (SA), whereas propagated authorizations represent authorizations that are generated due to the activation of propagation rules.

needed.

An *Access Control Model Instance* is a C-Datalog Program constructed over an ACMS, satisfying specific conditions. This program contains definitions (sets of facts and rules) for all the components presented above. The following example presents an ACMS and ACMI according to which an access control model is modeled into LAMP.

$\textbf{subject}(self : subject, name : string)$ $\quad$ $\textbf{user}(self : user)$

$\textbf{role}(self : role)$ $\quad$ $\textbf{object}(self : object, name : string)$

$\textbf{privilege}(self : privilege, name : string)$ $\quad$ $\textbf{session}(self : session, name : string)$

$Scheme_1^*(\textbf{user}) = (self : user, name : string)$ $\quad$ $Scheme_1^*(\textbf{role}) = (self : role, name : string)$

$user(self : \#1, name : Dan)$ $\quad$ $object(self : \#4, name : o1)$

$privilege(self : \#7, name : R)$

$user(self : \#2, name : Bob)$ $\quad$ $object(self : \#5, name : o2)$

$privilege(self : \#8, name : X)$

$user(self : \#3, name : Jill)$ $\quad$ $object(self : \#6, name : o3)$

$privilege(self : \#9, name : W)$

$role(self : \#10, name : r1)$ $\quad$ $session(self : \#13, name : sDan)$

$Play(U : \#1(Dan), R : \#10(r1))$

$role(self : \#11, name : r2)$ $\quad$ $session(self : \#14, name : sBob)$

$Play(U : \#2(Bob), R : \#11(r2))$

$role(self : \#12, name : r3)$ $\quad$ $session(self : \#15, name : sJill)$

$Play(U : \#3(Jill), R : \#12(r3))$

$subject(self : \#1, name : Dan)$ $\quad$ $subject(self : \#2, name : Bob)$

$subject(self : \#3, name : Jill)$

$subject(self : \#10, name : r1)$ $\quad$ $subject(self : \#11, name : r2)$

$subject(self : \#12, name : r3)$

$LessR(R_1 : \#11(r2), R_2 : \#10(r1))$ $\quad$ $LessR(R_1 : \#12(r3), R_2 : \#10(r1))$

$ActiveRole(U : \#1(Dan), S : \#13(sDan), R : \#10(r1))$

$ActiveRole(U : \#2(Bob), S : \#14(sBob), R : \#11(r2))$

$ActiveRole(U : \#3(Jill), S : \#15(sJill), R : \#12(r3))$

$Auth_d(O : \#4(o1), S : \#10(r1), P : \#7(R), G : \#SA, \epsilon : +)$

$Auth_d(O : \#4(o1), S : \#10(r1), P : \#9(W), G : \#SA, \epsilon : +)$

$Auth_d(O : \#5(o2), S : \#11(r2), P : \#7(R), G : \#SA, \epsilon : +)$

$Auth_d(O : \#6(o3), S : \#12(r3), P : \#9(W), G : \#SA, \epsilon : +)$

1 : $InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$

2 : $InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$

3 : $UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$

4 : $UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$

5 : $Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7) \leftarrow Auth_p(O : X_1, S : X_8, P : X_3, G : X_4, \epsilon : +, O' : X_5, S' : X_6, P' : X_7), InLessR(R_1 : X_8, R_2 : X_2)$

6 : $Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8) \leftarrow Auth_p(O : X_1, S : X_9, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8), UserPlay(U : X_2, R : X_9), ActiveRole(U : X_2, S : X_{10}, R : X_9)$

7 : $Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$

8 : $Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$

Figure 1: Example of LAMP usage

Example 1 *Suppose you want to model a role-based access control model into LAMP. Assume to consider three users (Dan, Bob, and $Jill$), three objects ($o1$, $o2$, and $o3$), three privileges (read (R), execute (X), and write (W)), three roles ($r1$, $r2$, and $r3$),*

and three sessions (sDan, sBob, and sJill), one for each user, such that: i) the role hierarchy is captured by the following partial ordered relationships: $r2 < r1$ *and* $r3 < r1$*; ii) authorizations involving lower roles in hierarchy are propagated upwards along the hierarchy; iii)* $r1$*,* $r2$*, and* $r3$ *are the roles that* Dan*,* Bob*, and* $Jill$ *can respectively play and have activated; in their own sessions; iv) role* $r1$ *is authorized to exercise privileges* R *and* W *on* $o1$*; v) role* $r2$ *is authorized to exercise privilege* R *on* $o2$*, whereas role* $r3$ *is authorized to exercise privilege* W *on* $o3$*.*

Figure 1 presents the ACMI that models the scenario described above. The meaning of the rules presented in Figure 1 is the following: i) the rules 1 *and* 2 *capture the direct and indirect relationships in the role hierarchy; ii) the rules* 3 *and* 4 *specify the roles that a user is directly (rule* 3*) or indirectly (rule* 4*) authorized to play, due to the role hierarchy; iii) rule* 5 *propagates an authorization for a given role to roles that are at an upper level in the hierarchy, whereas rule* 6 *propagates authorizations from each role to users that are authorized to play that role and have activated it; iv) finally, the rules* 7 *and* 8 *are used to distinguish between direct and propagated authorizations supported by LAMP framework: rule* 7 *allows authorizations that are not derived through propagation to be used through propagation, whereas rule* 8 *allow one to refer indiscriminately to the whole set of (direct and propagated) authorizations.* □

An important issue in LAMP, as well as in other framework, concerns the management of conflicts arising because of the possible presence of both a negative and a positive authorization for the same subject, object, and privilege. Conflicts require suitable conflict resolution policies to determine whether an access should be authorized or not. In this respect, LAMP provides a parametric conflict resolution policy that, for each conflict, specifies how the conflict has to be solved, by choosing whether the positive or the negative authorization should prevail. The exact conflict resolution policy depends on the access control model being modeled. Moreover, since LAMP does not impose any restriction on the type of negation that can be used, the semantics which is used is an extension of classical stable model semantics supporting a parametric conflict resolution policy. According to such semantics, each ACMI is associated with a set of (consistent) stable models, each representing a set of authorizations that can be possibly assigned to subjects according to the specified rules. The proposed semantics does not make any assumption on the set of authorizations to be selected, that may depend on the access control model being represented and can be chosen by the Security Administrator (SA).

3 Jajodia et al. framework

In the Jajodia et al. framework [6] access control models are represented by stratified logic programs [8], constructed over a given logical language. This language supports the specification of positive and negative authorizations, authorization propagation, conflict resolution, and constraints on authorizations. Each logic program represents a

Atom	Meaning
cando(o,s,$\overline{a}$)	represents a potential positive or negative (depending on the sign of $\overline{a} \in SA$) authorization for subject $s \in AS$ on object $o \in AO$ explicitly inserted by the SA
dercando(o,s,$\overline{a}$)	represents derived potential positive or negative authorization $\overline{a} \in SA$ for subject $s \in AS$ on object $o \in AO$
do(o,s,$\overline{a}$)	represents allowed or denied accesses of type $a \in A$ for subject $s \in AS$ on object $o \in AO$
done(o,u,r,a,n)	represents an allowed access at time $n \in \mathcal{N}$ according to access mode $\overline{a} \in A$ for user $u \in U$ having a role $r \in R$ active on object $o \in O$
$over_{AO}(o_1, o_2$, s,$\overline{a})$	needed in the definition of some of the overriding policies
$over_{AS}(s_1$, o, s_2,$\overline{a})$	needed in the definition of some of the overriding policies
error	captures an error due to the violation of some integrity constraints
in(e_1, e_2, e_3)	represents indirect membership of elements belonging to either AOH or ASH: $e_1, e_2 \in AO \cup AS$, $e_3 = AOH$ or $e_3 = ASH$
dirin(e_1, e_2, e_3)	represents direct membership of elements belonging to either AOH or ASH: if $e_1, e_2 \in AO$, $e_3 = AOH$, if $e_1, e_2 \in AS$, $e_3 = ASH$
isuser(s)	returns true if $s \in AS$ is a user
isgroup(s)	returns true if $s \in AS$ is a group
isrole(s)	returns true if $s \in AS$ is a role
owner(o,u)	represents the ownership of an object $o \in O \cup T$ by a user $u \in U$

Table 1: Jajodia et al. predicates and their meaning

set of authorizations entailed by a specific access control model. The basic elements used to represent a model are: Obj, a set of objects; T, a set of types (named groups of objects); U, a set of users; G, a set of groups, R, a set of roles (named groups of privileges); A, a set of authorization modes; $SA = \{+a, -a \mid a \in A\}$, a set of signed authorization modes; Rel, a set of unary, binary or n-ary relationships defined over the elements presented above. Elements in Rel represent application domain predicates capturing some relevant relationships for the considered application (the last part of Table 1 shows some examples of Rel predicates). Objects, types, users, groups, and roles [2] are assumed to be hierarchically organized, according to the following basic hierarchies: *Object-Type* (OTH), *User-Group* (UGH), and *Role* (RH). Since an authorization subject AS can be a user, a group or a role, whereas an authorization object AO can be an object, a type or a role, the basic hierarchies are composed together to define two additional hierarchies, the *Authorization Subject Hierarchy* (ASH) (combining together the User-Group and the Role hierarchy) and the *Authorization Object Hierarchy* (AOH) (combining together the Object-Type and the Role hierarchy).

An access control model in the Jajodia et al. framework corresponds to a a tuple $(OTH, UGH, RH, A,$
$Rel)$, called *Data System* DS. An authorization is a triple $(o, s, \langle sign \rangle a)$, where o is an authorization object, s is an authorization subject, and $\langle sign \rangle a \in SA$ is a signed authorization mode.

Logic programs are constructed upon a logical language, called *Authorization*

[2] Jajodia et al. assumes that each user can activate at most one role at time.

Specification Language (ASL). An *Authorization Specification AS* is then defined as a set of stratified rules satisfying some syntactic restrictions [6] that limit the set of predicates that can appear in their bodies. The resulting program is locally stratified, thus it admits just one stable model. Authorizations are specified through predicates `cando`, `dercando`, and `do`. Predicate `cando` represents explicit authorizations, predicate `dercando` represents derived authorizations whereas predicate `do` is used to solve conflicts. Authorizations may propagate along the AOH and the ASH hierarchies according to the specified `dercando`, $over_{AS}$, and $over_{AO}$ rules.

Since both positive and negative authorizations are allowed, conflicts may arise among the computed set of authorizations. To solve this problem, `do` rules are used. In some sense, `do` rules represent a specific conflict resolution policy. The policy is applied at the end of the derivation process, differently from LAMP, where the conflict resolution function is used during authorization computation.

$Obj = \{o1, o2, o3\}$ $T = \emptyset$ $U = \{Dan, Bob, Jill\}$ $G = \emptyset$
$R = \{r1, r2, r3\}$ $A = \{R, X, W\}$ $SA = \{+R, +X, +W, -R, -X, -W\}$ $Rel = \emptyset$
$OTH = (Obj, \emptyset, \leq_T)$ $\forall x, y \in Obj : y \leq_T x \Rightarrow y = x$ objects are flat
$UGH = (U, \emptyset, \leq_{UG})$ $\forall x, y \in U : y \leq_{UG} x \Rightarrow y = x$ users are flat
$RH = (\emptyset, R, \leq_R)$ $x \leq_R y \; iff \begin{cases} x \in \{r1, r2\} \;\&\; y = r1 \\ x \in \{r1, r3\} \;\&\; y = r1 \end{cases}$ r2 and r3 are specializations of r1
$ASH = (U, R, \leq_{AS})$ $x \leq_{AS} y \; iff \begin{cases} x, y \in U \;\&\; x \leq_{UG} y \\ x, y \in R \;\&\; x \leq_R y \end{cases}$
$AOH = (Obj, R, \leq_{AO})$ $x \leq_{AO} y \; iff \begin{cases} x, y \in Obj \;\&\; x \leq_{OT} y \\ x, y \in R \;\&\; y \leq_R x \end{cases}$
$1 : cando(o1, r1, +R) \leftarrow$ $2 : cando(o1, r1, +W) \leftarrow$ $3 : cando(o2, r2, +R) \leftarrow$
$4 : cando(o3, r3, +W) \leftarrow$ $5 : cando(r1, Dan, activate) \leftarrow$ $6 : cando(r2, Bob, activate) \leftarrow$
$7 : cando(r3, Jill, activate) \leftarrow$
$8 : dercando(o, s, +a) \leftarrow cando(o, s', +a) \;\&\; in(s', s, ASH)$
$9 : do(o, s, +a) \leftarrow dercando(o, s, +a)$
$10 : do(o, s, +a) \leftarrow \neg do(o, s, -a)$

Figure 2: Example of Jajodia et al. usage

Example 2 *Consider the scenario illustrated in Example 1 and suppose you want to model it by using the Jajodia et al. framework. Figure 2 presents the needed basic components. The meaning of the rules presented in Figure 2 is the following: i) rules* 1 *and* 2 *authorize role* $r1$ *to exercise privileges* R *and* W *on* $o1$*; ii) rule* 3 *authorizes role* $r2$ *to exercise privilege* R *on* $o2$*, whereas rule* 4 *authorizes role* $r3$ *to exercise privilege* W *on* $o3$*, iii) rules* 5, 6*, and* 7 *specify that roles* $r1$*,* $r2$*, and* $r3$ *have been respectively activated by* Dan*,* Bob*, and* $Jill$*; iv) rule* 8 *propagates positive authorizations according to the* ASH *hierarchy; vi) rule* 9 *allows the computation of the allowed accesses for subjects on objects according to the authorizations generated in the system; vii) finally, rule* 10 *is used to guarantee that, for every possible access request* `(o,s,+a)`*, exactly one fact between* `do(o,s,+a)` *and* `do(o,s,-a)` *will hold.* □

4 The NIST RBAC framework

NIST is a general framework for modeling RBAC models proposed by Sandhu et al. [2]. It is defined by four levels of increasing complexity such that each level adds to the previous one new features. These levels are described in the following.

Flat RBAC. *Flat RBAC* is the base level, able to capture the basic classical features of an RBAC model: users acquire permissions from roles; a user can be assigned to many roles and a role can refer to many users (the same holds for the relation existing between permissions and roles); users can simultaneously exercise permissions deriving from different roles. Additionally, Flat RBAC supports user-role review, that is, it must be possible to determine which roles are assigned to a specific user and which are the users authorized to play a specific role.
Hierarchical RBAC. *Hierarchical RBAC* adds to Flat RBAC the support for role hierarchies. Two different interpretations of role hierarchies are provided: the inheritance and the activation one. In the first case, the activation of a role r_i implies the activations of all roles r_j that are less powerful than r_i, and thus the inheritance of their permissions, whereas, in the second case, junior roles must be explicitly activated.
Constrained RBAC. *Constrained RBAC* adds to Hierarchical RBAC the support for separation of duty (SOD) constraints. Separation of duty is the ability to state which roles cannot be simultaneously assigned to the same user (static SOD) or which roles cannot be activated together by the same user (dynamic SOD).
Symmetric RBAC. *Symmetric RBAC* adds to Constrained RBAC the support for permission-role review. This is the ability to determine which are the roles to which a particular permission is assigned as well as which are the permissions assigned to a particular role.

The basic components of the NIST framework can be formally defined as follows:

- Sets $\overline{U}$, $\overline{R}$, $\overline{P}$, and $\overline{S}$, which represent, respectively, the sets of users, roles, permissions, and sessions. Each permission is a pair (a, o) and models a specific access mode a on object o. We thus denote with $\overline{A}$ and $\overline{O}$ the sets of access modes and objects, respectively. Thus, $\overline{P} \subseteq \overline{A} \times \overline{O}$. Moreover, let $p \in \overline{P}$ be a permission, we denote with p_a and p_o the access mode and the object in p, respectively.
- $\overline{X} \subseteq \overline{U} \times \overline{R}$ and $\overline{Y} \subseteq \overline{P} \times \overline{R}$ represent, respectively, the user-role and the permission-role assignments. Let $x \in \overline{X}$, we denote with x_u and x_r the user and role specified in x. Similarly, let $y \in \overline{Y}$, we denote with y_p and y_r the permission and role specified in y.
- $\overline{H} \subseteq \overline{R} \times \overline{R}$ represents the role hierarchy; $\forall r_i, r_j \in \overline{R}$, $\langle r_i, r_j \rangle \in \overline{H}$, if r_j precedes r_i in the role hierarchy (according to a role dominance relationship $\leq$).
- $user : \overline{S} \rightarrow \overline{U}$ is a function that maps each session onto a single user.
- $roles : \overline{S} \rightarrow 2^{\overline{R}}$ is a function that maps each session onto a set of roles de-

fined as follows: $\forall s \in \overline{S}$, $roles(s) \subseteq \{r_i \in \overline{R} | \langle r_i, r_j \rangle \in \overline{H}, x \in \overline{X}, x_u = user(s), x_r = r_j\}$; session s has the following permissions: $\bigcup_{r_i \in roles(s)} \{p \in \overline{P} | \langle r_k, r_i \rangle \in \overline{H}, y \in \overline{Y}, y_p = p, y_r = r_k\}$.

- $\overline{C}$ represents the set of specified constraints.

$\overline{U} = \{Dan, Bob, Jill\}$	$\overline{P} = \{p1 \equiv (R, o1), p2 \equiv (W, o1), p3 \equiv (R, o2), p4 \equiv (W, o3)\}$
$\overline{O} = \{o1, o2, o3\}$	$\overline{X} = \{\langle Dan, r1\rangle, \langle Bob, r2\rangle, \langle Jill, r3\rangle\}$
$\overline{R} = \{r1, r2, r3\}$	$\overline{Y} = \{\langle p1, r1\rangle, \langle p2, r1\rangle, \langle p3, r2\rangle, \langle p4, r3\rangle\}$
$\overline{A} = \{R, X, W\}$	$\overline{H} = \{\langle r2, r1\rangle, \langle r3, r1\rangle, \langle r1, r1\rangle, \langle r2, r2\rangle, \langle r3, r3\rangle\}$
$\overline{S} = \{sDan, sBob, sJill\}$	$\overline{C} = \emptyset$
$user(Dan) = sDan$	$roles(sDan) = \{r1, r2, r3\}$
$user(Bob) = sBob$	$roles(sBob) = \{r2\}$
$user(Jill) = sJill$	$roles(sJill) = \{r3\}$

Figure 3: Example of NIST usage

Example 3 *Consider the scenario illustrated in Example 1. Figure 3 presents its representation according to the NIST framework.* □

5 Comparative analysis

In the following, the LAMP framework is compared with the Jajodia et. al and NIST frameworks. We show that LAMP has a higher *expressive power* with respect to the other two frameworks. By expressive power of a framework, we mean the range of access control models the framework is able to represent.

5.1 Jajodia vs LAMP

In the following, we show how the general framework proposed by Jajodia et al. [6] can be represented by LAMP and how the converse is not true. In order to show that the Jajodia et. al framework can be represented by LAMP, we just show that for each authorization specification AS expressed in the Jajodia et. al framework, there exists an access control model instance $\mathcal{I}$ over an ACMS S in LAMP that entails the same set of authorizations.

subject$(self : subject)$	**object**$(self : object, name : string, own : user)$
group$(self : group, name : string)$	**role**$(self : role, name : string)$
privilege$(self : privilege, name : string)$	**user**$(self : user, name : string)$

Figure 4: Schema for Jajodia et. al classes

In LAMP, the Access Control Model Schema is used to define the structural components over which the model is built. In the following we present the ACMS needed to model the Jajodia et al. framework.

Access Control Model Schema. S consists of the following components:

1. $B = K \cup R$, such that:
   (a) $K = K_{built\_in} \cup K_{basic}$, where $K_{built\_in} = \{\bot, int, string\}$ and $K_{basic} = \{subject, user, group, object, role, privilege\}$;
   (b) $R = R_{domain} \cup R_{auth} \cup R_{constraint} \cup R_{user\_def}$, where $R_{domain} = \{SubG, InSubG, Belong, UserIn, LessR, InLessR, Play, UserPlay, PartOf, InPartOf\}$, $R_{auth} = \{Auth_d, Auth_p, Auth\}$, $R_{constraint} = \{ErrorC\}$, and $R_{user\_def} = \{CanDo, DerCanDo, Done, PlayG, GroupPlay, over_{AO}, over_{AS}, ActiveOneRole\}$;
2. function Scheme for the elements in K_{basic}, $R_{domain} \cup R_{auth}$, and R_{user_def}, presented in Figure 4, and in the second column of Table 2;
3. function ISA representing class-subclass organization, defined as follows: $ISA(user) = ISA(group) = ISA(role) = \{subject\}$, for all the other classes c, $ISA(c) = \emptyset$.
4. set A, containing the special name "self" and the attribute names appearing in Figure 4; in particular, the ownership of an object by a user is modeled by using an attribute $owner$ associated with objects;
5. Z containing oid's for the elements of K.

Based on the previous schema, the ACMI is constructed as follows; in the construction of the ACMI we have considered the correspondence between Jajodia et al. and LAMP predicates shown in Table 2.

Access Control Model Instance. Given an authorization specification AS over a Data System $DS = (OTH, UGH, RH, A, Rel)$, we construct an instance $\mathcal{I}$ over S as follows:

1. DC: we insert: *i)* a fact: $object(self : \#i, name : o, owner : u)$, for each $o \in Obj \cup T$, such that $\texttt{owner}(\texttt{o}, \texttt{u})$ holds in AS, *ii)* a fact: $privilege(self : \#i, name : p)$, for each $p \in A$, *iii)* a fact: $user(self : \#i, name : u)$, for each $u \in U$, *iv)* a fact: $group(self : \#i, name : g)$, for each $g \in G$, *v)* a fact: $role(self : \#i, name : r)$, for each $r \in R$, where $\#i \in Z$ denotes the unique identifier of each object i. Since "user", "group", and "role" are subclasses of "subject", the following sets of inherited elements $\{subject(self : \#i) | \exists\, user(self : \#i, name : u) \in DC\} \cup \{subject(self : \#j) | \exists\, group(self : \#j, name : g) \in DC\} \cup \{subject(self : \#k) | \exists\, role(self : \#k, name : r) \in DC\}$ hold.
2. DSC: predicates in R_{domain} are defined as follows:
   - for each $\texttt{dirin}(\texttt{e}_1, \texttt{e}_2, \texttt{e}_3) \in AS$, we insert in $\mathcal{I}$: *i)* the fact $Belong(U : e_1, G : e_2)$, if $e_1 \in U$, $e_2 \in G$, and $e_3 = ASH$; *ii)* the fact $SubG(G_1 : e_1, G_2 : e_2)$, if $e_1, e_2 \in G$, and $e_3 = ASH$; *iii)* the fact $LessR(R_1 :$

Jajodia et al. atom	Schema of equivalent LAMP atom	Equivalent LAMP atom
cando(o',s',$\overline{a}$), $s' \in U \cup G \cup R$, $o' \in Obj \cup T$	*CanDo*(O:object,S:subject,P:privilege, ϵ:string)	*CanDo*(O:o',S:s',P:p',ϵ:k), $\overline{a} \equiv kp'$
cando(o',s',activate), $o' \in R$, $s' \in U$	*Play*(U:user,R:role)	*Play*(U:s',R:o')
cando(o',s',activate), $o' \in R$, $s' \in G$	*PlayG*(G:group, R:role)	*PlayG*(G:s',R:o')
dercando(o',s',$\overline{a}$), $s' \in U \cup G \cup R$, $o' \in Obj \cup T$	$DerCanDo$(O:object,S:subject,P:privilege, ϵ:string)	$DerCanDo$(O:o',S:s',P:p',ϵ:k), $\overline{a} \equiv kp'$
do(o',s',$\overline{a}$), $s' \in U \cup G \cup R$, $o' \in Obj \cup T$	$Auth_d$(O:object,S:subject,P:privilege,G:subject, ϵ:string)	$Auth_d$(O:o',S:s',P:p',G:u',ϵ:k), $\overline{a} \equiv kp'$
done(o',u',r',a',n'), $o' \in Obj \cup T$	*Done*(O:object,U:user,R:role,P:privilege,T:integer)	*Done*(O:o',U:u',R:r',P:a',T:n')
over$_{AO}$(o', o", s',$\overline{a}$), $s' \in U \cup G \cup R$, $o', o'' \in Obj \cup T$	$Over_{AO}$(O_1:object, O_2:object,S:subject, P:privilege, ϵ:string)	$Over_{AO}$(O_1:o', O_2:o",S:s', P:p', ϵ:k), $\overline{a} \equiv kp'$
over$_{AS}$(s', o', s",$\overline{a}$), $s' \in U \cup G \cup R$, $o' \in Obj \cup T$	$Over_{AS}$(S_1:subject, O:object,S_2:subject, P:privilege, ϵ:string)	$Over_{AS}$(S_1:s', O:o',S_2:s", P:p', ϵ:k), $\overline{a} \equiv kp'$
error()	*ErrorC*()	*ErrorC*()
dirin(e'_1, e'_2, e'_3), $e'_1 \in U$, $e'_2 \in G$, $e'_3 = ASH$	*Belong*(U:user,G:group)	*Belong*($U : e'_1$,$G : e'_2$)
dirin(e'_1, e'_2, e'_3), $e'_1, e'_2 \in G$, $e'_3 = ASH$	*SubG*(G_1:group,G_2:group)	*SubG*($G_1 : e'_1$,$G_2 : e'_2$)
dirin(e'_1, e'_2, e'_3), $e'_1, e'_2 \in R$, $e'_3 = ASH$	*LessR*(R_1:role,R_2:role)	*LessR*($R_1 : e'_2$,$R_2 : e'_1$)
dirin(e'_1, e'_2, e'_3), $e'_3 = AOH$ and either $e'_1 \in Obj$ and $e'_2 \in T$, or $e_1, e_2 \in T$	*PartOf*(O_1:object,O_2:object)	*PartOf*($O_1 : e'_1$,$O_2 : e'_2$)
in(e'_1, e'_2, e'_3), $e'_1 \in U$, $e'_2 \in G$, $e'_3 = ASH$	*UserIn*(S:subject,G:group)	*UserIn*($S : e'_1$,$G : e'_2$)
in(e'_1, e'_2, e'_3), $e'_1, e'_2 \in G$, $e'_3 = ASH$	*InSubG*(G_1:group,G_2:group)	*InSubG*($G_1 : e'_1$,$G_2 : e'_2$)
in(e'_1, e'_2, e'_3), $e'_1, e'_2 \in R$, $e'_3 = ASH$	*InLessR*(R_1:role,R_2:role)	*InLessR*($R_1 : e'_1$,$R_2 : e'_2$)
in(e'_1, e'_2, e'_3), $e'_1 \in Obj$, $e'_2 \in T$, $e'_3 = AOH$	*InPartOf*(O_1:object,O_2:object)	*InPartOf*($O_1 : e'_1$,$O_2 : e'_2$)
isuser(s')	attribute *(name:string)* of class name subject	attribute *(name:s')* of subject s'
owner(o',u')	attribute *(own:user)* of class name object	attribute *(own:u')* of object o'
if user $u' \in U$ has activated role $r' \in R$ during its session	*ActiveOneRole*($U : user, R : role$)	*ActiveOneRole*($U : u'$, $R : r'$)

Table 2: Correspondence between atoms

$e_2, R_2 : e_1$), if $e_1, e_2 \in R$, and $e_3 = ASH$; *iv)* the fact $PartOf(G_1 : e_1, G_2 : e_2)$, if $e_3 = AOH$ and either $e_1 \in Obj$ and $e_2 \in T$, or $e_1, e_2 \in T$.

- for each role $r \in R$ and for each user $u \in U$ that is explicitly authorized to play role r, we insert in $\mathcal{I}$ the fact: $Play(U : u, R : r)$.
- we insert in DSC also the rules defining predicates $UserIn$, $InSubG$, $InLessR$,$InPartOf$, and $UserPlay$:
  - $UserIn(U : X, G : Y) \leftarrow Belong(U : X, G : Y)$,
    $UserIn(U : X, G : Y) \leftarrow Belong(U : X, G : Z), InSubG(G_1 : Z, G_2 : Y)$,

- $InSubG(G_1 : X, G_2 : Y) \leftarrow SubG(G_1 : X, G_2 : Y)$,
  $InSubG(G_1 : X, G_2 : Y) \leftarrow SubG(G_1 : X, G_2 : Z), InSubG(G_1 : Z, G_2 : Y)$,
- $InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Y)$,
  $InLessR(R_1 : X, R_2 : Y) \leftarrow LessR(R_1 : X, R_2 : Z), InLessR(R_1 : Z, R_2 : Y)$,
- $InPartOf(O_1 : X, O_2 : Y) \leftarrow PartOf(O_1 : X, O_2 : Y)$,
  $InPartOf(O_1 : X, O_2 : Y) \leftarrow PartOf(O_1 : X, O_2 : Z), InPartOf(O_1 : Z, O_2 : Y)$,
- $UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Y)$,
  $UserPlay(U : X, R : Y) \leftarrow Play(U : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$;

note that previous rules represent the transitive closure of predicates $Belong$, $SubG$, $LessR$, $PartOf$, and $Play$, respectively.

3. AC: predicates in R_{user_def} are defined as follows:
   - for each user $u \in U$ having an activated role $r \in R$, we insert in $\mathcal{I}$ the fact: $ActiveOneRole(U : u, R : r)$;
   - predicate $CanDo$ is defined by the same rules defining predicate `cando` in AS, by replacing each predicate with the corresponding predicate shown in Table 2; the only exception concerns `cando` atoms having a role as object. In this case, we use either $Play$ or $PlayG$ atoms, depending on the type of the `cando` subject (user or group) that is authorized to play that role;
   - the following rules represent the transitive closure of the $GroupPlay$ predicate:
     - $GroupPlay(G : X, R : Y) \leftarrow PlayG(G : X, R : Y)$,
       $GroupPlay(G : X, R : Y) \leftarrow PlayG(G : X, R : Z), InLessR(R_1 : Y, R_2 : Z)$;
   - predicate $DerCanDo$ is defined by the same rules defining predicate `dercando` in AS, by replacing each predicate with the corresponding predicate shown in Table 2.
   - predicates $Over_{AO}$ and $Over_{AS}$ are defined by the same rules defining predicates $\texttt{over}_{AO}$ and $\texttt{over}_{AS}$ in AS, by replacing each predicate with the corresponding predicate shown in Table 2.
   - for each $\texttt{done}(o, u, r, a, n) \in AC$, we insert in $\mathcal{I}$ the fact $Done(O : o, U : u, R : r, P : a, T : n)$;
   - predicate $Auth_d$ is defined as predicate `do` in AS, by replacing each predicate with the corresponding predicate shown in Table 2.
4. PC: No definition for predicate $Auth_p$ is required, since propagation rules are included in the definition of predicates $CanDo$ and $DerCanDo$; the only exception is represented by the following rule needed to define predicate $Auth$: $Auth_p(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_1, S' : X_2, P' : X_3) \leftarrow Auth_d(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5)$. Predicate $Auth$ is defined by the following rule: $Auth(O : X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5) \leftarrow Auth_p(O :$

$X_1, S : X_2, P : X_3, G : X_4, \epsilon : X_5, O' : X_6, S' : X_7, P' : X_8)$.

5. CC: predicate $ErrorC$ is defined by the same rules defining predicate `error` in AS, by replacing each predicate with the corresponding predicate shown in Table 2.

The following theorem states that the Jajodia et al. framework is representable in the LAMP framework.

Theorem 1 *All the access control models that can be represented by the Jajodia et al. framework are representable by the LAMP framework.*

Proof sketch. Any access control model that can be represented by the Jajodia et al. framework corresponds to a specific authorization specification AS. It is easy to prove that, given an authorization specification AS, if $\mathcal{I}$ is the ACMI constructed as above, the following conditions hold: I) facts that hold for predicate `do` in AS coincide with facts that hold for predicate $Auth$ in $D(\mathcal{I})$,[3]; II) facts that hold for predicate `error` in AS coincide with facts that hold for predicate $ErrorC$ in $D(\mathcal{I})$.

By construction : i) the domains over which variables in AS range correspond to the domain over which variables in $D(\mathcal{I})$ range; ii) the definition of predicates `dirin` and `in` in AS corresponds to the definition of predicates $Belong$, $SubG$, $LessR$, $PartOf$, and of rules specifying their transitive closure in $D(\mathcal{I})$; iii) the definition of predicates `isuser`, `owner` in AS corresponds to the definition of class names $subject$ and $object$ in $D(\mathcal{I})$. Thus, AS and $D(\mathcal{I})$ generate the same facts for predicates `cando`, `dercando`, and `done`, since in the two programs these predicates are defined in the same way, by using equivalent predicates. Moreover, in $D(\mathcal{I})$ $Auth$ is equivalent to predicate $Auth_d$, whose definition coincides with that of predicate `do` in AS, as shown in Table 2. So, the first initial condition has been proved. With a similar reasoning, it follows that facts that hold for predicate `error` in AS coincide with facts that hold for predicate $ErrorC$ in $D(\mathcal{I})$. Concluding, all the access control models that can be represented inside the Jajodia et. al. framework can also be represented in LAMP. □

By contrast, the converse is not true as stated by the following theorem.

Theorem 2 *The set of the access control models that can be represented by LAMP is greater than the one representable by the Jajodia et al. framework.*

Proof sketch. The logical language for specifying authorization rules proposed by Jajodia et al. [6] allows one to write locally stratified logic program generating a unique set of authorizations (only one stable model). By contrast, LAMP is based on a more general formalism: each program can generate more than one set of authorizations (one for each stable model of the program). Thus, LAMP allows a multiplicity of models to be associated with a given program. Each of these models represents a

[3] Given an ACMI $\mathcal{I}$, we denote with $D(\mathcal{I})$ the corresponding Datalog-like program.

set of consistent authorizations that can be possibly assigned to subjects. This aspect is crucial every time that we are interested in modeling an access control model that supports authorization rules able to generate different sets of consistent authorizations. □

5.2 NIST vs LAMP

In the following, we show how the NIST framework [2], can be represented by the LAMP framework and how the converse is not true. In order to show that the NIST framework can be represented by the LAMP framework, we have to show that for each access control model represented according to one of the NIST levels, there exists an ACMI $\mathcal{I}$ over an ACMS S that entails the same set of authorizations, as stated by the following theorem.

Theorem 3 *All the access control models that can be represented by the four NIST levels are representable by the LAMP framework.*

Proof sketch. In [1] we have shown that the first two NIST levels are representable by the LAMP framework. Since LAMP supports the representation of arbitrary integrity constraints, static and dynamic separation of duty can be modeled by specific constraint rules as conditions on roles, users, sessions, user-role and permissions-role assignments. All these conditions can be directly represented in Datalog, thus they can be represented as constraint rules in an ACMI. Moreover, since the ACMI corresponding to a Constrained RBAC does not contain negation, it has a single stable model. Thus, permission-role review can be performed by directly inspecting the authorization base by executing a specific query over the ACMI. Concluding, any instance of Constrained and Symmetric RBAC can be modeled in LAMP. □

By contrast, the converse is not true as stated by the following theorem.

Theorem 4 *The set of the access control models that can be represented by LAMP is greater than the one representable by the NIST framework.*

Proof sketch. The main differences between NIST and LAMP can be summarized as follows: i) RBAC is able to propagate authorizations only for subjects whereas LAMP supports propagation also along object and privilege hierarchies (multiple vs. single propagation); ii) RBAC can specify only a limited form of constraints on subjects and roles, whereas LAMP constraint rules can express a broader set of constraints, also due to the fact that user-defined predicates are allowed in constraint rule bodies (user-defined and constrained rules vs. separation of duty). In the following, based on the previous considerations, we prove the theorem by showing an example of constraint that can be represented in LAMP but not in NIST. The constraint we consider states that a user is denied to activate two roles $r1$ and $r2$ inside a session when at least two

other users have already activated such roles. This constraint represents a sort of "conditioned separation of duty". It depends on information that dynamically changes (the existence of a couple of users, which one is Bob, having activated $r1$ and $r2$ inside their sessions). In particular, the activation of this constraint depends on the behaviour of a specific user: only if Bob has activated both roles, we limit the possibility to activate such roles just to another user, otherwise there is no limit to the number of users that can do it. This constraint cannot be represented in NIST. In fact in the last proposed NIST RBAC standard, [3] dynamic separation of duty (DSD) constraints are defined as collections of pairs (rs, n), such that each rs represents a role set, whereas n is a natural number ≥ 2. The meaning of a DSD constraint (rs, n), is that no subject may activate n or more roles among those contained in the set rs. According to this syntax, there is no way to express a conditioned separation of duty, like the one above expressed, which depends on specific values of subject and/or object attributes. On the other hand, the constraint presented above can be represented in LAMP by the following constraint rule:

$$ErrorC \leftarrow ActiveRole(U : X_1, S : Y_1, R : r1), ActiveRole(U : X_1, S : Y_1, R : r2),$$
$$UserPlay(U : X_1, R : r1), UserPlay(U : X_1, R : r2), ActiveRole(U : Bob, S : Y_2,$$
$$R : r1), ActiveRole(U : Bob, S : Y_2, R : r2),$$
$$UserPlay(U : Bob, R : r1), UserPlay(U : Bob, R : r2), ActiveRole(U : X_2,$$
$$S : Y_3, R : r1), ActiveRole(U : X_2, S : Y_3, R : r2), UserPlay(U : X_2, R : r1),$$
$$UserPlay(U : X_2, R : r2), X_1 \neq X_2, X_1 \neq Bob, X_2 \neq Bob,$$
$$Y_1 \neq Y_2, Y_1 \neq Y_3, Y_2 \neq Y_3. \qquad \square$$

6 Conclusions

In an Adaptive Security Infrastructure, an important issue consists in the formal representation and analysis of security policies. In this paper, we made a step towards this direction, by focusing on access control policies. Since in the literature several frameworks have been proposed for policy specification, in this paper we investigated the expressive power of three different access control frameworks, namely, the LAMP framework [1], the Jajodia et al. framework [6], and the NIST one [2]. From the presented results, it follows that LAMP subsumes the other frameworks, that is, all the models that can be modeled in Jajodia et al. and the NIST frameworks can be modeled in LAMP but the converse is not true. Given a distributed system based on ASI, our analysis will help in the selection of a specific logical framework for such an environment. Moreover, it represents a first step towards the development of a formal theory for comparing and analysing access control model frameworks. Future work concerns the comparison of the presented frameworks according to other dimensions, for example the mapping complexity, that is, the computational complexity of representing a certain access control model inside the framework, the spatial complexity, that is, the amount of information used by a framework to model a certain policy, and

the temporal complexity, that is, the complexity of querying the authorization base according to the computational semantics supported by the framework. Additionally, we plan to develop a set of tools for specifying and analyzing access control policies under ASI using LAMP as core-system. A prototype is currently being developed at the University of Milano.

References

[1] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transaction on Information and System Security (TISSEC)*, 6(1), February 2003.

[2] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transaction on Information and System Security (TISSEC)*, 4(3):224–274, August 2001.

[3] American National Standard for Information Technology. Role Based Access Control, 4 2003. (http://csrc.nist.gov/rbac/rbac-std-ncits.pdf).

[4] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[5] S. Greco, N. Leone, and P. Rullo. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transactions on Knowledge and Data Engineering*, 4:72–87, 1992.

[6] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26(2):214–260, June 2001.

[7] A.Y. Levy, I.S. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, Query-Reachability, and Satisfiability in Datalog Extensions. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 109–122, 1993.

[8] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[9] O. Shmueli. Equivalence of Datalog Queries is Undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

Formalizing an Adaptive Security Infrastructure in Mob$_{adtl}$

Carlo Montangero Laura Semini
Dipartimento di Informatica, Università di Pisa,
{monta,semini}@di.unipi.it

Abstract

We discuss an approach to the logic specification of an Adaptive Security Infrastructure and, within it, of a specific adaptive policy, namely a routing one. Our proposal is based on previous work on logic based specification of secure mobile systems. The approach builds on Mob$_{adtl}$, a model for communication and mobility that has been formalized in the ΔDSTL(x) spatial-temporal logic.

1 Introduction

A lot of research is underway to develop infrastructures, programming languages and methodologies for the specification, implementation and verification of secure networking applications. In particular, interest is growing for *adaptive security*, where the security policies are not fixed, but vary in response to analysis of the sensed state of the distributed environment. Many sophisticated capabilities of intrusion detection, data mining, self-reconfiguring systems, policy management etc. are being developed, but there is no agreement yet on a unifying logical view of the general aspects of adaptive security systems. For example, it is not known how to prove (or even specify) capabilities or deduce rigorously the appropriate responses to security-relevant inputs. Also, issues arising from considering how an infrastructure for adaptive security (ASI) could be specified, designed, and verified are still wide open.

This paper will present tentative answers to some of the related questions, namely,

- How should the semantics of a dynamic security policy be specified, one that can deal with potential future security questions and facilitate proof that a candidate response is in fact consistent with current policy?
- How should we take into account the global-local (or distributed-centralized or hierarchical) nature of all components of an ASI?
- How should we specify the ”security-relevant resources” available so that at any time the analyzer can choose an appropriate response?

- How should we unify the temporal-spatial reasoning aspects?

Our answers are based on previous work on logic based specification of secure mobile systems [8]. The approach builds on Mob$_{adtl}$, a model for communication and mobility [18] that has been formalized in the ΔDSTL(x) spatial-temporal logic [15, 14, 12]. The most important characteristics of the logic are the following ones:

- thanks to novel Kripke models, it has enough expressive power to reason on the behavior of systems where communication is fully asynchronous, and it permits to exploit local theories smoothly in the proofs of distributed properties;
- it permits to name the components of a distributed system and to causally relate properties which might hold in distinguished components;
- it uses only a subset of linear time temporal logic, with a limited number of operators à la Unity;
- it has a primitive operator, Δ, to specify events;
- it supports a structured form of refinement that permits to build generic models and then specialize them in separate chunks;
- it has a semi-automatic proof support, the Mob$_{adtl}$ Reasoning Kit MaRK [9], built on top of Isabelle [16].

In Mob$_{adtl}$, the network infrastructure consists of a net of elaboration nodes called *neighborhoods*. A neighborhood is a bounded environment where computational entities live. The net of neighborhoods is populated by elaboration units called *agents*. Agents communicate remotely and can move from one neighborhood to another. Each neighborhood is associated with one particular stationary entity called *guardian*.

Each guardian monitors its neighborhood limiting the resources the agents can use, intercepting messages and agents and deciding which of them can enter or leave the neighborhood they control. Guardians also provide the routing facilities needed to forward messages and to handle moving agents. Since there is one guardian per neighborhood and guardians do not move, we can identify a neighborhood with its guardian.

In this paper we present only the Mob$_{adtl}$ communication features, for space reasons. The ones regarding mobility follow the same philosophy and are described in previous work (e.g. [18, 9]).

Communications are based on fully asynchronous message passing and occur under to the mediation and control of the guardians, at least those at the ending points of a message exchange.

The net of neighborhoods is flat, i.e. neighborhoods are not nested. This does not imply a loss of generality, since routing policies can be specified to describe more structured topologies, e.g. one where a neighborhood plays the role of firewall for a set of neighborhoods.

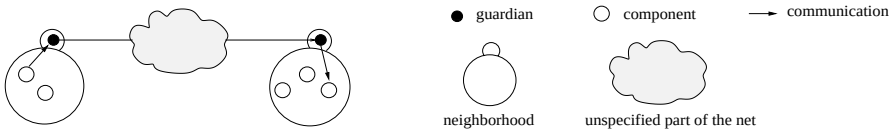


Figure 1: The Mob$_{adtl}$ model: two neighborhoods with their guardians and communicating agents.

There is a mechanism of *profiles* to identify entities through a set of properties they are called to satisfy, and an exception mechanism to deal with the impossibility of resolving a profile, the violation of a security policy, and the failure of the network infrastructure.

Figure 1 presents a pictorial representation of two neighborhoods with their guardians and agents that communicate and move.

Coming to security, neighborhoods basically reflect administrative domains, where components run under the control of a specific authority. In our case, the guardian acts as an interposition interface among agents and neighborhoods: it specifies and implements communication (and mobility) policies. In other words, guardians monitor the components and limit the resources they can use. More precisely, communication occur via the guardians that provide routing facilities to forward messages: they intercept messages and decide which messages can enter or leave the neighborhood they control, for instance because of security reasons.

The Mob$_{adtl}$ model itself does not embody any policy: security policies must be explicitly specified through suitable refinement steps. In the following, we will assume that one such policy has been specified, to give each guardian the ability to prevent a message from being routed through a guardian that is not trusted. At the core of the specification of this policy stays predicate *trusted(G)*: it holds in a guardian if and only if its own security requirements are fulfilled by *G*. Then, to adapt the policy we will need to find ways to update the information encoded by the *trusted* predicate.

To show how, in our framework, one can approach the logic specification of an ASI and, within it, of specific adaptive policies, we will use a simple scenario. The infrastructure is as described in the WOLFASI call [1]. Its conceptual components are Detector, Analyzer, and Responder:

- the *Detector* senses, collects, and distributes information about the security environment;
- the *Analyzer* processes Detector data, along with other information (e.g. security policy, threat levels, or node trust levels) and occasionally proposes actions to bring about a new state;
- the *Responder* executes the actions as directed by the Analyzer. These actions could include adjusting preventive mechanisms, adjusting detector settings, adjusting internal system parameters, etc.

In our scenario, the agents exchange messages that are routed by the guardians, each with its own policy, expressed as a function of the trustworthiness of the guardians. As the simplest source of information about the security environment, we imagine that the agents can issue security warnings that question the trustworthiness of some guardian. The warnings are intercepted by the Detector, and sent to the Analyzer, which implements a simplistic strategy: once the number of warnings about a guardian reaches a given threshold, it proposes to consider that guardian no longer trustworthy to route the messages. The Responder implements this change in the policy, affecting the other guardians consequently.

The Detector and the Responder will be specified as refinements of the generic guardians of Mob$_{adtl}$, while the Analyzer will be specified as an independent agent, albeit in a dedicated neighborhood for simplicity (and security). Indeed, the former ASI components need to interfere with the Mob$_{adtl}$ communication infrastructure, while the Analyzer is best seen as a functional component of the ASI.

The next two sections introduce the logic and the model, respectively. Section 4 present the formalization of the infrastructure, the example policy and discusses the properties that can be proved. The last section discusses related work.

2 $\Delta DSTL(x)$: a mini–tutorial

2.1 Syntax

We assume a denumerable set of component names $\{m, n, m_1, m_2, \ldots\}$, and a denumerable set of variables, which includes the set of component variables, $\{M,\ N,\ M_1,\ M_2, \ldots\}$.

We introduce location modalities for each component in a system: we use component names, with a different font. For instance, $\mathsf{m_1}$ is the location modality corresponding to component m_1, and $\mathsf{m_1}Iam(m_1)$ stands for "in component m_1, $Iam(m_1)$ holds". We let quantifiers range over modalities, and M , N , $\mathsf{M_i}$ ... are location modality variables. Binding between location variables and regular variables is possible. For example, saying that for all M, $\mathsf{M}Iam(M)$ holds, means that for all components m_i, $\mathsf{m_i}Iam(m_i)$ holds. Quantification over modality variables is done in a standard way, following, for instance [6].

$$
\begin{array}{rcl}
F & ::= & A \mid \bot \mid \sim F \mid F \wedge F' \mid \Delta F \mid \mathsf{M_i}F \\
\phi & ::= & F \mid \exists F \mid F \ \textsc{leads\_to}\ F' \mid F \ddot{\cdot} F' \mid F \ \textsc{leads\_to\_c}\ F' \mid F \ \textsc{because\_c} \\
 & & F' \mid \textsc{init}\ F \mid F \ \textsc{unless}\ F'
\end{array}
$$

The first equation defines distributed state formulae, which are used to build $\Delta DSTL(x)$ formulae: A is an atom, $\bot$ is the propositional constant *false*, ΔF is an event. With $\bar{\mathsf{M}}_\mathsf{i}$ we denote the dual of $\mathsf{M_i}$, i.e., $\bar{\mathsf{M}}_\mathsf{i}F \equiv \sim \mathsf{M_i} \sim F$. With $\top$ we denote *true*, i.e. $\top \equiv \sim \bot$. The second equation defines $\Delta DSTL(x)$ formulae. They

are implicitly quantified, apart from those variables explicitly prefixed as existential. Otherwise, the intended meaning is that a formula F is universally quantified over all values of the variables appearing in the premises of F, and existentially quantified on the remaining variables. For instance, $\exists y.\ \mathsf{M}\Delta p(x) \rightarrow q(x,y)$, is implicitly prefixed by $\forall M, x$, and $\mathsf{M}\, p(x) \wedge q(y)$ LEADS_TO $\mathsf{N}\, r(x, M, z)$ should be understood as prefixed by $\forall\, M, x, y\ \ \exists\, N, z$. The variables in INIT are all universally quantified. A special case of UNLESS is stability: STABLE $F \stackrel{def}{=} F$ UNLESS $\bot$

The domain over which a variable is quantified (i.e. its sort) can be understood from the context or explicitly defined. We assume that these domains are invariant during time and in space.

The next sections informally present the semantics of the logic. The formal definitions are in [12, 15].

2.2 Models

The models for $\Delta DSTL(x)$ formulae are built on n–partite directed acyclic graphs like the one in Figure 2, which describes the computation of a system with two components. Here, p, q, ... are the properties holding in the states, arrows from a component to another denote communications, and arrows in one component denote local state transitions, as the computation progresses locally.

We call S_i the set of states of component m_i, and S the set of all the states of the computation. A *distributed state* is any subset ds of S, i.e. any set of states, and ds^0 is the set of the initial states. The states represented with $\circ$ in the figure precede the initial states, and are used to give semantics to the events in the initial state. We say that ds *follows* ds' if and only if each state in ds' is followed by a state in ds, and each state in ds is preceded by a state in ds', where a state s follows s' if and only if there is a path in the model graph from s to s'.

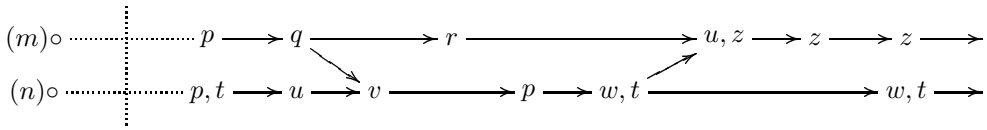


Figure 2: Models for $\Delta DSTL(x)$ formulae.

2.3 Semantics by examples

F, $\exists F$ a distributed state formula is also a $\Delta DSTL(x)$ formula, with the meaning of being an invariant: all distributed states must satisfy the formula. For instance, $\exists G.\ (\mathsf{A}\top \rightarrow \mathsf{A}guardedby(G))$ means that, chosen an arbitrary state of an arbitrary agent a, there is a guardian, g, such that in a predicate $guardedby(g)$ holds, i.e. all agents are always guarded by some guardian.

In particular, a distributed state ds satisfies the distributed state formula

- $\mathsf{m}F$ iff ds contains a (singleton) state of component m satisfying F. Consider, in figure 2, the distributed state ds composed of the first two states of m: ds satisfies $\mathsf{m}p$, $\mathsf{m}q$, and $\mathsf{m}p \wedge \mathsf{m}q$. On the contrary, ds does not satisfy $\mathsf{m}(p \wedge q)$: no singleton state satisfies the conjunction.
- ΔF iff ds is a model for F, and there exists a state $ds' <_l ds$ not satisfying F. We say that $ds' <_l ds$ if all the states in ds have a local (same component) immediate predecessor in ds', and all the states in ds' have a local immediate successor in ds. By definition, the "$\circ$" state of component m satisfies a literal L iff the initial state of m does not satisfy L: everything is an event in the initial state.

F LEADS_TO F' means that F is always followed by F': each distributed state satisfying F is followed by a distributed state satisfying F'. Operator LEADS_TO expresses a liveness condition, and is similar to Unity's $\mapsto$ (leads to). The difference with Unity is that properties can be localized to distinguished components, and that we can express events. For instance, the formula

$$\mathsf{A}\,\Delta moveTo(G) \;\text{ LEADS\_TO }\; \mathsf{A}\,guardedby(G) \wedge \mathsf{G}\,guarding(A)$$

is to be read as follows: for all agent a and all guardian g, each state of a where predicate $moveTo(g)$ becomes true, expressing the wish of a to move and go under the control of g, is followed by a state of a satisfying $guardedby(g)$ and by a state of g satisfying $guarding(a)$.

$F \because F'$ says that F must be preceded by F': BECAUSE is a safety operator, used to express "only if" temporal conditions. For instance, the formula above could have an "only if" counterpart, saying that an agent can be under the control of a guardian g only if it previously asked to move under g's control:

$$\mathsf{A}\,guardedby(G) \because \mathsf{A}\,moveTo(G)$$

Formally, a system satisfies $F \because F'$ if and only if each distributed state (not including initial states) that satisfies F is preceded by a distributed state satisfying F'.

FOL	axioms of the 1^{st} order logic	**K**	$\bar{\mathsf{M}}(F \rightarrow F') \rightarrow (\bar{\mathsf{M}}F \rightarrow \bar{\mathsf{M}}F')$
DSL1	$\bar{\mathsf{M}}(\bar{\mathsf{M}}F \leftrightarrow F)$	**DSL2**	$\mathsf{M} \neq \mathsf{N} \rightarrow \bar{\mathsf{M}}\bar{\mathsf{N}}\bot$
MP	$\dfrac{F \quad F \rightarrow F'}{F'}$	**Nec**	$\dfrac{F}{\bar{\mathsf{M}}F}$

Table 1: Axioms and rules of DSL(x).

• **Necessitation.** (We use $\vdash_{\Delta DSL(x)}$ and $\vdash_{\Delta DSTL(x)}$ for the sake of comprehension).

$$\frac{\text{for all } x_1 \dots x_n \text{ exists } y_1 \dots y_n \ \vdash_{\Delta DSL(x)} F}{\vdash_{\Delta DSTL(x)} \ \exists y_1 \dots y_n F} \mathbf{Nec}$$

• **Introduction and Elimination.** **LcI** $F \text{ LEADS\_TO\_C } F$ **BcI** $F \text{ BECAUSE\_C } F$ $\Delta\mathbf{E}$ $\Delta F \rightarrow F$

$$\frac{F \text{ LEADS\_TO\_C } G}{F \text{ LEADS\_TO } G} \mathbf{LI} \qquad \frac{F \text{ BECAUSE\_C } G}{F \cdot\!\!:\!\!\cdot G} \mathbf{BI} \qquad \frac{F}{\text{STABLE } F} \mathbf{SI}$$

$$\frac{F \text{ LEADS\_TO } G}{F \wedge \sim G \text{ LEADS\_TO } \Delta G} \mathbf{\Delta I} \qquad \frac{F \text{ LEADS\_TO } \bot}{\sim F} \mathbf{LE} \qquad \frac{\text{INIT} \sim F \ F \cdot\!\!:\!\!\cdot \bot}{\sim F} \mathbf{BE}$$

$$\frac{\text{INIT}\mathsf{M}F \ \ \text{STABLE}\ \mathsf{M}F}{\bar{\mathsf{M}}F} \mathbf{SE}$$

$$\frac{\mathsf{M}F \cdot\!\!:\!\!\cdot \mathsf{M}G \quad \text{STABLE}\ \mathsf{M}G}{\bar{\mathsf{M}}(F \rightarrow G)} \mathbf{BSE}$$

• **Transitivity.** $$\frac{F \text{ LEADS\_TO } F' \quad F' \text{ LEADS\_TO } G}{F \ \text{ LEADS\_TO } G} \mathbf{LTR} \qquad \frac{F \cdot\!\!:\!\!\cdot F' \quad F' \cdot\!\!:\!\!\cdot G}{F \ \cdot\!\!:\!\!\cdot G} \mathbf{BTR}$$

• **Premises and consequences strengthening and weakening.** Similar rules hold for BECAUSE, LEADS_TO_C, and BECAUSE_C.

$$\frac{\exists_{\mathrm{F}}\, G \rightarrow F \quad F \ \text{ LEADS\_TO } \ F' \quad \exists_{\mathrm{G}'}\, F' \rightarrow G'}{G \ \text{ LEADS\_TO } G'} \mathbf{LSW}$$

$$\frac{F \ \text{ LEADS\_TO } \ G \quad F' \ \text{ LEADS\_TO } \ G}{F \vee F' \ \text{ LEADS\_TO } G} \mathbf{LPD} \qquad \frac{G \ \text{ LEADS\_TO } \ F \quad G \ \text{ LEADS\_TO } \ F'}{G \ \text{ LEADS\_TO } F \wedge F'} \mathbf{LCC}$$

Table 2: Axioms and Rules of ΔDSTL(x). We list only those related to operators which are new with respect to Unity and those used in the proof of the ASI specification.

• **Notification and Confluence.**

$$\frac{F \cdot_{\cdot} G \;\; G \;\textsc{leads\_to}\; \mathsf{M}G' \;\; \textsc{stable}\; \mathsf{M}G'}{F \wedge \mathsf{M}\top \;\textsc{leads\_to}\; \mathsf{M}G'}\;\mathbf{Nf} \qquad \frac{\textsc{stable}\; \mathsf{M}F \;\; \textsc{stable}\; \mathsf{M}F'}{\mathsf{M}F \wedge \mathsf{M}F' \rightarrow \mathsf{M}(F \wedge F')}\;\mathbf{Cf}$$

$$\frac{\exists_{\mathrm{X}} \bar{\mathsf{M}} F(X)}{\exists_{\mathrm{X}} (\mathsf{M}\top \rightarrow \mathsf{M}F(X))}\;\mathbf{ID} \qquad \frac{\exists_{\mathrm{X}} (\mathsf{M}\top \rightarrow \mathsf{M}F(X))}{\exists_{\mathrm{X}} (\mathsf{M}G \rightarrow \mathsf{M}(G \wedge F(X)))}\;\mathbf{DC}$$

• **Properties of the initial state.**

$$\mathbf{I1}\;\; \textsc{init}\; \mathsf{m}\top \qquad \frac{\textsc{init}\; \mathsf{m}F}{\textsc{init}\; \bar{\mathsf{m}}F}\;\mathbf{I2} \qquad \frac{\textsc{init}\; \bar{\mathsf{m}}F}{\textsc{init}\; \mathsf{m}F}\;\mathbf{I3} \qquad \frac{\textsc{init}\; F \;\; F \rightarrow G}{\textsc{init}\; G}\;\mathbf{IW} \qquad \frac{\textsc{init}\; F \;\; \textsc{init}\; F'}{\textsc{init}\; F \wedge F'}\;\mathbf{IC}$$

• **Quantification.**
We recall that non temporal ΔDSTL(x) formulae are implicitly universally quantified in all variables with the exception of those explicitly bound by existential quantification.
Here, in the case of implication, with $\exists_{\mathrm{G}}\; F \rightarrow G$ we denote a formula universally quantified in all the variables of premise F, and existentially quantified in all the remaining variables of the consequence G.

INIT F describes the initial state.

UNLESS F extends Unity's UNLESS to the distributed case.

F LEADS_TO_C F', F BECAUSE_C F' express that the consequence has to hold in a state *close* to the state satisfying the premise: F LEADS_TO_C F' requires F' to hold in the same state in which F occurs, or in the next one; F BECAUSE_C F' requires F' to hold in the same state of F or in the previous one.

The ability to deal with events explicitly may enhance the expressivity and simplicity of logical specifications, and actually the interest for event description is growing in the literature [2, 7, 3].

The use of events proves particularly useful in distributed systems. For instance, formula

$\mathsf{M}\,F$ LEADS_TO $\mathsf{N}\,G$

says that formula F holding in component M implies that G must hold in a future state of component N. This means that if F is stable in M, not only G has to be true infinitely often as it would be the case in a non–distributed setting, but also that infinitely many communications must occur between M and N. In the practice, it is convenient to let the specifier state a weaker assertion, namely

$\mathsf{M}\,\Delta F$ LEADS_TO $\mathsf{N}\,G$

where the premise is restricted to the states where the condition is established, and a single communication from M to N is sufficient to establish the consequence.

Finally, the use of a mix of events and conditions as in $\mathsf{M}(\Delta F \wedge G)$, offers a straightforward way to express the premises of event–condition rules.

2.4 Design methodology and tools

Axioms and rules of the logic are presented in Tables 1 and 2. Our design methodology is based on refinements [4], and the refinement relation between two logical theories corresponds to logical deduction.

In the case of Mob$_{adtl}$, each refinement step extends the axiomatic presentation to specify new properties. In particular, the refinement steps of the basic model result in models with a more elaborated structure, with respect to the network topology, the security policies, etc. For instance, a set of refinement steps can lead to the complete specification of the properties of the underlying middleware. Some experiments in this direction, targeted to a specific technology, namely CORBA, are reported in [13]. More refinements can lead to actual programs [17].

MaRK, our proof assistant [9] that partially automates the verification process, is a valuable tool supporting the refinement process, making it feasible to avoid error prone "by hand" arguments.

3 Mob$_{adtl}$: a mini–tutorial

To describe Mob$_{adtl}$, we let A, A_1, A_2, R, $S \ldots$ range over agent names, (we use S and R –sender and the receiver of a message– for readability) and G, G_1, $G_2, \ldots$ range over guardian names. P is a profile, and D specifies the details of an exception.

3.1 Structural properties

Each guardian holds a representation of the neighborhood it controls, encoded via predicate *guarding(A)*, where A is an agent in the neighborhood. The flat topology of the network of neighborhoods is stated by axiom S1 that says that guardians cannot control other guardians

S1: $\bar{\mathsf{G}}_1(\neg guardedby(\mathsf{G}_2))$

Location awareness is encoded in the agents' states via predicate *guardedby(G)*, where G is the name of the guardian of the current neighborhood. In the initial state, every guardian knows the agents in its neighborhood, and every agent knows the name of its guardian, as said by axiom S2

S2: $\forall \mathsf{A}, \mathsf{G}.\ \text{INIT}\, \mathsf{A}(guardedby(G)) \leftrightarrow \mathsf{G}(guarding(A))$

Moreover, we require that agents are always in a neighborhood, axiom S3, and that are not ubiquitous, i.e. they cannot be in more than a neighborhood at the same time, axiom S4

S3: $\exists \mathsf{G}.\mathsf{A}(True) \rightarrow \mathsf{A}(guardedby(G))$
S4: $\bar{\mathsf{A}}((guardedby(G_1) \wedge guardedby(G_2)) \rightarrow G_1 = G_2)$

Note that the formula in axiom S2 is false in any state that is not the initial one because of the particular structure of our models and of the initial state itself. Our logic is such that, apart from the initial state, it is impossible to have any form of global knowledge about the system. This reflects the particular features or our reference setting, i.e. asynchronous communications in the absence of a global clock.

3.2 Communication properties

We use predicate *out(M,P)* to represent the will of an agent to send message *M* to a receiver that satisfies profile *P*. The request can result in a successful delivery or in an exception:

C: $\mathsf{S}(\Delta out(M,P) \wedge guardedby(G))\ \text{LEADS\_TO}$
$\mathsf{R}(\Delta in(M,\mathsf{S})) \vee \mathsf{S}(\Delta exc(msg(M,S,P),D))$

Note that the disjunction is not exclusive: in those settings in which the network is scarcely reliable, the message can reach its destination, and the sender receive a warning exception, saying that locally it is not known what happened to the message. This reflects the possible lack of global knowledge, typical of asynchronous settings with unreliable communication. To exclude the warning, a refinement should foresee that all outgoing messages are logged, and that a ticket is sent back upon reception. Then, the warning is not issued, if the sending guardian receives the confirmation ticket before a given deadline.
Conversely, a message is received only if it was: (i) sent; (ii) processed by the sender guardian; (iii) processed by the receiver guardian; (iv) processed by a guardian resolving the profile. These guardians don't need to be distinct.

C1/A2: $\mathsf{R}(in(M,S)) \ \dot{\cdot}\ \mathsf{S}\Delta out(M,P) \ \wedge$
$\mathsf{G}(msgReq(M,S,P,St) \wedge guarding(S)) \ \wedge$
$\mathsf{G}_1(msgReq(M,S,P,St_1) \wedge guarding(R)) \ \wedge$
$\mathsf{G}_2(msgReq(M,S,P,St_2) \wedge satisfy(\{R\},P))$

The mechanism of profile resolution is defined by a simple theory stating that a profile is specified as a set of constraints, and distinct profiles can have overlapping constraints. If an entity satisfies a profile, it means that it satisfies all the constraints that define the profile. If a set of entities satisfy a profile, it means that all its element entities satisfy the profile.

Exceptions are due to the decision of one of the guardians involved in the delivery:

C2: $\mathsf{S}(exc(msg(M,S,P),D)) \ \dot{\cdot}$
$\mathsf{S}(\Delta out(M,P)) \wedge \mathsf{G}(msgReq(M,S,P,St)) \wedge$
$\mathsf{G}(toBeVetoed(msg(M,S,P),D)))$

A communication request is attributed to an agent only if that agent actually committed to send a message:

Id2: $\mathsf{G}(\Delta msgReq(A,O,P,St) \ \dot{\cdot}\ \mathsf{S}(\Delta out(M,P)))$

As we will see in Section 4, the Responder will be part of a guardian. So, we need to look at some of the reference axioms for communication, which describe the internal working of guardians, and justify the properties given above.

When an agent *S* wants to send a message *M*, it notifies a communication request *msgReq(M,S,P,u)* to its guardian *G*, specifying the profile *P* the receiver must satisfy:

Cc1: $\mathsf{S}(\Delta out(M,P) \wedge guardedby(G)) \ \textsc{leads\_to}\ \mathsf{O}(\Delta msgReq(M,S,P,i))$

The last argument to $msgReq$ models the possible state of a communication request: either *u*, saying that profile *P* is still unresolved, *rec(R)*, where *R* is the name of the receiver, once selected.

The agent request either triggers a communication process that leads to the identification of a suitable recipient, or is immediately rejected for reasons that are not specified in the reference model, but can be specified by refinements (as we will do for the example below):

Gc1: $\mathsf{G}(\Delta msgReq(M,S,P,u))$ LEADS_TO
$\mathsf{G}_1(\Delta msgReq(M,S,P,rec(R))) \vee \mathsf{G}(\Delta toBeVetoed(msg(M,S,P),D))$

If the request is accepted, it is routed until it is resolved (i.e. a receiver is selected). Guardians are in charge of finding a receiver that satisfies the requested profile

Gc1_2': $\mathsf{G}(\Delta msgReq(M,S,P,rec(R)))$ $\ddots$
$\mathsf{G}_1(msgReq(M,S,P,u) \wedge satisfy(R,P))$

and to route the communication request to the guardian of the receiver, that delivers the message.
It can happen that no receiver satisfying *P* exists. Moreover, any of the involved guardians can veto the request because of some policy, see Gc1. In this case an exception is thrown and communicated to the sender. To avoid fake communication requests, message delivery requests must be actually originated by the involved agent

Cc1_2': $\mathsf{G}(\Delta msgReq(M,S,P,u) \wedge guarding(S))$ $\ddots$
$\mathsf{S}(\Delta out(M,P) \wedge guardedby(G))$

3.3 Trusted guardians

As we said, security policies must be explicitly specified through suitable refinements of the general model. To specify an example policy, we have introduced the predicate *trusted(G),* which holds in a guardian state when the neighborhood (guarded by) G fulfills the guardian security requirements [8]. The refinement step strengthens axiom Gc1_2' as follows, and leaves the other axioms unchanged.

Tr: $\mathsf{G}(\Delta msgReq(M,S,P,rec(R)))$ $\ddots$ $\mathsf{G}_1(msgReq(M,S,P,u) \wedge trusted(G) \wedge satisfy(R,P))$

In the next section we will show how to adapt the policy: we will need to find ways to update the information encoded by the *trusted* predicate.

4 Adaptive security: an example

In the first part of this section (4.1), we describe the conceptual components of the ASI of the WOLFASI call, namely Detector, Analyzer, and Responder, as Mob_{adtl} entities. The Detector and the Responder are specified as refinements of the generic

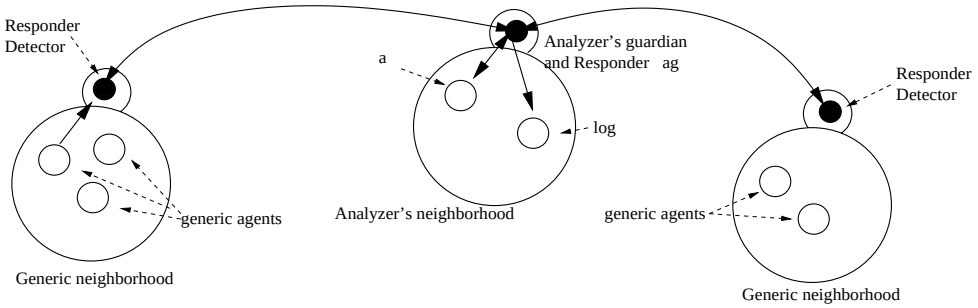


Figure 3: Describing an ASI in the Mob$_{adtl}$ model: Detectors, Analyzer, and Responders.

guardians of Mob$_{adtl}$, while the Analyzer will be specified as an agent, living in a dedicated neighborhood, as illustrated in Figure 3. We add a log component, living in the Analyzer's neighborhood, to the infrastructure. Then, in Section 4.2 we specify how to update the set of trusted neighborhoods, as a response to an action taken by the Analyzer on the base of the information received by the Detectors.

4.1 ASI specification

We will assume just one analyzer, the agent a, in neighborhood ag. Detection is a function performed in each guardian. Responses may require distributed computations involving many guardians, but will originate in the Analyzer's guardian ag.

To allow the detection mechanism to route the security warnings to the analyzer, we introduce the profiling tag sec_w, that identifies a as the target of each warning message:

D: $\bar{\mathsf{G}}(satisfy(\{X\},\{sec_w\}) \leftrightarrow X = a)$

Axiom **D** is a global invariant, expressing the fact that all agents know of the possibility of sending security warnings. However, the profile mechanism decouples the use of the mechanism from its implementation: we could think of scenarios where there are cooperating analyzers, but still triggered by the same messages. Note that security warnings are transported by the normal communication mechanism: they can reach the Analyzer only if there is a path of trusted guardians to do so.

The Responder has two parts: a generic one and one related to the specific policy. The generic part states the relation between a and ag.

R1: $\bar{\mathsf{ag}}\, guarding(a)$

An1: $\bar{\mathsf{a}}\, guardedby(ag)$

Responses are triggered by messages of the Analyzer. The $adapt$ profile tag is used by a to identify messages that carry adaptation suggestions. The responder will intercept all messages tagged $adapt$, and signal the Analyzer that it is taking actions.

The remaining part of the Responder, namely which actions it takes, is specified per policy, according to the following pattern, where F and F' will be side conditions on the request and on the state of the system, and the result of the actions, respectively:

RP: $\mathsf{ag}\ \Delta msgReq(M, a, \{adapt\}, u) \wedge F$ LEADS_TO F'

4.2 Policy adaptation

Policy adaptation has two parts: analysis and action. Analysis, in our simple scenario, boils down to let the Analyzer maintain a $counter$ per each guardian in the system, a $threshold$, and in counting the incoming warning messages:

An2: $\mathsf{a}\ (\Delta in(demote(X, D), S) \wedge counter(X, Y))$ LEADS_TO_C $\mathsf{a}\ (counter(X, Z) \wedge Z = Y + 1)$

To guarantee the correctness of the counter mechanism, we need to refine the Mob_{adtl} communication model by adding an axiom requiring that the messages are processed one at a time.

Buff: $\bar{\mathsf{R}}((\Delta in(M, S) \wedge \Delta in(M_1, S_1)) \rightarrow (M = M_1 \wedge S = S_1))$

Once a counter crosses the threshold, the Analyzer proposes to demote the involved guardian, by issuing an adaptation message:

An3: $\mathsf{a}\ (\Delta counter(X, C) \wedge threshold(M) \wedge C \geq M)$ LEADS_TO $\mathsf{a}\ \Delta out(demote(X, D), \{adapt\})$

The action starts in the Responder ag. We instantiate pattern **RP**, using the side condition in the premise to quantify over all locally trusted guardians. The postcondition permits to deal with events like network failures:

R4: $\mathsf{ag}\ \Delta msgReq(demote(X, D), a, \{adapt\}, St) \wedge trusted(G)$ LEADS_TO $\mathsf{G}\Delta msgReq(demote(X, D), a, \{adapt\}, St_1) \vee \mathsf{G}'toBeVetoed(msg(demote(X, D), a, \{adapt\}), \{unreachable(G)\})$

The next action takes part in all the Responders. We instantiate again pattern **RP**, using an empty side condition:

R5: $\mathsf{G}\Delta msgReq(demote(X, D), a, \{adapt\}, u)$ LEADS_TO $\mathsf{G} \sim trusted(X)$

The next axiom is a global invariant, used to record in the analyzer's log all the actions taken to adapt the policy.

R6: $satisfy(\{X\}, \{adapt\}) \leftrightarrow X = log$

Once this core specification is completed with a few technical axioms, e.g. stating initial conditions, and with a small theory of counters, we can prove the following threshold policy property **TP**, which, if generalized, would characterize the policy itself. It states that two warnings against a guardian may succeed in banning it, if the threshold is two:

TP: $\mathsf{a}\, threshold(2) \wedge \mathsf{ag}\, trusted(G) \wedge$
$\mathsf{C}_1 \Delta out(demote(X, D), \{sec_w\}) \wedge \mathsf{C}_2 \Delta out(demote(X, D'), \{sec_w\}) \wedge C_1 \neq C_2$
LEADS_TO $\mathsf{G} \sim trusted(X) \vee$
$\mathsf{a}\, exc(msg(demote(X, D), a, \{adapt\}), \{unreachable(G)\}) \vee$
$\mathsf{C}_1 exc(msg(demote(X, D), C_1, \{sec_w\}), \{unreachable(a)\}) \vee$
$\mathsf{C}_2 exc(msg(demote(X, D'), C_2, \{sec_w\}), \{unreachable(a)\})$

4.3 Discussion

The major problem when reasoning about distributed global systems, is to foresee, during specification, all the possible evolutions of the system, due to the fully asynchronous setting. Formal reasoning permits to highlight the wrong assumptions that can easily be made, when stating the properties a system should satisfy. For instance, the proof of property **TP** requires that the threshold is invariant. In an approach to adaptation, this is clearly a very strong assumption, one that we would avoid. However, there is a real difficulty here, which impacts on the locality/globality of the properties we can state. **TP** is a global property that relates the originators of the security warnings and the analyzer: the "synchronization" of the consequences of two events in the originators with the threshold in the analyzer can be guaranteed only if the threshold is fixed. Otherwise, when the warnings reach the Analyzer, the threshold could be higher, in a typical race, and the property cannot be stated. A property similar to **TP** can be derived, with weaker assumptions (we can drop threshold invariance), at the price of reasoning locally in the analyzer. We only need that the incoming warnings are stable. This is a technical assumption, that can be interpreted as the request that there is a log of the incoming security warnings. The property says that an action against X is attempted either when the threshold is 2 and the second warning against X is logged, or when the threshold goes down to 2, and there are at least two warnings logged against X:

LTP: $\mathsf{ag}\, trusted(G) \wedge C_1 \neq C_2 \wedge ($
$\mathsf{a}\, (threshold(2) \wedge \Delta(in(demote(X, D), C_1) \wedge in(demote(X, D'), C_2))) \vee$
$\mathsf{a}\, (\Delta threshold(2) \wedge in(demote(X, D), C_1) \wedge in(demote(X, D'), C_2))$
LEADS_TO
$\mathsf{G} \sim trusted(X) \vee \mathsf{a}\, exc(msg(demote(X, D), a, \{adapt\}), \{unreachable(G)\})$

An even more local property can refer explicitly to a counter holding the number of logged messages regarding X. This formulation could be easily parameterized with respect to the value of the threshold.

Looking at the policy itself, either property shows that our scenario is prone to attacks: a sufficiently large coalition of agents could succeed in banning any chosen guardian, even ag itself. This last event can be avoided by disregarding warnings that refer to ag, i.e. not counting these warnings:

SR: $\bar{\mathsf{ag}}\, counter(X, Y) \rightarrow X \neq ag$

More general protection could be gained by more complex behaviors of the Analyzer, e.g. rating also how much an agent sending a security warning can be trusted.

Finally, our scenario could be generalized to have distributed analyzers. If a common policy is required, distributed consensus algorithms should be used: a formalization in our logic is given in [12].

5 Related work

We are not aware of any ASI formulation that permits to reason on the global properties of a fully asynchronous system. On the other side, asynchronous communication is the most used abstraction when modeling global applications, and recent initiatives in the USA and in Europe, like that on Global Computing in the IST/FET program, well represent the relevance of this area. Particularly timely is the interest for foundational work to define theories that can support and underlie the development of Global Computing [11].

A major benefit of $\Delta DSTL(x)$ is that the exploitation of the local theories in the proofs of the distributed properties becomes smooth and robust. An interesting development would be the integration in the local theories of policy specifications à la Halpern and Weissman [10]. A further example of refinement is predicate *trusted*, which could be better detailed, to capture the reasons for trustworthiness or to carry a "policy compliance value" as in the KeyNote Trust–management System [5].

Acknowledgments

We would like to thank Simone Semprini for his work with the theorem prover MaRK [9].

The work is partly supported by the DEGAS (Design Environments for Global ApplicationS) IST-2001-32072 project funded by the FET Proactive Initiative on Global Computing.

References

[1] Workshop on logical foundations of an adaptive security infrastructure., July 12-13 2004. Turku, Finland, http://www.aero.org/wolfasi.

[2] L. Andrade and J.L. Fiadeiro. Coordination primitives for event-based systems. In *In Proceedings of the* 1^{st} *Int. Workshop on Distributed Event-Based Systems (DEBS'02)*, 2002.

[3] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The forspec temporal logic: A new temporal property-specification language. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211, Grenoble, 2002. Springer-Verlag.

[4] R.J.R. Back and J. von Wright. *Refinement Calculus. A Systematic Introduction*. Graduate texts in computer science. Springer-Verlag, 1998.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D.Keromytis. The keynote trust-management system. www.cis.upenn.edu/$\sim$ keynote/.

[6] T. Costello and A. Patterson. Quantifiers and operations on modalities and contexts. In A.G. Cohn, L. Schubert, and S.C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 270–281. Morgan Kaufmann, San Francisco, 1998.

[7] M.S. Dias and D.J. Richardson. The role of event description in architecting dependable systems. In 1^{st} *Workshop on Architecting Dependable Systems (WADS'02)*, Orlando, May 2002.

[8] G. Ferrari, C. Montangero, L. Semini, and S. Semprini. Multiple Security Policies in Mob_{adtl}. In P. Degano, editor, *Proc. Workshop on Issues in the Theory of Security (WITS'00)*, Geneva, 7,8 July 2000.

[9] G. Ferrari, C. Montangero, L. Semini, and S. Semprini. Mark, a reasoning kit for mobility. *Automated Software Engineering*, 9(2):137–150, Apr 2002.

[10] J.Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003)*, pages 187–201, Pacific Grove, CA, USA, 2003. IEEE Computer Society.

[11] R. Milner. Theories for the global ubiquitous computer. In I.Walukiewicz, editor, *7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 5–11, Barcelona, Spain, 2004. Springer. Invited Contribution.

[12] C. Montangero and L. Semini. Distributed states temporal logic. The Computing Research Repository (CoRR): cs.LO/0304046, 2003.

[13] C. Montangero and L. Semini. Software specification and design: from formal methods to standard middleware. In *Proc. 6th ERCIM Inter. Workshop on Formal Methods for Industrial Critical Systems (FMICS'01)*, 2001.

[14] C. Montangero and L. Semini. Distributed states logic. In 9^{th} *International Symposium on Temporal Representation and Reasoning (TIME'02)*, Manchester, UK, July 2002. IEEE CS Press.

[15] C. Montangero, L. Semini, and S. Semprini. Logic Based Coordination for Event–Driven Self–Healing Distributed Systems. In R.De Nicola, G.Ferrari, and G. Meredith, editors, *Proc. 6th Int. Conf. on Coordination Models and Languages, COORDINATION'04*, volume 2949 of *Lecture Notes in Computer Science*, pages 248–262, Pisa, Italy, Feb. 2004. Springer-Verlag.

[16] L. Paulson and T. Nipkow. Isabelle. www.cl.cam.ac.uk/Research/HVG/ Isabelle/.

[17] L. Semini and C. Montangero. A Refinement Calculus for Tuple Spaces. *Science of Computer Programming*, 34:79–140, 1999.

[18] S. Semprini. *Specification and verification of mobile systems*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2004.

Towards an Algebraic Approach to Solve Policy Conflicts*

Cataldo Basile Antonio Lioy
Politecnico di Torino
Dip. Automatica ed Informatica
Torino - Italy

17th June 2004

Abstract

Policy-based security is one of the most innovative area in the security arena. A policy represents the high level targets and is applied by using sets of rules. A rule consists in a set of conditions expressing the domain of application and a set of actions that must be performed when conditions are met. One of the major problems is the conflict management, that is the decision of the action to be executed when more than one rule applies.

In this paper we present a formal definition of policy, policy rules (in if-conditions-then-actions format) and policy conflicts and we use semi-lattices to solve inconsistencies. These algebraic structures are helpful to convey information about the actions to enforce when conflicts occur as well as the importance or the severity of the actions. We also extend the semi-lattice based approach to `AND`*-ed and* `OR`*-ed sets of actions.*

Keywords: *Security Policy; Policy Conflicts.*

1 Introduction

Policy-based methodologies are one of the most effective approaches to reduce the management costs because they permit the definition of general high-level targets without the need of a detailed specification of the environment where the policy will be applied. Since the policy drives the behaviour of each system component, the presence of inconsistencies may lead the system to unknown states or errors. Another large problem area is related to rule creation. Quite often, the rules for

*This work is part of POSITIF project, funded by the EC under contract IST-2002-002314

access control devices, are written by many administrators at different times and in many cases without a clear idea of who wrote what, when the rules were written and for what purposes. The problem of inconsistencies is particularly important for systems implementing adaptive security. In fact, to maintain the system in a well defined state, the reactive part could autonomously change various parameters or perform actions whose coherence with the policy must always be guaranteed.

To solve these problems, we propose an analysis of the policy rules actions to identify and resolve conflicts in a centralized manner. Our solution relies on two ideas: conditions can be treated with a *set-based* approach, and *semi-lattices* are algebraic structures useful for rule selection in conflict resolution. The use of semi-lattices is justified because we found out a necessary and sufficient condition that a set of actions must respect in order to be used for conflict resolutions. A semi-lattice may be viewed as a partially ordered set and it is possible to represent it as a *graph*, and this may give a great impact because it enables the inheritance of all the mathematical results and algorithms from graph theory. In addition, our method is independent of the target syntaxes and of the layer at which the rules are defined. In fact, we abstract the conflict resolution as the selection of a new action to enforce when two or more rules conflict, and this process is independent of the details that characterize a specific context.

Natural application scenarios for this theory are filtering rules where the applicable actions are a priori known and their number is finite or an ordering relation between actions is defined, e.g. IPsec or SSL/TLS rules [1, 2], group key management [3] policies and all the contexts where an autonomous or semi-autonomous decision mechanism must be implemented.

This paper is organized as follows: in section 2 we briefly sketch the general background of the field by referencing other works on policy-based security; in section 3 we define policy terminology by using mathematical notation; in section 4 we fix the concept of policy conflict and we present an example of policy rules that enforce only one action; in sections 5 and 6 we extend our analysis to the sets of actions connected with logical operations; finally, in section 7 we briefly draw conclusions and give some hints for future works.

2 Background and contributions

The process of selecting a new action when conflicts may arise in security is an interesting field of investigation. In this area, semiring-based approaches are used [5] to define constraints and to build an infrastructure to negotiate access control policies [6]. Other works present an algebra to compose access control policies [7], the hierarchy of policy [8], the conflicts that may happen in their definition [9], and

in a recent paper [10] event calculus is used to apply abductive reasoning to analyse policy and resolve conflicts. Another, more general, field of research use logic to solve conflicts in many areas, from databases coherence and logic programming, to applications of artificial intelligence such as knowledge assimilation, default reasoning, high-level goal-reduction, truth maintenance, retractability and so on [11].

The Policy Core Information Model (PCIM) [12] is an object-oriented information model for representing policy information as extensions to the Common Information Model (CIM) activity within the Distributed Management Task Force (DMTF). The RFC-3198 is a glossary of policy-related terms that defines the concepts and eliminates the inconsistencies in writing documents.

The definition of policy and policy rule presented in PCIM and its extension shown in RFC-3198 gave us the starting point to refine these concepts in a way useful for a formal approach. The *policy* is defined in two ways, "a definite goal" and "a set of rules to administer, manage, and control access to network resources"; furthermore the "policy is applied using a set of policy rules". The policy rules or simply the *rules* are described as "the binding of a set of actions to a set of conditions - where the conditions are evaluated to determine whether the actions are performed". And a *policy conflict* "occurs when the actions of two rules (that are both satisfied simultaneously) contradict each other" and it creates a problem because "the entity implementing the policy would not be able to determine which action to perform". Furthermore, the same authors state that "the implementers of policy systems must provide conflict detection and avoidance or resolution mechanisms to prevent this situation".

The purpose of this paper is not to propose another declarative language but it wants to offer formal studies and requirements to make conflicts automatically resolvable. Note that our approach is different from the cited works because we concentrate on the security area, in particular on rules that are directly derived from access control, and we try to give a solution to conflicts in an algebraic manner. Another important aspect that is missed in other works is the mathematical definition of this kind of policy rules extracted from security area. These rules are used in many applications such as firewalls, screening routers, gateways, but no formal treatment nor mathematical interpretation is given. We expect that this paper stimulates further work in this area because we try to introduce concepts and terms not yet defined in RFC-3198 and we open the way to the formalization based on sets and graph theory as well as algebra and lattice theory. Even if we start from a narrow domain, the formalism and the abstraction make straightforward the extension to every case with a finite number of actions an order relation on the set of actions.

3 Formal definition of policy rules

In this section we start from definitions and interpretations of the policy terminology to formalize them. For this purpose we will present the necessary background required for a mathematical model of the policy conflicts. From the application of logical operations on the set of condition we show that these can be mapped on union, difference and intersection of sets or cartesian products. By taking advantage of the equivalence of boolean algebras and the algebra of the subsets it is possible to apply logical or set operations in indifferent manner. The merging of these two worlds is a relevant result; in fact, it can give improvements in computational aspects since implementations may use the most convenient one. Every argument is dealt in two steps: firstly practical cases extracted from real world applications are introduced in order to illustrate basic ideas and their connections to computer science security; then a formal modeling of problems and concepts is presented in an axiomatic way.

Our treatment starts with some examples of rules in "if-*condition*-then-*action*" format:

if `Source_IP_Address` $== 192.168.3.34$ **then** `permit connection`
if `TCP_Destination_Port` > 1023 **then** `deny connection`

The first rule states that the connections from $192.168.3.34$ are allowed and the second rule denies connections to non privileged ports.

To comply with RFC-3198, we define the first model of a policy rule:

$$\{C_i\}_{i\in I} \rightarrow \{A_j\}_{j\in J} \tag{1}$$

in which $\{C_i\}_{i\in I}$ is a family of conditions indexed by i (that ranges over the elements of the indexing set I) and $\{A_j\}_{j\in J}$ is a family of actions (indexed by $j \in J$). It is worth noting that the symbol "$\rightarrow$" in formula (1) has no logical meaning and it is only the way to represent the concept of "binding" of the RFC-3198.

Conditions express limits on specific fields and are often collected in tables having in the same column conditions related to the same category. In fact, in a more restricted yet concrete view, conditions cannot express abstract constraints, but it is always possible to find a more general category, a wider set, from which they select the piece to consider. Last ideas are summarized in table 1 by using a simplified abstract language [14, 15]; the categories are in this case `SOURCE_PORT`, `IP_SOURCE_ADDR`, `IP_DEST_ADDR`, `DEST_PORT` in IP and TCP packets. R_1 states that from the subnet $192.168.1.0/24$ it is possible to use HTTP services, R_2 blocks connections from the same subnet between non privileged ports and R_3 denies telnet connection from the subnet $192.168.3/24$ to the subnet $192.168.0.0/16$. The symbol "*" means that any value is admitted.

	IP_SOURCE_ADDR	SOURCE_PORT	IP_DEST_ADDR	DEST_PORT	ACTIONS
R_1	192.168.1.*	*	*	80	ALLOW
R_2	192.168.1.*	> 1024	*	> 1024	DENY
R_3	192.168.3.*	*	192.168.*	23	DENY
R_4	192.168.*	*	*	$\neg 80$	DENY

Table 1: Example of filter rules

An important characteristic of these categories is that they are non overlapping, even if they may consist of values of the same type. For example, in the field IP_SOURCE_ADDR or IP_DEST_ADDR the conditions regard IP addresses but they impose restrictions on different portions of the IP packet. We call the different non overlapping categories *selectors*. Each selector defines the set of the acceptable values i.e. it must be of a specific *selector type*, for example integers, real numbers, strings, regular expressions or user defined types.

The evaluation of a condition consists in controlling that the value of a monitored parameter belongs to the ranges imposed by each condition. For example, the condition on the IP source address in R_1 of table 1 is evaluated to true if an IP packet comes from the $192.168.1.123$ but is evaluated to false if an IP packet comes from $192.168.4.12$. This consideration enables us to state that every condition can be viewed, via set interpretation, as a *subset* of a specific selector.

It is often useful to impose conditions by specifying what is not acceptable. It is the case of *negative conditions* like the rule R_4 in table 1 that permits just HTTP connections. The symbol $\neg$ indicates that is a negative condition.

We detail the simple case of conditions belonging to the same selector, next the more complex case of conditions that are not related to the same selector.

3.1 Conditions that belong to the same selector

We need to formalize the above ideas in order to map of policy concepts to the set-based framework. We assume in this section that a selector is a *connected* set [17], i.e. it is not composed by more, logically separated, portions.

In order to define standard terminology the concept of *condition* is introduced then it is refined in *basic* and *compound condition*.

Definition 1 (condition) *A* condition c*, in a given selector* S*, is a subset* c *of* S*. A condition is* basic *if it is representable as a connected subset*[1] *in a given selector,* compound *otherwise.*

[1] When the selector is endowed with an order relation, connected subsets correspond to intervals [17].

A condition c belongs to a selector S if $c \subseteq S$. All the compound conditions can be expressed as the union of non overlapping basic conditions.

Let us introduce a function to express that a condition is satisfied.

Definition 2 (Valuation function) *The* valuation function *associated to the condition $c \subseteq S$ is the function:*

$$\chi_c : \begin{array}{ccc} S & \longrightarrow & \{0,1\} \\ x & \longmapsto & \begin{cases} 1 & \textit{if } x \in c \\ 0 & \textit{if } x \notin c \end{cases} \end{array}$$

The function χ_c is the characteristic function associated to the subset c of S. It assigns the value 1 (i.e. `TRUE`) if an element of the selector belongs to c, 0 (i.e. `FALSE`) otherwise. A condition c is satisfied (in S) for a given value x if $\chi_c(x) = 1$; a condition is a tautology if $c = S$ or alternatively if $\chi_c = 1$ for each $x \in S$. An example of *tautology* is the "*" symbol of previous tables. In the opposite case, a condition always false $(c = \emptyset)$ is *impossible* or *absurd*.

Negative conditions may be expressed as subsets of the selector. The "negation" is a unary operation and there is a precise connection between a condition and its negated form: in classical logic, where a condition is satisfied, it is not satisfied its negation. For this reason it is possible to map the negation to set minus operation.

Definition 3 (negative condition) *The negation $\neg c$ of a condition c belonging to S, is the condition $S \setminus c$.*

The valuation function associated to $\neg c$ is $\chi_{\neg c} = \chi_{S \setminus c}$.

The set of all the subsets of a given set S is a boolean algebra with respect to the union and intersection operation, the empty set and S [18]. This property may be used to study the behaviour of conditions joined with `AND` or `OR`. In fact, due to this equivalence, $c_1 \wedge c_2 = c_1 \cap c_2$ and $c_1 \vee c_2 = c_1 \cup c_2$ hold. In terms of valuation functions $\chi_{c_1 \wedge c_2} = \chi_{c_1} \cdot \chi_{c_2}$ and $\chi_{c_1 \vee c_2} = \chi_{c_1} + \chi_{c_1} - \chi_{c_1 \cap c_2} = \chi_{c_1} + \chi_{c_1} - \chi_{c_1} \cdot \chi_{c_2} = \chi_{c_1 \cup c_2}$ are valid. Due to the associativity of boolean operations, these results can be extended to more than two conditions. Two conditions may not intersect, in this case their `AND` is impossible, and the union of two conditions may cover the whole selector, that is their `OR` is tautological. Logical expressions containing `AND` and `OR` can be always viewed as a unique condition, since union, intersection and difference of subsets of the same set are another subset, that is, a condition.

3.2 Conditions that belong to different selectors

In this section we complete the mapping of policy related concepts to set-based terminology. Starting from formula (1), we study a more complex example of conditions belonging to different selectors joined by logical AND or OR operations. These expressions can be written in the *Conjunctive Normal Form* $\bigwedge\bigvee c_{ij}$ (CNF), or in *Disjunctive Normal Form* $\bigvee\bigwedge c_{ij}$ (DNF) [13]. We prefer the DNF:

$$\{C_i\}_{i\in I} = \bigvee_{o\in O}\bigwedge_{a\in A} c_{oa} = (c_{11}\wedge\cdots\wedge c_{1n})\vee(c_{21}\wedge\cdots\wedge c_{2n})\vee\cdots\vee(c_{m1}\wedge\cdots\wedge c_{mn})$$

We firstly concentrate on policy rules of the form:

$$\bigwedge_{a\in A} c_a \rightarrow \{A_j\}_{j\in J} \tag{2}$$

where $c_a \subseteq S_a$ and S_a is the a-th selector. Conditions belonging to the same selector can be collapsed to one condition with the intersection operation and we assume that all the c_a belong to different selectors, that is $S_x \neq S_y$ if $x \neq y$.

If two conditions $c_1 \subseteq S_1$ and $c_2 \subseteq S_2$ are joined with AND operation, it is impossible to derive a unique condition, because selectors are non overlapping. To satisfy $c_1 \wedge c_2$ it is needed that c_1 is satisfied in S_1 ($\chi_{c_1\subseteq S_1} = 1$) and c_2 is satisfied in S_2 ($\chi_{c_2\subseteq S_2} = 1$). In this way the space where the conditions are evaluated is extended to the cartesian product of both selectors $S_1 \times S_2$ and the complex condition corresponds to the region $c_1 \times c_2 \subseteq S_1 \times S_2$. The valuation function $\chi_{c_1\wedge c_1}$ is the characteristic function of $c_1 \times c_2$ in $S_1 \times S_2$.

It is possible to show that, by fixing the order of selectors, formula 2 is equivalent to:

$$C \rightarrow \{A_j\}_{j\in J} \qquad C \subseteq \mathcal{S} = S_1 \times S_2 \times \cdots \times S_n \tag{3}$$

where n represents the number of selectors. The elements C and $\mathcal{S}$ in formula (3) are called respectively the *selection condition* of the policy rule and the *policy rules domain*. We call *region* or *area* any subset of the domain $\mathcal{S}$. The definitions introduced for conditions/selectors can be extended to selection conditions/policy rules domain:

Definition 4 (selection condition) *A selection condition in* $\mathcal{S} = S_1\times S_2\times\cdots\times S_n$, *is the cartesian product* $C = c_1\times c_2\times\cdots\times c_n$ *of conditions* $c_1 \subseteq S_1, \cdots, c_n \subseteq S_n$. *A selection condition is said* basic *if each* c_i *is basic,* compound *otherwise.*

Since the selection condition of a rule is obtained as cartesian product of unions of intervals, it can be considered as the *union of rectangles* if the number of different selectors $n = 2$, the *union of parallelepiped* if $n = 3$ and the *union of hyper-rectangles* if $n > 3$ (see figure 1(a) for case $n = 2$).

Even in this case the valuation function associated to a selection condition $C \subseteq \mathcal{S}$ is the characteristic function of C in $\mathcal{S}$

The policy rules domain $\mathcal{S}$ is a set, and it forms a boolean algebra with respect to the union and intersection operations, the empty set, and $\mathcal{S}$ itself So, the mapping of `OR` to union ($\vee \Leftrightarrow \cup$) and of `AND` to intersection ($\wedge \Leftrightarrow \cap$) hold. For this reason:

$$\{C_i\}_{i\in I} = \bigvee_{o\in O} \bigwedge_{a\in A} c_{oa} = \bigvee_{o\in O} C_o = \bigcup_{o\in O} C_o \qquad \text{with } C_o \subseteq \mathcal{S}$$

Even in this case, the negation of selection conditions can be obtained by using set minus operation.

Definition 5 (negative selection condition) *The* negation $\neg C$ *of a selection condition $C \subseteq \mathcal{S}$ is the subset $\mathcal{S} \setminus C$.*

While the negation of a condition is another condition, it is not generally true that the negation of a selection condition is another selection condition.

3.3 Policy rules set and policy function

We formalize in this section the usual fact that a policy is expressed with rules and the rules are generally collected in tables and add to our study the actions. We may consider that the actions that can be applied are well known and organized in a set $\mathcal{A}$, that we call *actions set*.

When a rule $C \rightarrow \{A_j\}_{j\in J}$ is stated, the concept of "bind" is well depicted with a function. In fact, a rule defines the mapping between the selection condition and the power set of $\mathcal{A}$ (i.e. the set of all the subsets of $\mathcal{A}$, normally represented by $2^{\mathcal{A}}$). To completely define the function in all the policy rules domain a dummy action can be introduced, the *undefined action* A_u. The actions set is extended to $\mathcal{A} \cup \{A_u\}$. We assume that A_u cannot be used in any policy rule, i.e. $\{A_j\}_{j\in J} \subseteq 2^{\mathcal{A}}$.

Definition 6 (rule function) *The rule function associated to the rule $C \rightarrow \{A_j\}_{j\in J}$ is a function*

$$\begin{array}{rccl} \mathrm{r}: & \mathcal{S} & \longrightarrow & 2^{\mathcal{A}} \cup \{A_u\} \\ & x & \longmapsto & \begin{cases} \{A_j\}_{j\in J} & \textit{if } x \in C \\ \{A_u\} & \textit{otherwise} \end{cases} \end{array}$$

A compound selection condition $C = \bigvee_{o=1}^{n} C_o$ can be decomposed in its basic

parts. The corresponding rule function can be rewritten as

$$x \longmapsto \begin{cases} \{A_j\}_{j\in J} & \text{if } x \in C_1 \\ \cdots & \\ \{A_j\}_{j\in J} & \text{if } x \in C_n \\ \{A_u\} & \text{otherwise} \end{cases}$$

It is clear that the same result can be obtained by superposing n rules $C_1 \rightarrow \{A_j\}_{j\in J}$, $\cdots$, $C_n \rightarrow \{A_j\}_{j\in J}$. The OR symbol can be therefore extracted from condition clause in order to obtain a set of rules having the same action clause. By extending this approach, we formally define the policy rules sets.

Definition 7 (policy rules set) *A* set of policy rules *is defined as:*

$$\bigvee_{k\in K} \left(\{c_i\}_{i\in I_k} \rightarrow \{A_j\}_{j\in J_k}\right) = \bigvee_{k\in K} \left(C_k \rightarrow \{A_j\}_{j\in J_k}\right) \tag{4}$$

that is the OR *of many rules (1).*

The k variable defines the indexing sets I_k and J_k or C_k in case of selection conditions.

Furthermore, it is possible to consider that every database or collection of rules determines a relation between the policy rules domain and the power set of $\mathcal{A}$.

Definition 8 (policy function) *A policy defined by a set of rules*

$$\bigvee_{k\in K} \left(C_k \rightarrow \{A_j\}_{j\in J_k}\right)$$

is a function:

$$p: \quad \mathcal{S} \longrightarrow \quad 2^{\mathcal{A}} \cup \{A_u\}$$

$$x \longmapsto \begin{cases} \{A_{j_1}\}_{j_1\in J_1} & \textit{if } x \in C_1 \\ \cdots & \\ \{A_{j_s}\}_{j_s\in J_s} & \textit{if } x \in C_s \\ \{A_u\} & \textit{otherwise} \end{cases}$$

In practice, a Policy Analyzer should return a decision by using previous formula. The assumption made here is that a policy is a function i.e. there is no conflict between rules. An example for the case $n = 2$ of policy function is shown in figure 1(b).

	IP_SOURCE ADDR	SOURCE PORT	IP_DEST ADDR	DEST PORT	ACTIONS
R_1	192.168.1.* $\vee$ 192.168.2.*	*	*	23 $\vee$ 21	DENY

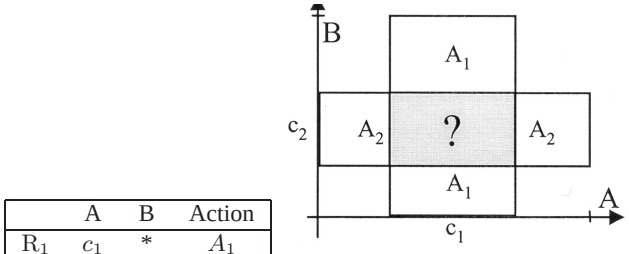
	IP_SOURCE ADDR	SOURCE PORT	IP_DEST ADDR	DEST PORT	ACTIONS
R_{11}	192.168.1.*	*	*	23	DENY
R_{12}	192.168.1.*	*	*	21	DENY
R_{13}	192.168.2.*	*	*	23	DENY
R_{14}	192.168.2.*	*	*	21	DENY

Table 2: Example of database including OR and its expanded form

Consequently, by applying AND and OR operation to a policy rule we may obtain a policy rules set whose rules have basic selection conditions:

$$\{C_i\}_{i\in I} \rightarrow \{A_j\}_{j\in J} = \left(\bigvee_{o\in O}\left(\bigwedge_{a\in A} c_{oa}\right)\right) \rightarrow \{A_j\}_{j\in J} = \underbrace{\bigvee_{o\in O} \underbrace{\left(C_o \rightarrow \{A_j\}_{j\in J}\right)}_{\text{formula (1)}}}_{\text{formula (4)}}$$

An example including the OR in the conditions and the corresponding expanded database is shown in table 2.



	A	B	Action
R_1	c_1	*	A_1
R_2	*	c_2	A_2

Figure 2: Simple representation of two conflicting rules

4 Formal definition of policy composition

The actual complexity of conflicts detection originates from the lack of formal treatments that limits current practices to the observation of the rules simultaneously activated. A systematic analysis of security policies is usually not foreseen.

In this section we concentrate on the formalization of the interactions between policy rules to understand the meaning of the conflicts and the methods to resolve them.

The first step is the precise definition of what a policy conflict is by using the notation introduced in section 3.

Definition 9 (Policy conflict) *Given two policy rules*

$$\mathrm{R}_1 = \left(C_1 \to \{A_{j_1}\}_{j_1 \in J_1}\right) \quad \mathrm{R}_2 = \left(C_2 \to \{A_{j_2}\}_{j_2 \in J_2}\right)$$

a policy conflict occurs if:

$$C_\cap = C_1 \cap C_2 \neq \varnothing \quad \text{and} \quad \{A_{j_1}\}_{j_1 \in J_1} \neq \{A_{j_2}\}_{j_2 \in J_2}$$

In other words, the policy conflict happens if the selection conditions of the two rules intersect but they do not specify the same set of actions. The only region where errors originate is $C_\cap$. Indeed, in the regions out of the intersection there are no conflicts because it is clear which actions to apply (see figure 2). In $C_\cap$ it is impossible to define a policy function since for each element of the domain it is needed the association to only one element of the codomain.

An example is useful to explain a particular case that may seem in contrast with our model. Given two policy rules, the first rule is activated when the condition c_1 is satisfied on attribute A, and the second is activated when the condition c_2 is satisfied on attribute B. It may seem that the rules have no intersection but, due to the independence of attributes A and B, it is possible that both rules are satisfied at the same time. To solve this apparent contradiction it is necessary to think that if a rule does not foresee limits for a given selector, a condition always true must be implicitly assumed. Figure 2 illustrates the problem: R_1 and R_2 are in conflict according to definition (9).

Different methods for conflict resolution were presented [16]: the *prioritization* of the rules based on the order in the database, *Deny Take Precedence* (DTP), if the actions or policy rule more restrictive are preferred and Most/Least Specific Take Precedence (MSTP/LSTP), if the policy rule with the most/least specific condition is preferred. We divide them in two categories: methods that solve conflict by using only the action clause of the policy rule and methods that use the entire rule. In this paper we focus only on the first category, i.e. we formalize resolution strategies based only on the value of the actions imposed by the rules because for these methods we can give a complete theory.

Next, we concentrate on the region $C_\cap$ and choose the resolution methods based only on the action values. In that area, the resulting rule will impose a new

action in function of the conflicting actions:

$$C_{\cap} \rightarrow f\left(\{A_{j_1}\}_{j_1 \in J_1}, \{A_{j_2}\}_{j_2 \in J_2}\right)$$

We start from a simple case to extend these results to more general instances.

4.1 Simple case: policy rules with action clause composed by one action

Let us focus on conflicting rules in which the action clause is composed by only one action, for example $\mathrm{R}_1 = (C_1 \rightarrow A_1)$ and $\mathrm{R}_2 = (C_2 \rightarrow A_2)$. Therefore, a new policy rule is needed for $C_{\cap}$, that is $\mathrm{R}_{\cap} = (C_{\cap} \rightarrow f(A_1, A_2))$. We introduce an abstract operation "$\circ$" in the actions set $\mathcal{A}$:

$$\circ : \mathcal{A} \times \mathcal{A} \longrightarrow \mathcal{A}$$

that must represent the act of choosing the action to be applied when two policy rules conflict. With this operation $\mathrm{R}_{\cap}$ becomes $(C_{\cap} \rightarrow A_1 \circ A_2)$. Let us show by an example the application of "$\circ$" operation. If a rule requires confidentiality for a VPN channel by using $RC4$ and it is in conflict with another rule requiring confidentiality by using DES, one of the possible solutions of this inconsistency may be $RC4 \circ DES = 3DES$.

We work now to define an algebraic structure on the set $\mathcal{A}$ endowed with the "$\circ$" operation represented as $(\mathcal{A}, \circ)$. The first property required from an operation describing the interactions between rules is *associativity*:

$$\forall a, b, c \in \mathcal{A} \quad a \circ (b \circ c) = (a \circ b) \circ c \qquad (\mathrm{SL}_1)$$

This characteristic reflects that the composition is done from a static point of view: rules that conflicts in a particular area are identified and the action to be applied is selected by using only the value taken by the rules in this area. Another important property is *commutativity*:

$$\forall a, b \in \mathcal{A} \quad a \circ b = b \circ a \qquad (\mathrm{SL}_2)$$

In general, commutativity does not hold for any conflict resolution mechanism. For example, in the filtering database of routers, the action selected depends on the order. If two rules $\mathrm{R}_1 = (C_1 \rightarrow A_1)$ and $\mathrm{R}_2 = (C_2 \rightarrow A_2)$ appear in different order the result is different; in fact, if R_1 is found before R_2, then the action applied is A_1 and, if R_2 is found before R_1, then the action applied is A_2. It may seem that $A_1 \circ A_2 \neq A_2 \circ A_1$ and that commutativity must not be required. But this kind of resolution processes do not base their decision only on the information contained

in the policy rules. Indeed, they make use of an external property, the order of the database, to select the action to perform. For the resolution strategies based only on the action clause, the commutativity is adequate in every case. For completeness, we need to define the composition of an action with itself. We impose to the "$\circ$" operation the *idempotence*:

$$\forall a \in \mathcal{A} \quad a \circ a = a \tag{SL$_3$}$$

The decision of imposing the idempotence is due to the meaning of the "$a \circ a$" operation: this operation corresponds to the case of composition of rules that do not conflict or the composition of a policy rule with itself.

In conclusion, our structure must respect the axioms SL_1, SL_2, SL_3, that characterize a well known algebraic structure: a *semi-lattice* [20]. It is important to highlights that these properties are needed even when the action clause is a set.

We can conclude that *every actions set used in policy description, must be at least in the form of semi-lattice, if it should support "conflict detection and avoidance or resolution mechanisms" based on the value of the action clause*. This condition is sufficient because, if the actions set is a semi-lattice, it is always possible to select a new action to apply (by using the "$\circ$" operation); it is necessary because, to automatically select a new action for each conflict that may arise, the composition of every couple of actions must be known.

4.2 The semi-lattice

A semi-lattice can be also viewed as a set provided of a specific order relation. In this section we briefly sketch main definitions and results to introduce in our treatment the connection to graph theory [19].

A *partially ordered set* is a system consisting of a set S and a relation "$\leq$" that is *reflexive* ($\forall a \ a \leq a$), *anti-symmetric* (If $a \leq b$ and $b \leq a$ then $a = b$) and *transitive* (If $a \leq b$ and $b \leq c$, then $a \leq c$). An element $u \in S$ is an *upper bound* for a subset A of a partially ordered set S, if $a \leq u$, for all $a \in A$. An element $u \in S$ is a *least upper bound* (lub) if it is an upper bound and, for all the upper bounds v of A, the relation $u \leq v$ holds. Two elements $a, b \in \mathcal{S}$ are called *comparable* if $a \leq b$ or $b \leq a$ holds. In general, not all the elements are comparable.

Definition 10 (finite semi-lattice) *A* finite (and thus complete) semi-lattice (structure) $\mathcal{S}$*, is a partially ordered set that includes a finite number of elements, in which any subset* $A \subseteq \mathcal{S}$*, have a least upper bound in* $\mathcal{S}$*.*

Another important property of finite semi-lattice is the presence of an element greater than all the others usually called *maximum* or *top element*. The calculation

of the least upper bound is directly defined for more than two actions and this property is important because it permits to resolve conflicts in regions where more than two rules intersect.

A common representation of ordered sets is *the cover graph* and it is shown in figure 3. An element a *is a cover of* b if and only if $a \leq b$, and no element u exists such that $a \leq u \leq b$. The cover graph is a directed acyclic graph where the edges indicate the cover relation, and the nodes are the elements of the set.

By defining the "$\leq$" operation on the actions set $(\mathcal{A}, \circ)$ with the following binary relation:

$$\leq \subseteq \mathcal{A} \times \mathcal{A} \qquad a_1 \leq a_2 \Longleftrightarrow a_1 \circ a_2 = a_2$$

the definition (10) is equivalent to the three axioms SL_1, SL_2, SL_3 [20]. The set of actions can be completed to a semi-lattice by associating a least upper bound to the couples of actions and, if needed, by extending the actions set with artificial actions. In many practical cases, the actions sets do not include the top element that must be artificially created. This is the case of IPsec.

5 More complex example of composition: the `AND` case

We investigated the simple cases of policy rules in order to find structures where the conflict resolution can be applied. We examine now the behaviour of the conflict resolution mechanisms when logical `AND` and `OR` operations are applied to the action clause of the rules. Firstly, we study how the actions set changes if we add `AND` of actions and the relations between `AND`-ed conditions and lest upper bounds. Finally we show, by presenting an example, that in almost all the practical cases these problems have an easy and natural solution.

The structure of $\{A_j\}_{j\in J}$ by using the Disjunctive Normal Form (DNF) is:

$$\{A_j\}_{j\in J} = \bigvee_{k\in K} \bigwedge_{i\in I} a_{ki} = (a_{11}\wedge\cdots\wedge a_{1n})\vee(a_{21}\wedge\cdots\wedge a_{2n})\vee\cdots\vee(a_{m1}\wedge\cdots\wedge a_{mn}) \tag{5}$$

The right side of the formula (5) means that "if the conditions are all met then apply $(a_{11} \wedge a_{12} \wedge \cdots \wedge a_{1n})$ or apply $(a_{21} \wedge a_{22} \wedge \cdots \wedge a_{2n})$ or $\cdots$ or apply $(a_{m1} \wedge a_{m2} \wedge \cdots \wedge a_{mn})$.

Every $(a_{k1} \wedge a_{k2} \wedge \cdots \wedge a_{kn})$ can be considered as a single non-atomic action, i.e. a sequence or a parallel execution of the actions a_{ki}. Further studies must be done in order to characterize the resolution processes for more complex type of actions in practical applications. We study the behaviour of a conflict resolution process, if we take into account, for each couple of actions $a, b \in \mathcal{A}$, a new action

$z = a \wedge b$ and we extend the previous definition to the general case of $\bigwedge_{i \in I} a_i$. In practical applications, it is not needed to consider all the n-tuples, but it suffices to restrict the attention to those that appear in the set of rules. In this way, $\mathcal{A}$ is enlarged to $\mathcal{A}^{\wedge}$ including all the considered n-tuples. The actions set $\mathcal{A}$ is a semi-lattice and this algebraic structure changes when it is extended to $\mathcal{A}^{\wedge}$. The action $z = a \wedge b$ can be considered greater than the starting actions because imposes not only a or b but both[2]. So the problem is to relate the action $z = a \wedge b$ with the least upper bound $l = \mathrm{lub}\{a, b\}$ (see figure 4 or figure 5). Moreover, it must be considered the relation of $a \wedge b$ with other elements of $\mathcal{A}$. It is impossible to define a systematic technique to select the correct mode of interconnecting z and l, and this decision must be taken in function of the policy management strategy. The result of this task is a rearrangement of the starting semi-lattice that maintains the form of a partially ordered set. But in order to make $\mathcal{A}^{\wedge}$ a semi-lattice it is needed to add missing least upper bounds.

Previous problem may appear difficult to solve and too abstract to be interesting for practical applications. But is rarely needed to consider them in detail. In fact, actions often concern separated domains of application. This is the case of many cryptographic actions where three categories may be identified: hash, cipher and compression. These kind of actions are implicitly combined by using `AND` connection. We introduce abstract non overlapping categories, $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, endowed with a partial order $(\mathcal{A}_i, \leq_i)$, e.g. RC2-40 $<$ DES $<$ 3DES $<$ AES. Since all the actions in the same category are mutually exclusive, they are obtained by `AND`-ing elements that belong to different categories. For this reason, the structure of $\mathcal{A}$ is the cartesian product of partially ordered sets $\mathcal{A}_\pi = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$. It can be shown that the cartesian product of partially ordered sets is another partially ordered set with the "$\leq$" defined by the relation $a_1 \leq a_2 \Longleftrightarrow a_{1i} \leq_i a_{2i}, \forall i \leq n$ where $a_1 = (a_{11}, a_{12}, \ldots, a_{1n}) \in \mathcal{A}_\pi, a_2 = (a_{21}, a_{22}, \ldots, a_{2n}) \in \mathcal{A}_\pi$. $\mathcal{A}_\pi$ must be a semi-lattice, but, if $a_i \in \mathcal{A}_i$ and $a_j \in \mathcal{A}_j$, with $i < j$, it is possible to write $a_i = (\varnothing, \ldots, \varnothing, a_i, \varnothing, \ldots, \varnothing) \in \mathcal{A}_\pi$, $a_j = (\varnothing, \ldots, \varnothing, \varnothing, a_j, \varnothing, \ldots, \varnothing) \in \mathcal{A}_\pi$ and $a_i \wedge a_j = (\varnothing, \ldots, \varnothing, a_i, \varnothing, \ldots, \varnothing, a_j, \varnothing, \ldots, \varnothing)$. We note that $a_i \wedge a_j$ is also the least upper bound $\mathrm{lub}\{a_i, a_j\}$ and we no rearrangement is needed.

[2]This is not true when the "$\circ$" operation abstracts a conflict resolution strategy that selects less restrictive actions. However, these methods can be reduced to the examined case by defining the "$\leq$" operations as $a_1 \leq a_2 \Leftrightarrow a_1 \circ a_2 = a_1$ and the actions in a negative form, i.e. "do not apply a_1", "do not apply a_2" and so on.

6 More complex example of composition: the OR case

After having analysed the single components of formula (5), we will now consider the case of $\bigvee_{k\in K} a_k$ with $a_k \in \mathcal{A}^{\wedge}$. The first step consists in assigning to the "$\vee$" logical operation the meaning of the union in the sets. In fact, if a rule imposes a_1 or a_2 or $\ldots$ or a_n, it means that it is possible to choose the action in a set of allowed actions. It is important to distinguish the OR case from a conflict. In fact, a policy rule having more than one action in the action clause expresses the will of allowing different actions; conversely a conflict is in general non expected. This type of rules is found in all the SSL browsers where it is possible to choose between different Cipher Suites (e.g. RSA+SHA1+RC4-128 or AES+RSA+DH+SHA1).

Let us introduce an internal operation "$*$" that represents the conflict resolution process in $2^{\mathcal{A}^{\wedge}}$:

$$\begin{array}{rccccc} * : & 2^{\mathcal{A}^{\wedge}} & \times & 2^{\mathcal{A}^{\wedge}} & \longrightarrow & 2^{\mathcal{A}^{\wedge}} \\ & S_a & * & S_b & \longmapsto & S_* \end{array}$$

with $S_* = \{a \circ b, \forall a \in S_a, \forall b \in S_b\}$. In other words, the composition of two subsets of actions gives another subset constituted by all the compositions in $\mathcal{A}^{\wedge}$ of elements in S_a with elements in S_b. $(2^{\mathcal{A}^{\wedge}}, *)$ must be a semi-lattice, i.e. associative, commutative and idempotent.

Associativity ($S_a * (S_b * S_c) = (S_a * S_b) * S_c$) and commutativity ($S_a * S_b = S_b * S_a$) are trivially verified since they derive from SL_1 and SL_2 in $\mathcal{A}^{\wedge}$.

The idempotence is in general non valid. It is possible to choose $a, b \in S_a$ and $a \circ b \notin S_a$ and this demonstrates that $S_a * S_a \neq S_a$. The construction of a subset of the subsets of $\mathcal{A}^{\wedge}$ in which the idempotence holds (i.e. a subsets $S \subseteq \mathcal{A}^{\wedge}$ such that $S * S = S$ or, equivalently, $\forall\ a, b \in S \Rightarrow a \circ b \in S$), means that the "$\circ$" operation must be closed in S. Since associativity, commutativity, idempotence and closure hold in S, S is a sub-semi-lattice. We call $\mathcal{A}^{\vee}$ the set of the sub-semi-lattices of $\mathcal{A}^{\wedge}$.

The requirement of action clauses in form of sub-semi-lattice is not so restrictive. In fact, single actions of $\mathcal{A}^{\wedge}$ and intervals in a semi-lattice are sub-semi-lattices. The union of intervals or, in general, the union of sub-semi-lattices is not a sub-semi-lattice, but it is easy to show that the subset obtained by adding the least upper bound of the involved elements is a sub-semi-lattice.

We conclude that the action set $\mathcal{A}^{\vee}$, needed to use the logical OR operation in the action clause, must be the *set of all the sub-semi-lattices of the semi-lattice* $\mathcal{A}^{\wedge}$.

7 Conclusions and future work

In this paper we presented a formal model to describe policy rules and policy sets. We formalized a mathematical background needed to define policy conflicts and to solve them with methods that use only the value of the action clause. We deduced that every actions set must be a semi-lattice in order to be used for conflict resolution. We analysed a simple case where action clauses are composed only by one action, and we extended it to action clauses formed by actions connected with logical `AND` and `OR` operations. We listed a series of activities and tasks that must be considered by the policy management to assure that the policy conflict resolution mechanism works.

The results obtained in this paper can be extended to the case of policy rules that include the `NOT` operation for actions. It is also possible to add logical constraints between actions that modify the actions set. The application of the semi-lattice based approach must be investigated for more complex resolution strategies. In addition, we are working to an object-oriented application for policy conflict resolution simulations. Moreover, we think that it is straightforward to use this model for IPsec channels or web applications protected with SSL.

References

[1] S. Kent, R. Atkinson, "IP Encapsulating Security Payload (ESP)", *RFC-2406*

[2] T. Dierks, R. Atkinson, "The TLS Protocol – Version 1.0", *RFC-2246*

[3] M. Baugher, B. Weis, T. Hardjono, H. Harney, "The Group Domain of Interpretation", *RFC-3547*

[4] Westerinen et al., "Terminology for Policy-Based Management", *RFC-3198*

[5] S. Bistarelli, U. Montanari, F. Rossi, "Semiring-Based Constraint Solving and Optimization", *Journal of ACM, vol.44, n.2, pp. 201-236, 1997*

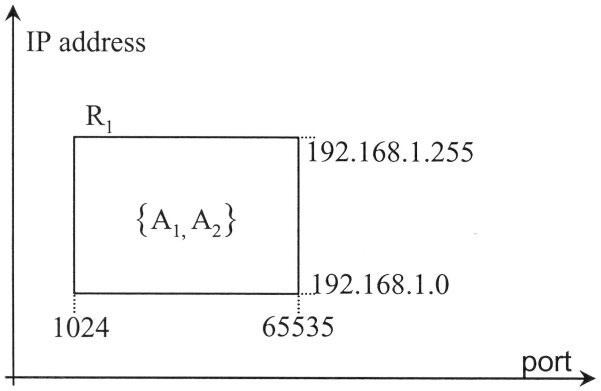
[6] V.G. Bharadway, J.S. Baras "Towards Automated Negotiation of Access Control Policies", *in Policy 2003: Workshop on Policies for Distributed Systems and Networks, pp. 111-119 June 04 - 06, 2003 Lake Como, Italy*

[7] P. Bonatti, S. de Capitani di Vimercati, P. Samarati, "An Algebra for Composing Access Control Policies", *ACM Transactions on Programming Languages and Systems, Vol. 5, Issue 1, pp. 1-35, 2002*

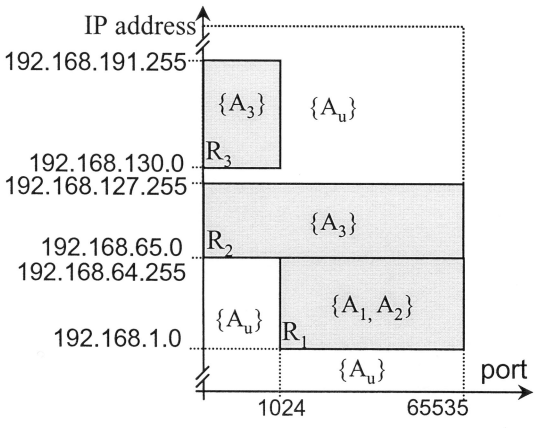
[8] J. Moffet, M.S. Sloman, "Policy Hierarchies for Distributed System Management", *IEEE JSAC, Vol. 11, n.9, 1993*

[9] E.C. Lupu, M.S. Sloman, "Conflicts in Policy-Based Distributed Systems Management", *in IEEE Transactions on Software Engineering, vol. 25, Issue 6, pp. 852-869, 1999*

[10] A.K. Bandara, E.C. Lupu, A. Russo, "Using Event Calculus to Formalise Policy Specification and Analysis", *in Policy 2003: Workshop on Policies for Distributed Systems and Networks, pp. 26-41, June 04 - 06, 2003 Lake Como, Italy*
[11] A.c. Kakas, R.A. Kowalski and F. Toni, The role of abduction in logic programming, *in Handbook of logic in Artificial Intelligence and Logic Programming 5, 5, pp. 235-324, Oxford University Press, 1998.*
[12] Moore et al., "Policy Core Information Model – Version 1 Specification", *RFC-3060*
[13] S.C. Kleene, "Mathematical Logic", *John Wiley & Sons Inc., 1967*
[14] J. Jason, L. Rafalow, E. Vyncke, "IPsec Configuration Policy Information Model", *RFC-3585*
[15] Check Point Software L.T.D., "Firewall-1 Access Control", www.checkpoint.com/products/protect/firewall-1_access.html
[16] S. Castano, M. Fugini, G. Martella, P. Samarati, "Database Security", *Addison Wesley, 1994*
[17] N. Bourbaki, "General Topology", *Springer-Verlag, 1971*
[18] F. M. Brown, "Boolean Reasoning", *Dover Publications Inc., 1990*
[19] G. Birkhoff, "Lattice theory", *Amer. Math. Soc. Colloquium publication 25, 3rd edition, 1967*
[20] G. Szasz, "Théorie des treillis", *Dunod Éditeur, 1971*



(a)



(b)

Figure 1: (a) simple rule expressed as direct product of two functions; (b) policy function

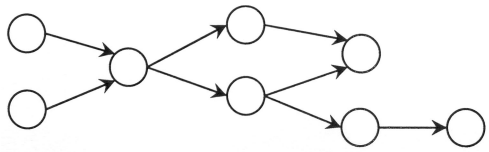


Figure 3: An example of representation of partial odered set

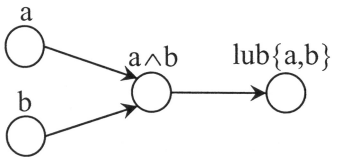


Figure 4: Example of cover graph if $z \leq l$

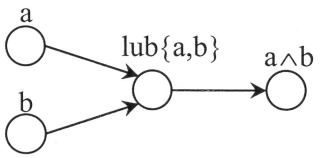


Figure 5: Example of cover graph if $l \leq z$

Performance-sensitive Real-time Risk Management is NP-Hard

Ashish Gehani
Department of Computer Science, Duke University

Abstract

This paper introduces a formal model for quantifying host-based risk, and two classes of primitives that can be utilized to manage it. The performance overhead introduced by selecting a set of response primitives to manage the risk is also factored into the framework. The resulting problem, of managing risk while minimizing the impact on performance is shown to be NP-hard. Since the goal is real-time response, a heuristic is described that allows the first primitive to be chosen in constant time (which is frequently sufficient to disrupt an attack).

1 Introduction

If a system is simple, its properties can be completely ascertained, either analytically or empirically. When a system is complex, analytical tools can not address all issues and empirical techniques require more resources than can typically be devoted to the task of verification. To secure an information processing system, it is necessary to ensure that it obeys a set of rules and maintains a set of properties. Modern computing systems are complex. This makes it infeasible to address the task empirically. Analytical techniques can alleviate the issue, but they can not resolve it completely [Harrison76]. In this context, *risk* serves as a measure of the extent to which the security of the system is likely to be violated.

Early approaches to computer security *risk management* employed static strategies, such as the use of passwords, discretionary or mandatory access control, and encrypted network connections [Fletcher95]. These approaches did not provide the flexibility needed to allow risk managers to alter the levels of risk they were willing to tolerate in exchange for commensurate costs. Hence, techniques to dynamically vary the risk were developed. Inherent in the new approach was the need for *risk analysis* to quantify the level of risk present in each configuration of a system.

Large data processing centers started to use the Annual Loss Expectancy (ALE) metric [FIPS31], [FIPS65]. To compute it, the set of all possible *hazards* that could impact the system over the course of a year was enumerated as $H = \{h_1, h_2, \ldots, h_n\}$.

The loss associated with each hazard h_α was denoted by $l(h_\alpha)$ and the frequency with which it was likely to occur was denoted by $f(h_\alpha)$. The risk was then calculated using:

$$ALE = \sum_{\alpha=1}^{\alpha=n} f(h_\alpha) \times l(h_\alpha) \quad (1)$$

The utilization of the paradigm by a number of commercial tools [NIST91] coupled with a focused research effort [CSRMMBW88], [CSRMMBW89], [CSRMMBW90],[CSRMMBW91] resulted in a number of improvements. The hazard construct was decomposed into threat and vulnerability components. The likelihood of a threat being present was added as a factor, replacing the hazard frequency. Each vulnerability was coupled with an associated safeguard. The loss from a hazard's occurrence was modeled as a set of consequences that could affect each asset under consideration. Finally, risk management was formulated as the maintenance of a set of requirements framed as constraints on the aforementioned factors [NIST800-12].

This paper contributes specific semantics in the operating system paradigm for the generic risk analysis concepts of threats, likelihoods, exposures, safeguards, assets and consequences. These are then utilized to construct a model for managing a host's risk in real-time.

2 Runtime Risk Management

The primary goal of an intrusion response system is to guard against attacks. Primitives that address specific threats have been developed. However, invoking these arbitrarily may safeguard part of the system but leave other weaker areas exposed. Thus, to effect a rational response, it is necessary to weigh all the possible alternatives. A course of action must then be chosen which will result in the least damage, while simultaneously assuring that cost constraints are respected. This is the very problem that risk management addresses.

2.1 Response Primitives

Two types of response primitives are required. They need modifications of the access control subsystem and the filesystem, respectively. The first type institutes further runtime checks before granting specific permissions, in order to reduce the host exposure. The second type requires data to be stored in a protected state, and allows the rapid disabling of transparent decryption, signing of modifications and remote replication of specific files, in order to curtail the consequences of an attack. An instance of either type can be chosen as a response at any time. Choosing a primitive imposes an overhead on system performance that is proportional to the frequency with which the primitive is used in a typical workload.

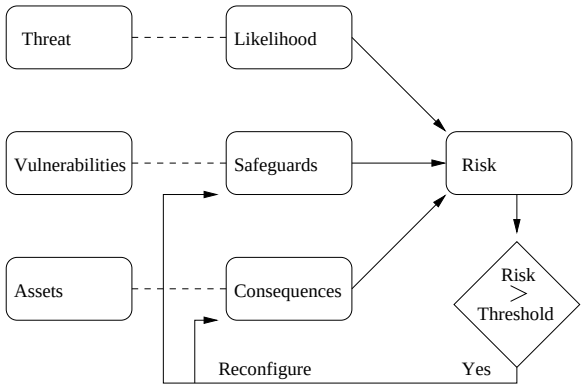


Figure 1: Risk can be analyzed as a function of the threats, their likelihood, the vulnerabilities, the safeguards, the assets and the consequences. Risk can be managed by using the safeguards to control the exposure of vulnerabilities and manipulating the assets to limit the consequences.

2.2 Risk Factors

Analyzing the risk that a system is faced with requires knowledge of a number of factors. Below we describe each of these factors along with its associated semantics. We define these in the context of the operating system paradigm since our goal is host-based response.

The paradigm assumes the existence of an operating system with a trusted reference monitor that mediates access by subjects to objects in the system. In addition, the file system and auditing subsystem are assumed to be trusted components in the operating system. Finally, a host-based intrusion detection system is assumed to be present and operational.

Threats A *threat* is an agent that can cause harm to an asset in the system. We define a threat to be a specific attack against any of the application or system software that is running on the host. It is characterized by an intrusion detection signature. The set of threats is denoted by $T = \{t_1, t_2, \ldots\}$, where $t_\alpha \in T$ is an intrusion detection signature. Since t_α is a host-based signature, it is comprised of an *ordered set* of events $S(t_\alpha) = \{s_1, s_2, \ldots\}$. If this set occurs in the order recognized by the rules of the intrusion detector, it signifies the presence of an attack.

Likelihood The *likelihood* of a threat is the hypothetical probability of it occurring. If a signature has been partially matched, the extent of the match serves as a predictor of the chance that it will subsequently be completely matched. A function μ is used to compute the likelihood of threat t_α. μ can be threat specific and will

depend on the history of system events that are relevant to the intrusion signature. Thus, if $E = \{e_1, e_2, \ldots\}$ denotes the ordered set of all events that have occurred, then:

$$\mathcal{T}(t_\alpha) = \mu(t_\alpha, E \overset{\prec}{\cap} S(t_\alpha)) \tag{2}$$

where $\overset{\prec}{\cap}$ yields the set of all events that occur *in the same order* in each input set.

Assets An *asset* is an item that has value. We define the assets to be the data stored in the system. In particular, each file is considered a separate object $o_\beta \in O$, where $O = \{o_1, o_2, \ldots\}$ is the set of assets. A set of objects $A(t_\alpha) \subseteq O$ is associated with each threat t_α. Only objects $o_\beta \in A(t_\alpha)$ can be harmed if the attack that is characterized by t_α succeeds.

Consequences A *consequence* is a type of harm that an asset may suffer. Three types of consequences can impact the data. These are the loss of confidentiality, integrity and availability. If an object $o_\beta \in A(t_\alpha)$ is affected by the threat t_α, then the resulting costs due to the loss of confidentiality, integrity and availability are denoted by $c(o_\beta)$, $i(o_\beta)$, and $a(o_\beta)$ respectively. Any of these values may be 0 if the attack can not effect the relevant consequence. However, all three values associated with a single object can not be 0 since in that case $o_\beta \in A(t_\alpha)$ would not hold. Thus, the consequence of a threat t_α is:

$$\mathcal{C}(t_\alpha) = \sum_{o_\beta \in A(t_\alpha)} c(o_\beta) + i(o_\beta) + a(o_\beta) \tag{3}$$

By removing an asset from the system, the consequences it faces can be *curtailed* [Gehani03]. In the case of data availability, replication serves this purpose, while in the case of confidentiality and integrity, cryptographic operations can be used. For the purpose of estimating risk, a consequence *curtailment* effectively removes the asset from the analysis.

Vulnerabilities A *vulnerability* is a weakness in the system. It results from an error in the design, implementation or configuration of either the operating system or application software. The set of vulnerabilities present in the system is denoted by $W = \{w_1, w_2, \ldots\}$. $W(t_\alpha) \subseteq W$ is the set of weaknesses exploited by the threat t_α to subvert the security policy.

Safeguards A *safeguard* is a mechanism that controls the exposure of the system's assets. The reference monitor's set of permission checks $P = \{p_1, p_2, \ldots\}$ serve as safeguards in an operating system. Since the reference monitor mediates access to all objects, a vulnerability's exposure can be limited by denying the relevant permissions. The set $P(w_\gamma) \subseteq P$ contains all the permissions that

are requested in the process of exploiting vulnerability w_γ. The static configuration of a conventional reference monitor either grants or denies access to a permission p_λ. This *exposure* is denoted by $v(p_\lambda)$, with the value being either 0 or 1. An *active reference monitor* [1] [Gehani03] can reduce the exposure of a statically granted permission to $v'(p_\lambda)$, a value in the range $[0, 1]$. This reflects the nuance that results from evaluating predicates as *auxiliary safeguards*.)

Thus, if all auxiliary safeguards are utilized, the total exposure to a threat t_α is:

$$\mathcal{V}(t_\alpha) = \sum_{p_\lambda \in \hat{P}(t_\alpha)} \frac{v(p_\lambda) \times v'(p_\lambda)}{|\hat{P}(t_\alpha)|} \tag{4}$$

where:

$$\hat{P}(t_\alpha) = \bigcup_{w_\gamma \in W(t_\alpha)} P(w_\gamma) \tag{5}$$

2.3 Risk Analysis

The risk to the host is the sum of the risks that result from each of the threats that it faces. The risk from a single threat is the product of the chance that the attack will occur, the exposure of the system to the attack, and the cost of the consequences of the attack succeeding [NIST800-12]. Thus, the cumulative risk faced by the system is:

$$\mathcal{R} = \sum_{t_\alpha \in T} \mathcal{T}(t_\alpha) \times \mathcal{V}(t_\alpha) \times \mathcal{C}(t_\alpha) \tag{6}$$

2.4 Risk Management

If the risk posed to the system is to be managed, the current level must be continuously monitored. When the risk rises past the threshold that the host can tolerate, the system's security must be tightened. Similarly, when the risk decreases, the restrictions can be relaxed to improve performance and usability. This process is elucidated below.

The system's risk can be reduced either by reducing the exposure of vulnerabilities or limiting the consequences to the data in the event of a successful attack. The former is effected through the use of auxiliary safeguards prior granting a permission. The latter is realized by cryptographically protecting threatened files. Additionally, both approaches may be used simultaneously. Similarly, if the threat reduces, the restrictive permission checks and data protection can be relaxed.

[1] An *active reference monitor* allows each permission to be associated with an independent set of constraints which are verified at runtime prior to granting the permission. By limiting the circumstances under which the permission will be granted, the exposure of the resource being protected is reduced by a pre-determined fraction.

2.4.1 Managed Risk

The set of permissions P is kept partitioned into two disjoint sets, $\Psi(P)$ and $\Omega(P)$, that is $\Psi(P) \cap \Omega(P) = \phi$ and $\Psi(P) \cup \Omega(P) = P$. The set $\Psi(P) \subseteq P$ contains the permissions for which auxiliary safeguards are currently active. The remaining permissions $\Omega(P) \subseteq P$ are handled conventionally by the reference monitor, using only static lookups rather than evaluating associated predicates prior to granting these permissions. Similarly, the set of files O is kept partitioned into two disjoint sets, $\Psi(O)$ and $\Omega(O)$, where $\Psi(O) \cap \Omega(O) = \phi$ and $\Psi(O) \cup \Omega(O) = O$. The set $\Psi(O) \subseteq O$ contains the files that are currently inaccessible and unmodifiable due to their cryptographic encapsulation. The remaining files $\Omega(O) \subseteq O$ are transparently accessible and modifiable.

At any given point, when safeguards $\Psi(P)$ and curtailments $\Psi(O)$ are in use, the current risk $\mathcal{R}'$ is calculated with:

$$\mathcal{R}' = \sum_{t_\alpha \in T} \mathcal{T}(t_\alpha) \times \mathcal{V}'(t_\alpha) \times \mathcal{C}'(t_\alpha) \tag{7}$$

where:

$$\mathcal{V}'(t_\alpha) = \sum_{p_\lambda \in \hat{P}(t_\alpha) \cap \Omega(P)} \frac{v(p_\lambda)}{|\hat{P}(t_\alpha)|} + \sum_{p_\lambda \in \hat{P}(t_\alpha) \cap \Psi(P)} \frac{v(p_\lambda) \times v'(p_\lambda)}{|\hat{P}(t_\alpha)|} \tag{8}$$

and:

$$\mathcal{C}'(t_\alpha) = \sum_{o_\beta \in A(t_\alpha) \cap \Omega(O)} c(o_\beta) + i(o_\beta) + a(o_\beta) \tag{9}$$

2.4.2 Risk Tolerance

While the risk must be monitored continuously, there is a computational cost incurred each time it is recalculated. Therefore, the frequency with which the risk is estimated must be minimized to the extent possible. Instead of calculating the risk synchronously at fixed intervals in time, we exploit the fact that the risk level only changes when the threat to the system is altered.

An intrusion detector is assumed to be monitoring the system's activity. Each time it detects an event that changes the extent to which a signature has been matched, it passes the event e to the intrusion response subsystem. The level of risk $\mathcal{R}_b$ before e occurred is noted, and then the level of risk $\mathcal{R}_a$ after e occurred is calculated. Thus, $\mathcal{R}_a = \mathcal{R}_b + \epsilon$, where ϵ denotes the change in the risk. Since the risk is recalculated only when it actually changes, the computational cost of monitoring it is minimized.

Each time an event e occurs, either the risk decreases, stays the same or increases. Each host is configured to tolerate risk upto a threshold, denoted by $\mathcal{R}_\emptyset$. After each event e, the system's response guarantees that the risk will return to a level below

this threshold. As a result, $\mathcal{R}_b < \mathcal{R}_0$ always holds. If $\epsilon = 0$, then no further risk management steps are required.

If $\epsilon < 0$, then $\mathcal{R}_a < \mathcal{R}_0$ since $\mathcal{R}_a = \mathcal{R}_b + \epsilon < \mathcal{R}_b < \mathcal{R}_0$. At this point, the system's security configuration is more restrictive than it needs to be. To improve system usability and performance, the response system must deactivate appropriate safeguards and curtailments, while ensuring that the risk level does not rise past the threshold $\mathcal{R}_0$.

If $\epsilon > 0$ and $\mathcal{R}_a \leq \mathcal{R}_0$, then no action needs to be taken. Even though the risk has increased, it is below the threshold that the system can tolerate, so no further safeguards or curtailments need to be introduced. In addition, the system will not be able to find any set of unused safeguards and curtailments whose removal will increase the risk by less than $\mathcal{R}_0 - \mathcal{R}_b - \epsilon$, since the presence of such a combination would also mean that the set existed before e occurred. It is not possible that such a combination of safeguards and curtailments existed before e occurred since they would also have satisfied the condition of being less than $\mathcal{R}_0 - \mathcal{R}_b$ and would have been utilized before e occurred in the process of minimizing the impact on performance in the previous step.

If $\epsilon > 0$ and $\mathcal{R}_a > \mathcal{R}_0$, then action is required to reduce the risk to a level below the threshold of tolerance. The response system must search for and implement a set of safeguards and curtailments to this end.

2.4.3 Recalculating Risk

When the risk is calculated the first time, Equation 6 is used. Therefore, the cost is $O(|T| \times |P| \times |O|)$. Since the change in the risk must be repeatedly evaluated during real-time reconfiguration of the runtime environment, it is imperative the cost is minimized. This is achieved by caching all the values $\mathcal{V}(t_\alpha) \times \mathcal{C}'(t_\alpha)$ associated with threats $t_\alpha \in T$ during the evaluation of Equation 6. Subsequently, when an event e occurs, the change in the risk $\epsilon = \delta(\mathcal{R}', e)$ can be calculated with cost $O(|T|)$ as described below.

The ordered set E refers to all the events that have occurred in the system prior to the event e. The change in the likelihood of a threat t_α due to e is:

$$\delta(\mathcal{T}(t_\alpha), e) = \mu(t_\alpha, (E \cup e) \overset{\prec}{\cap} S(t_\alpha)) - \mu(t_\alpha, E \overset{\prec}{\cap} S(t_\alpha)) \tag{10}$$

The set of threats affected by e is denoted by $\Delta(T, e)$. A threat $t_\alpha \in \Delta(T, e)$ is considered to be affected by e if $\delta(\mathcal{T}(t_\alpha), e) \neq 0$, that is its likelihood changed due to the event e. The resultant change in the risk level is:

$$\delta(\mathcal{R}', e) = \sum_{t_\alpha \in \Delta(T,e)} \delta(\mathcal{T}(t_\alpha), e) \times \mathcal{V}'(t_\alpha) \times \mathcal{C}'(t_\alpha) \tag{11}$$

2.5 Cost/Benefit Analysis

After an event e occurs, if the risk level $\mathcal{R}_a$ increases past the threshold of risk tolerance $\mathcal{R}_0$, the goal of the response engine is to reduce the risk by $\delta_g \geq \mathcal{R}_a - \mathcal{R}_0$ to a level below the threshold. To do this, it must select a subset of permissions $\rho(\Omega(P)) \subseteq \Omega(P)$ and a subset of objects $\rho(\Omega(O)) \subseteq \Omega(O)$, such that adding safeguards and curtailments respectively to the two sets will reduce the risk to the desired level. By ensuring that the permissions in $\rho(\Omega(P))$ are granted only after relevant predicates are verified and files in $\rho(\Omega(O))$ are cryptographically protected, the resulting risk level is reduced to:

$$\mathcal{R}'' = \sum_{t_\alpha \in T} \mathcal{T}(t_\alpha) \times \mathcal{V}''(t_\alpha) \times \mathcal{C}''(t_\alpha) \tag{12}$$

where the new vulnerability measure, based on Equation 4, is:

$$\mathcal{V}''(t_\alpha) = \sum_{p_\lambda \in (\hat{P}(t_\alpha) \cap \Omega(P) - \rho(\Omega(P)))} \frac{v(p_\lambda)}{|\hat{P}(t_\alpha)|} + \sum_{p_\lambda \in (\hat{P}(t_\alpha) \cap \Psi(P) \cup \rho(\Omega(P)))} \frac{v(p_\lambda) \times v'(p_\lambda)}{|\hat{P}(t_\alpha)|} \tag{13}$$

and the new consequence measure, based on Equation 3, is:

$$\mathcal{C}''(t_\alpha) = \sum_{o_\beta \in (A(t_\alpha) \cap \Omega(O) - \rho(\Omega(O)))} c(o_\beta) + i(o_\beta) + a(o_\beta) \tag{14}$$

Instead, after an event e occurs, if the risk level $\mathcal{R}_a$ decreases, the goal of the response engine is to allow the risk to rise by $\delta_g \leq \mathcal{R}_0 - \mathcal{R}_a$ to a level below the threshold of risk tolerance $\mathcal{R}_0$. To do this, it must select a subset of permissions $\rho(\Psi(P)) \subseteq \Psi(P)$ and a subset of objects $\rho(\Psi(O)) \subseteq \Psi(O)$, such that removing the safeguards and curtailments currently in use for these two sets will yield the maximum improvement to runtime performance. After the safeguards and curtailments are relaxed, the risk level will rise to:

$$\mathcal{R}'' = \sum_{t_\alpha \in T} \mathcal{T}(t_\alpha) \times \mathcal{V}''(t_\alpha) \times \mathcal{C}''(t_\alpha) \tag{15}$$

where the new vulnerability measure, based on Equation 4, is:

$$\mathcal{V}''(t_\alpha) = \sum_{p_\lambda \in \hat{P}(t_\alpha) \cap \Omega(P) \cup \rho(\Psi(P))} \frac{v(p_\lambda)}{|\hat{P}(t_\alpha)|} + \sum_{p_\lambda \in \hat{P}(t_\alpha) \cap \Psi(P) - \rho(\Psi(P))} \frac{v(p_\lambda) \times v'(p_\lambda)}{|\hat{P}(t_\alpha)|} \tag{16}$$

and the new consequence measure, based on Equation 3, is:

$$\mathcal{C}''(t_\alpha) = \sum_{o_\beta \in A(t_\alpha) \cap \Omega(O) \cup \rho(\Psi(O))} c(o_\beta) + i(o_\beta) + a(o_\beta) \tag{17}$$

There are $O(2^{(|P|+|O|)})$ ways of choosing subsets $\rho(\Omega(P)) \subseteq \Omega(P)$ and $\rho(\Omega(O)) \subseteq \Omega(O)$ for risk reduction or subsets $\rho(\Psi(P)) \subseteq \Psi(P)$ and $\rho(\Psi(O)) \subseteq \Psi(O)$ for risk relaxation. When selecting from the possibilities, the primary constraint is the maintenance of the bound $\mathcal{R}'' < \mathcal{R}_0$, where $\mathcal{R}'' = \mathcal{R}_a - \delta_g$ in the case of risk reduction, and $\mathcal{R}'' = \mathcal{R}_a + \delta_g$ in the case of risk relaxation.

The choice of safeguards and curtailments also impacts the performance of the system. Evaluating predicates prior to granting permissions introduces latency in system calls. Cryptographically protecting objects decreases usability. Hence, the choice of subsets $\rho(\Omega(P))$ and $\rho(\Omega(O))$ or subsets $\rho(\Psi(P))$ and $\rho(\Psi(O))$ is subject to the secondary goal of minimizing the overhead introduced.

The adverse impact of a safeguard or curtailment is proportional to the frequency with which it is utilized in the system's workload. Given a typical workload, we can count the frequency $f(p_\lambda)$ with which permission p_λ is requested in the workload. Similarly, we can count the frequency $f(o_\beta)$ with which file o_β is accessed in the workload. This can be done for all permissions and files. The cost of utilizing subsets $\rho(\Omega(P))$ and $\rho(\Omega(O))$ for risk reduction can then be calculated with:

$$\zeta(\rho(\Omega(P)), \rho(\Omega(O))) = \sum_{p_\lambda \in \rho(\Omega(P))} f(p_\lambda) + \sum_{o_\beta \in \rho(\Omega(O))} f(o_\beta) \tag{18}$$

Similarly, if the safeguards of subset $\rho(\Psi(P))$ and the curtailments of consequences to assets in subset $\rho(\Psi(O))$ are relaxed, the resulting reduction in runtime cost can be calculated with:

$$\zeta(\rho(\Psi(P)), \rho(\Psi(O))) = \sum_{p_\lambda \in \rho(\Psi(P))} f(p_\lambda) + \sum_{o_\beta \in \rho(\Psi(O))} f(o_\beta) \tag{19}$$

The ideal choice of safeguards and curtailments will minimize the safeguards' and curtailments' impact on performance, while simultaneously ensuring that the risk remains below the threshold of tolerance. Thus, for risk reduction we wish to find:

$$\min \zeta(\rho(\Omega(P)), \rho(\Omega(O))), \quad \mathcal{R}'' \leq \mathcal{R}_0 \tag{20}$$

In the context of risk relaxation, we wish to find:

$$\max \zeta(\rho(\Psi(P)), \rho(\Psi(O))), \quad \mathcal{R}'' \leq \mathcal{R}_0 \tag{21}$$

2.6 Complexity

We note that the semantics of risk management require that at each step the risk must be reduced below the threshold of tolerance. This precludes optimization strategies such as minimizing a weighted sum of risk and runtime performance. We conclude that runtime risk management is a $0-1$ integer non-linear programming problem with a linear objective function and quadratic constraint. The decision problem that corresponds to the aforementioned optimization problem is NP-hard [Garey79] as argued below.

2.6.1 Optimization Problem

Risk reduction can viewed as selecting a set of vertices in a vertex-weighted, edge-weighted bipartite graph, such that the sum of the weights of the vertices selected is minimized, subject to the constraint that the sum of the weights of the edges present in the subgraph induced by the selected vertices is greater than a fixed threshold. The vertices in one partition correspond to the set of unsafeguarded permissions $\Omega(P)$, while the vertices in the other partition correspond to the set of objects whose access is uncurtailed $\Omega(O)$. The weight of each vertex $p_\lambda \in \Omega(P)$ is $f(p_\lambda)$, the frequency of the permission in the workload, while the weight of each vertex $o_\beta \in \Omega(O)$ is $f(o_\beta)$, the frequency of the object in the workload.

The weight of an edge (p_λ, o_β) between a permission p_λ and an object o_β is the contribution to the total risk that results from the exposure of the corresponding permission and the cost of the corresponding object's security being subverted. Thus the weight of the edge is:

$$\begin{aligned} w(p_\lambda, o_\beta) &= \sum_{t_\alpha \in T \,:\, p_\lambda \in \hat{P}(t_\alpha) \cap \Omega(P) \;\wedge\; o_\beta \in A(t_\alpha) \cap \Omega(O)} \mathcal{T}(t_\alpha) \times \frac{v(p_\lambda) \times v'(p_\lambda)}{|\hat{P}(t_\alpha)|} \\ &\times \; c(o_\beta) + i(o_\beta) + a(o_\beta) \end{aligned} \tag{22}$$

The fixed threshold is the risk tolerance, $\mathcal{R}_0$. Risk relaxation is similar, with the exception that the sum of weights of the chosen vertices (which are from the sets $\Psi(P)$ and $\Psi(O)$ instead of sets $\Omega(P)$ and $\Omega(O)$) must be *maximized*, while the sum of the weights of the edges in the induced subgraph must remain below a fixed threshold. The two optimization problems are equivalent.

2.6.2 Decision Problem

The decision problem for risk reduction takes as input: (i) a bipartite graph of the form described above, (ii) a fixed threshold which is the risk tolerance, and (iii) the sum of the weights of the subset of vertices to be chosen, which corresponds to the runtime cost of the primitives in a proposed response. The output is only `true` if the algorithm is able to find a subset of vertices whose weights add up to the specified total, while the sum of the weights of the edges in the induced subgraph is at least the specified threshold.

2.6.3 NP-Hard

Given an algorithm for the risk reduction optimization problem, the decision problem can be solved by checking if the target sum of vertices' weights is less than, equal or greater than the minimum cost output by the optimization algorithm.

Given a decision algorithm for risk reduction, we can solve the *maximum edge biclique problem* which takes a bipartite graph and a threshold as inputs and outputs

whether the graph includes a biclique that is the size of the threshold or larger. The problem is known to be NP-complete [Peeters03].

To solve the maximum edge biclique problem, we repeatedly invoke the risk reduction decision algorithm. The number of invocations is bounded above by the size of the vertex set in the graph. The input is the bipartite graph (with all vertices and edges weighted 1), the threshold and a target number of vertices that ranges during invocations from 1 to the total number of vertices in the bipartite graph. The first time the output is `true`, we stop and output `true` for the maximum edge biclique problem.

Since the risk reduction decision algorithm output `true`, the sum of the weights of the vertices (which is equal to the number of vertices since all the weights were set to 1) is the least possible such that the sum of the weights of the edges (which is equal to the number of edges since the weights of all the edges were set to 1) was at least the threshold specified. The total number of edges in a biclique is the maximum possible for a subset of vertices of the biclique's size. Thus, the smallest subset of vertices that will contain a specified number of edges is a biclique.

If all the invocations of the risk reduction decision problem produced an output of `false`, then the output for the maximum edge biclique problem is also `false`. This completes the reduction. If the risk reduction decision problem were *tractable*, then the maximum edge biclique problem would also be tractable, but it is known to be NP-complete. Therefore, the risk reduction (and risk relaxation since it is analogous) decision problems are NP-hard.

2.7 Response Selection

Determining the optimal choice of safeguards and curtailments for risk management corresponds to an NP-hard problem, as argued in Section 2.6. Additionally, there is evidence that the maximum edge biclique problem is difficult to approximate [Kogan04]. Since the choice is to be made in real-time, we will use a heuristic which guarantees that the risk threshold is maintained. The heuristic uses the greedy strategy of picking the response primitive with the highest benefit-to-cost ratio repeatedly till the constraint is satisfied. By maintaining the choices in a *heap* data structure keyed on the benefit-to-cost ratio, the first primitive in the response set can be chosen in $O(1)$ time. This is significant since implementing a single response primitive is often sufficient for disrupting an attack in progress.

Since the benefit associated with each unutilized safeguard or curtailment is the degree to which the risk will be reduced if it is used, this is a function of other safeguards or curtailments related to the threats that it affects. Similarly, since the loss of benefit associated with each currently utilized safeguard or curtailment is the degree to which the risk will increase if it is used, this is also a function of the other safeguards or curtailments associated with the threats that it affects. As a result, the benefit of adding or removing each safeguard or curtailment must be recalculated each time other safeguards or curtailments are added or removed.

2.7.1 Risk Reduction

We outline the algorithm for the case where the risk needs to be reduced. The first two steps constitute pre-processing and therefore only occur during system initialization.

Step 1 The benefit-to-cost ratio of each candidate safeguard permission $p_\lambda \in \Omega(P)$ can be calculated by:

$$\kappa(p_\lambda) = \frac{\sum\limits_{t_\alpha : p_\lambda \in (\hat{P}(t_\alpha) \cap \Omega(P))} \mathcal{T}(t_\alpha) \times \dfrac{v(p_\lambda) \times (1 - v'(p_\lambda))}{|\hat{P}(t_\alpha)|} \times \mathcal{C}'(t_\alpha)}{f(p_\lambda)} \tag{23}$$

Step 2 Similarly, the benefit-to-cost ratio of protecting an object $o_\beta \in \Omega(O)$ can be calculated by:

$$\kappa(o_\beta) = \frac{(c(o_\beta) + i(o_\beta) + a(o_\beta)) \quad \times \sum\limits_{t_\alpha : o_\beta \in (A(t_\alpha) \cap \Omega(O))} \mathcal{T}(t_\alpha) \times \mathcal{V}'(t_\alpha)}{f(o_\beta)} \tag{24}$$

Step 3 The response sets are defined as empty, that is $\rho(\Omega(P)) = \rho(\Omega(O)) = \phi$.

Step 4 The single risk reducing measure with the highest benefit-to-cost can be selected, that is:

$$\begin{aligned} &\max\ p_{max}, o_{max} \quad where: \\ p_{max} &= \max\ \kappa(p_\lambda), \quad p_\lambda \in \Omega(P) \\ o_{max} &= \max\ \kappa(o_\beta), \quad o_\beta \in \Omega(O) \end{aligned} \tag{25}$$

If it is a permission it is added to $\rho(\Omega(P))$ and if it is an object it can be added to $\rho(\Omega(O))$.

Step 5 If the choice was a permission p_λ, then the value $\kappa(o_\beta)$ must be recalculated for all objects o_β that are affected by threats which utilize p_λ in the course of their attacks. Thus, each $\kappa(o_\beta)$ must be updated if:

$$o_\beta \in \bigcup_{t_\alpha : p_\lambda \in \hat{P}(t_\alpha)} A(t_\alpha) \tag{26}$$

Instead, if the choice was an object o_β, then the value $\kappa(p_\lambda)$ must be recalculated for all permissions p_λ that are utilized in the course of an attack that affects object o_β. Thus, each $\kappa(p_\lambda)$ must be updated if:

$$p_\lambda \in \bigcup_{t_\alpha : o_\beta \in A(t_\alpha)} \hat{P}(t_\alpha) \tag{27}$$

Step 6 The risk before the candidate responses were utilized is $\mathcal{R}_a$. If the responses were activated the resulting risk $\mathcal{R}''$ is given by:

$$\mathcal{R}'' = \mathcal{R}_a \quad - \sum_{p_\lambda \in \rho(\Omega(P))} \kappa(p_\lambda) \times f(p_\lambda) \quad - \sum_{o_\beta \in \rho(\Omega(O))} \kappa(o_\beta) \times f(o_\beta) \quad (28)$$

This is equivalent to using Equations 12, 13 and 14. While the worst case complexity is the same, when few protective measures are added the cost of the above calculation is significantly lower.

Step 7 If $\mathcal{R}'' > \mathcal{R}_0$ then the system repeats the above from Step 4 onwards. However, if $\mathcal{R}'' \leq \mathcal{R}_0$ then the set of safeguards $\rho(\Omega(P))$ and the set consequence curtailing measures $\rho(\Omega(O))$ must be applied. $\rho(\Omega(P))$ should be transferred from $\Omega(P)$ to $\Psi(P)$ and $\rho(\Omega(O))$ should be transferred from $\Omega(O)$ to $\Psi(O)$. Then the response sets should be reset so $\rho(\Omega(P)) = \rho(\Omega(O)) = \phi$.

The time complexity is:

$$O((\rho(\Omega(P)) + \rho(\Omega(O))) \times (\log |P| + \log |O| + \sum_{t_\alpha \in T} (|\hat{P}(t_\alpha)| + |A(t_\alpha)|))) \quad (29)$$

In the worst case, this is $O(|P| + |O|)^2$. Unless a large variety of attacks are simultaneously launched against the target, the first factor will remain small. Additionally, if there is a strong correlation between the exposures and the consequences, then the second factor will also remain small. Thus, in practice it is likely to achieve acceptable results.

2.7.2 Risk Relaxation

In the case of risk relaxation, the algorithm becomes:

Step 1 For $p_\lambda \in \Psi(P)$ calculate:

$$\kappa(p_\lambda) = \frac{\sum_{t_\alpha : p_\lambda \in (\hat{P}(t_\alpha) \cap \Psi(P))} \mathcal{T}(t_\alpha) \times \frac{v(p_\lambda) \times (1 - v'(p_\lambda))}{|\hat{P}(t_\alpha)|} \times \mathcal{C}'(t_\alpha)}{f(p_\lambda)} \quad (30)$$

Step 2 For $o_\beta \in \Psi(O)$ calculate:

$$\kappa(o_\beta) = \frac{(c(o_\beta) + i(o_\beta) + a(o_\beta)) \quad \times \sum_{t_\alpha : o_\beta \in (A(t_\alpha) \cap \Psi(O))} \mathcal{T}(t_\alpha) \times \mathcal{V}'(t_\alpha)}{f(o_\beta)} \quad (31)$$

Step 3 Set $\rho(\Psi(P)) = \rho(\Psi(O)) = \phi$.

Step 4 Find the safeguard or curtailment which yields the least risk reduction per instance of use:

$$\begin{aligned} &\min\ p_{min}, o_{min} \quad where: \\ p_{min} &= \min\ \kappa(p_\lambda), \quad p_\lambda \in \Psi(P) \\ o_{min} &= \min\ \kappa(o_\beta), \quad o_\beta \in \Psi(O) \end{aligned} \tag{32}$$

Add it to $\rho(\Psi(P))$ if it is a permission. Instead, add it to $\rho(\Psi(O))$ if it is a file.

Step 5 Update $\kappa(o_\beta)$ if:

$$o_\beta \in \bigcup_{t_\alpha : p_\lambda \in \hat{P}(t_\alpha)} A(t_\alpha) \tag{33}$$

or update $\kappa(p_\lambda)$ if:

$$p_\lambda \in \bigcup_{t_\alpha : o_\beta \in A(t_\alpha)} \hat{P}(t_\alpha) \tag{34}$$

depending on whether a permission or a file was chosen in the previous step.

Step 6 Calculate $\mathcal{R}''$:

$$\mathcal{R}'' = \mathcal{R}_a \ + \sum_{p_\lambda \in \rho(\Psi(P))} \kappa(p_\lambda) \times f(p_\lambda) \ + \sum_{o_\beta \in \rho(\Psi(O))} \kappa(o_\beta) \times f(o_\beta) \tag{35}$$

Step 7 If $\mathcal{R}'' < \mathcal{R}_0$, repeat from Step 4. If $\mathcal{R}'' = \mathcal{R}_0$, proceed to next step. If $\mathcal{R}'' > \mathcal{R}_0$, undo last iteration of Step 4.

Step 8 Relax all measures in $\rho(\Psi(P))$ and $\rho(\Psi(O))$ and transfer them to $\Omega(P)$ and $\Omega(O)$, respectively. Set $\rho(\Psi(P)) = \rho(\Psi(O)) = \phi$.

3 Related Work

3.1 Intrusion Response

Frameworks have previously been proposed for adding response capabilities. DCA [Fisch96] introduced a taxonomy for response and a tool to demonstrate the utility of the taxonomy. EMERALD's [Porras97] design allows customized responses to be invoked automatically, but does not define them by default. AAIR [Carver01] describes an expert system for response based on an extended taxonomy.

Our approach creates a framework for systematically choosing a response in real-time. This allows an attack to be contained automatically instead of being limited to raising an alarm, and does not require a new response subsystem to be developed for each new class of attack discovered.

3.2 Risk Management

Risk analysis has been utilized to manage the security of systems for several decades [FIPS31]. However, its use has been limited to offline risk computation and manual response. [SooHoo02] proposes a general model using decision analysis to estimate computer security risk and automatically update input estimates. [Bilar03] uses reliability modeling to analyze the risk of a distributed system. Risk is calculated as a function of the probability of faults being present in the system's constituent components. Risk management is framed as an integer linear programming problem, aiming to find an alternate system configuration, subject to constraints such as acceptable risk level and maximum cost for reconfiguration.

In contrast to previous approaches, we use the risk computation to drive changes in the operating system's security mechanisms. This allows risk management to occur in real-time and reduces the window of exposure.

The model described has been realized in a prototype, as described in [Gehani03]. It includes the following: an implementation of a state transition [Ilgun95] based intrusion detector; an extension of the Java access control subsystem to support predicated permissions; the ability to specify the exposure reduction resulting from each predicate's evaluation; a modification of the Java file input/output facilities to allow explicit guarantees about the confidentiality, integrity and availability of the data; the ability to explicitly associate costs for the loss of each of the security characteristics of any file; a risk manager that coordinates the other subsystems. The modified platform was able to respond and contain a suite of synthetic attacks against a web server running on it. This was expected since the configuration had been manually tuned. Further work is needed to automate the process of configuring the system.

4 Conclusion

This paper addresses the issue of modeling the detection and response process by equating the insecurity of a system with the risk it faces. The risk is defined formally in terms of known threats, exposures and consequences. Each threat is characterized using a host-based intrusion detection signature. Its likelihood is estimated by the current extent of the signature's match when a system is running. The system's exposure to a threat is a function of the permissions requested in the course of the attack and the auxiliary checks being performed before they are granted. The consequence of a threat is calculated as a function of the data that can lose confidentiality, integrity or availability when the attack is successful. The products of each threat's likelihood, the system's exposure to it and its consequence, contribute additively to form the total known risk of the host.

Two types of responses are utilized. One type performs further runtime checks before granting specific permissions, while the other allows disabling of transparent decryption, signing of modifications and remote replication of specific protected files.

Intrusion response is framed as maintaining the risk below a threshold of tolerance with a simultaneous goal of minimizing the impact of the selected responses on runtime performance. This requires minimizing a particular linear objective function with a quadratic constraint, which we have shown is NP-hard. Given that the response must be chosen in real-time, an optimal choice is ruled out. Instead, a greedy heuristic is used. It allows the first response primitive to be chosen in constant time, which is frequently sufficient to disrupt an attack. The remaining primitives, required to adequately manage the risk, are selected in quadratic time.

References

[Bilar03] Daniel Bilar, Quantitative Risk Analysis of Computer Networks, PhD thesis, Dartmouth College, 2003.

[Carver01] Curtis Carver, Adaptive, Agent-based Intrusion Response, PhD thesis, Texas A and M University, 2001.

[CSRMMBW88] Proceedings of the 1st Computer Security Risk Management Model Builders Workshop, Martin Marietta, Denver, Colorado, National Bureau of Standards, May 1988.

[CSRMMBW89] Proceedings of the 2nd Computer Security Risk Management Model Builders Workshop, AIT Corporation, Ottawa, Canada, National Institute of Standards and Technology, June 1989.

[CSRMMBW90] Proceedings of the 3rd International Computer Security Risk Management Model Builders Workshop, Los Alamos National Laboratory, Santa Fe, New Mexico, National Institute of Standards and Technology, August 1990.

[CSRMMBW91] Proceedings of the 4th International Computer Security Risk Management Model Builders Workshop, University of Maryland, College Park, Maryland, National Institute of Standards and Technology, August 1991.

[FIPS31] Guidelines for Automatic Data Processing Physical Security and Risk Management, National Bureau of Standards, 1974.

[FIPS65] Guidelines for Automatic Data Processing Risk Analysis, National Bureau of Standards, 1979.

[Fisch96] Eric Fisch, Intrusive Damage Control and Assessment Techniques, PhD thesis, Texas A and M University, 1996.

[Fletcher95] Sharon Fletcher et al, Software System Risk Management and Assurance, Proceedings of the New Security Paradigms Workshop, August 1995.

[Garey79] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[Gehani03] Ashish Gehani, Support for Automated Passive Host-based Intrusion Response, PhD thesis, Duke University, 2003.

[Harrison76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman, Protection in operating systems, Communications of the ACM, 19(8):461 471, August 1976.

[NIST800-12] Guidelines for Automatic Data Processing Physical Security and Risk Management, National Institute of Standards and Technology, 1996.

[NIST91] Description of Automated Risk Management Packages That NIST/NCSC Risk Management Research Laboratory Has Examined, National Institute of Standards and Technology, 1991.

[Peeters03] Rene Peeters, The maximum edge biclique problem is NP-complete, Discrete Applied Mathematics, Volume 131, Issue 3, p651-654, 2003.

[Ilgun95] Koral Ilgun, Richard A. Kemmerer and Phillip A. Porras, State Transition Analysis: A Rule-Based Intrusion Detection Approach, IEEE Transactions on Software Engineering, 21(3), pp. 181-199, March 1995.

[Kogan04] Uriel Feige and Shimon Kogan, Hardness of Approximation of the Balanced Complete Bipartite Subgraph Problem, Technical Report MCS04-04, Department of Computer Science and Applied Math, Weizmann Institute of Science, May 2004.

[Porras97] P.A. Porras and P.G. Neumann, EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances, Proceedings of the Nineteenth National Computer Security Conference, p353-365, Baltimore, Maryland, 22-25 October 1997.

[SooHoo02] Kevin Soo Hoo, Guidelines for Automatic Data Processing Physical Security and Risk Management, PhD Thesis, Stanford University, 2002.

Turku Centre for Computer Science
TUCS General Publications

1. **Joakim von Wright, Jim Grundy and John Harrison (Eds.)**, Supplementary Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics: TPHOLs'96
2. **Mikko Ruohonen and Juha Pärnistö (Eds.)**, Proceedings of the First European Doctoral Seminar on Strategic Information Management
3. **Christer Carlsson (Eds.)**, Exploring the Limits of Support Systems
4. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi and Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1996
5. **Wolfgang Weck, Jan Bosch, Clemens Szyperski (Eds.)**, Proceedings of the Second International Workshop on Component–Oriented Programming (WCOP '97)
6. Working Material from the School on Natural Computation, SNAC
7. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi, Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1997
8. **Reima Suomi, Paul Jackson, Laura Hollmén and Mats Aspnäs (Eds.)**, Teleworking Environments, Proceedings of the Third International Workshop on Telework
9. **Robert Fullér**, Fuzzy Reasoning and Fuzzy Optimization
10. **Wolfgang Weck, Jan Bosch, Clemens Szyperski (Eds.)**, Proceedings of the Third International Workshop on Component–Oriented Programming (WCOP '98)
11. Abstracts from the 10th Nordic Workshop on Programming Theory (NWPT'98)
12. **Edward M. Roche, Kalle Kangas, Reima Suomi (Eds.)**, Proceedings of the IFIP WG 8.7 Helsinki Working Conference, 1998
13. **Christer Carlsson and Franck Tétard (Eds.)**, Intelligent Systems and Active DSS, Abstracts of the IFORS SPC-9 Conference
14. **Mats Aspnäs, Ralph-Johan Back, Timo Järvi, Martti Kuutti, Tiina Lehto (Eds.)**, Turku Centre for Computer Science, Annual Report 1998
15. **Tero Harju and Iiro Honkala (Eds.)**, Proceedings of the Seventh Nordic Combinatorial Conference
16. **Christer Carlsson (Editor)**, The State of the Art of Information System Applications in 2007
17. **Christer Carlsson (Editor)**, Information Systems Day
18. **Ralph-Johan Back, Timo Järvi, Nina Kivinen, Leena Palmulaakso-Nylund and Thomas Sund (Eds.)**, Turku Centre for Computer Science, Annual Report 1999
20. **Reima Suomi, Jarmo Tähkäpää (Eds.)**, Health and Wealth trough Knowledge
21. **Johan Lilius, Seppo Virtanen (Eds.)**, TTA Workshop Notes 2002
22. **Mikael Collan**, Investment Planning – An Introduction
23. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi, Nina Kivinen, Maria Prusila, Thomas Sund (Eds.)**, Turku Centre for Computer Science, Annual Report 2000-2001
24. **Ralph-Johan Back and Victor Bos**, Centre for Reliable Software Technology, Progress Report 2003
25. **Pirkko Walden, Stina Störling-Sarkkila, Hannu Salmela and Eija H. Karsten (Eds.)**, ICT and Services: Combining Views from IS and Service Research
26. **Timo Järvi and Pekka Reijonen (Eds.)**, People and Computers: Twenty-one Ways of Looking at Information Systems
27. **Tero Harju and Juhani Karhumäki (Eds.)**, Proceedings of WORDS'03
28. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2002
29. **João M. Fernandes, Johan Lilius, Ricardo J. Machado and Ivan Porres (Eds.)**, Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software
30. **Mats Aspnäs, Christel Donner, Monika Eklund, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2003
31. **Andrei Sabelfeld (Editor)**, Foundations of Computer Security

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics

Åbo Akademi University

- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration

- Institute of Information Systems Sciences

ISBN 952-12-1372-8
ISSN 1239-1905