Preface

The present report contains the proceedings of the first international *Workshop on Formal Aspects in Security and Trust (FAST) 2003*, held in Pisa, September 8-9 2003. FAST is affiliated to the 12^{th} International Formal Methods Europe Symposium (FME'03).

FAST aims to contribute to the aggregation of researchers in the areas of security and trust. The new challenges offered by the so-called ambient intelligence space, as a future paradigm in the information society, demand for a coherent and rigorous framework of concepts, tools and methodologies to provide user's trust&confidence on the underlying communication/interaction infrastructure. It is necessary to address issues relating to both guaranteeing security of the infrastructure and the perception of the infrastructure being secure. In addition, user confidence on what is happening must be enhanced by developing trust models effective but also easily comprehensible and manageable by users.

FAST sought for original papers focusing of formal aspects in: security and trust policy models; security protocol design and analysis; formal models of trust and reputation; logics for security and trust; distributed trust management systems; trust-based reasoning; digital assets protection; data protection; privacy and ID issues; information flow analysis; language-based security; security and trust aspects in ubiquitous computing; validation/analysis tools; web service security/trust/privacy; GRID security; security risk assessment; case studies ....

This proceedings consists of 11 regular papers plus 3 short papers selected out of 31 submissions (1 withdrawn). A few selected papers will be invited for possible publication on a special issue of Journal of *Theoretical Computer Science*.

We wish to thank the invited speakers for contributing to set up a very interesting program, the PC members for their valuable efforts in properly evaluating the submissions, the FME organizers for accepting FAST as an affiliated event and Adriana Lazzaroni and her collaborators in Pisa for the local organization.

Thanks are also due to BITD-CCLRC and IIT-CNR for the financial support for organizing FAST, in particular for making available some grants for attending FAST to Ph.D. students and young researchers.

September 2003

Theo Dimitrakos and Fabio Martinelli
FAST 2003 Organization co-Chairs

Organization

Workshop Organizers

Theo Dimitrakos, BITD-CLRC
Fabio Martinelli, IIT-CNR

Invited Speakers

Andy Gordon, Microsoft Research
Pierpaolo Degano, University of Pisa

Program Committee

Patrizia Asirelli, INTAS
Bjørn Axel Gran, IFE
Elisa Bertino, University of Milan
David W. Chadwick, University of Salford
Pravir Chawdhry, IPSC-JRC
Theo Dimitrakos, BITD-CCLRC
Sandro Etalle, University of Twente
Simon Foley, University College Cork
Andrew Jones, King's College (London)
Emil Lupu, Imperial College (London)
Heiko Mantel, DFKI
Fabio Martinelli, IIT-CNR
Brian Matthews, BITD-CCLRC
Catherine Meadows, NRL
Paddy Nixon, University of Strathclyde
Peter Ryan, University of Newcastle
Andrei Sabelfeld, Cornell University
Babak Sadighi Firozabadi, SICS
Pierangela Samarati, University of Milan
Ketil Stølen, SINTEF

Local Organization

Beatrice Lami, IIT-CNR
Adriana Lazzaroni, IIT-CNR (chair)
Marinella Petrocchi, IIT-CNR

Table of Contents

A Logical Model for Security of Web Services

Hristo Koshutanski and Fabio Massacci

Dip. di Informatica e Telecomunicazioni - Univ. di Trento
via Sommarive 14 - 38050 Povo di Trento (ITALY)
{hristo,massacci}@dit.unitn.it

Abstract. Business Processes for Web Services are the new paradigm for the lightweight integration of business from different enterprises.
Yet, there is not a comprehensive proposal for a logical framework for access control for business processes though logics for access control policies for basic web services are well studied.
In this paper we propose a logical framework for reasoning (deduction, abduction, consistency checking) about access control for business processes for web services.

1 Introduction

In the past millennium the development of middleware connected the IT efforts to integrate distributed resources of a single enterprise. The new century has seen the rise of a new IT concept: virtual enterprise. Virtual enterprise is born when a business process is no longer closed within the boundary of a single corporation. The business process is thus composed by partners that offer their services on the web and integrate each other efforts into one (hopefully) coherent process.

The scenario offered by business processes for web services is particularly challenging for the definition of its security features. Indeed it has some aspects of trust management systems and some aspects of workflow security management.

From the trust management systems (see e.g. [13, 7]) it takes the credential-based view: a (web) service is offered on its own and the decision to grant or deny access can only be made on the basis of the credentials sent by the client.

From workflow authorization systems (see e.g. [2, 9]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permission to users according to the least privilege principles. In contrast with workflow security management schemes a business process for web services crosses organizational boundaries and is provided by entities that sees each other as just partners and nothing else. We have something even more loosely coupled than federated databases.

So, it is not a surprise that there is not a comprehensive proposal for a logical framework that tackles these aspects, though logics for access control policies for basic web services, workflows, and distributed systems are well studied.

We identify a number of differences w.r.t. traditional access control:

– credential vs user based access control,

– interactive vs one-off evaluation of credentials (i.e., controlled disclosure of information vs all-or-nothing decision),
– on-line vs off-line analysis of consistency of roles and users assignments (e.g., for separation of duties),

In this paper we propose a logical framework for reasoning about access control for business processes for web services. We identify the different reasoning tasks (deduction, abduction, consistency checking) that characterize the problem and clarify the problems of temporal evolution of the logical model (addition and revocation of credentials).

2 The Formal Framework

Our formal model for reasoning on access control is based variants of Datalog with the stable model semantics and combines in a novel way a number of features:

– logics for trust management by Li et al. [11];
– logic for workflow access control by Bertino et al. [2];
– logic for release and access control by Bonatti and Samarati [3].

We consider the view of the single partner, as we cannot assume sharing of policies between partners. In [10] it is explained how the entire process can be orchestrated by using "mobile" business processes, while keeping each partner policy decision process as a black-box.

In our framework each partner has a *security policy for access control* $\mathcal{P}_{\mathcal{A}}$ and a *security policy for interaction control* $\mathcal{P}_{\mathcal{I}}$, whose syntax will be defined later in §3.

The policy for access control is used for making decision about usage of all web services offered by the partner. The policy for interaction control is used to decide which credentials must be additionally provided or must be revoked by the user if those available are not adequate to obtain the service.

In many workflow authorization schemes, the policy is not sufficient to make an access control decisions and thus we need to identify the *history of the execution* $\mathcal{H}$ of the business process as perceived by the current partner, and a set of *active (unrevoked) credentials* $\mathcal{C}_{\mathcal{A}}$ that have been presented by the agent in past requests to other services comprised in the same business process.

To execute a service of the fragment of business process under the control the partner the user will submit a set of *presented credentials* $\mathcal{C}_{\mathcal{P}}$, a set of *revoked credential* $\mathcal{C}_{\mathcal{R}}$ and a *service request* $\mathcal{R}(.)$ We assume that $\mathcal{C}_{\mathcal{R}}$ and $\mathcal{C}_{\mathcal{P}}$ are disjoint.

To specify how the access control decision is made we now assume the usual inference capabilities, that is, for any set of formulae (Datalog rules and facts) $\mathcal{F}$ and any formula f:

deduction determines whether f is a logical consequence of $\mathcal{F}$, $\mathcal{F} \models f$;
consistency determines whether $\mathcal{F}$ is consistent, $\mathcal{F} \not\models \bot$;

abduction given an additional set of atoms $\mathcal{A}$ called the abducible atoms, and a partial order relation $\prec$ between subsets of A determine a set of atoms $\mathcal{E} \subset \mathcal{A}$ such that (i) f is a logical consequence of $\mathcal{F}$ and $\mathcal{E}$, namely $\mathcal{F} \cup \mathcal{E} \models f$, (ii) adding $\mathcal{E}$ to $\mathcal{F}$ does not generate an inconsistency, namely $\mathcal{F} \cup \mathcal{E} \not\models \bot$, and finally (iii) $\mathcal{E}$ is a $\prec$-minimal subset of $\mathcal{A}$ having this property (See further Def. 1).

For an introduction to abduction see Shanahan [12], for a survey of complexity results - Eiter et al. [6]. We shall see later on in section 5 how abduction is used.

3 Logical Syntax

For the syntax we build upon [2, 3, 11]. We have a four disjoint sets of constants, one for users identifiers denoted by $\mathsf{User}:U$, one for roles $\mathsf{Role}:R$, one for services $\mathsf{WebServ}:S$, and finally one for keys that are used to certify credentials $\mathsf{Key}:K$.

We assume that we have the following security predicates:

$\mathsf{Role}:R_i \succ \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates in the global role hierarchy role $\mathsf{Role}:R_j$.

$\mathsf{Role}:R_i \succ_{\mathsf{WebServ}:S} \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates in the local role hierarchy for service $\mathsf{WebServ}:S$ role $\mathsf{Role}:R_j$.

$\mathsf{assign}\,(P, \mathsf{WebServ}:S)$ when an access to the service $\mathsf{WebServ}:S$ is granted (assigned) to P. Where P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

$\mathsf{forced}\,(P, \mathsf{WebServ}:S)$ if the predicate is true then an access right to access the service $\mathsf{WebServ}:S$ must be given (forced) to P. Where P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

We have three type of predicates for credentials:

$\mathsf{declaration}\,(\mathsf{User}:U)$ it is a statement declared by the $\mathsf{User}:U$ for its identity.

$\mathsf{credentialID}\,(\mathsf{Key}:K, \mathsf{User}:U, \mathsf{Role}:R)$ it is a statement declared and signed by $\mathsf{Key}:K$ corresponding to some trusted authority that $\mathsf{User}:U$ has activated $\mathsf{Role}:R$.

$\mathsf{credentialTask}\,(\mathsf{Key}:K, \mathsf{User}:U, \mathsf{WebServ}:S)$ it is a statement declared and signed by $\mathsf{Key}:K$ corresponding to some trusted authority that $\mathsf{User}:U$ has the right to access $\mathsf{WebServ}:S$.

Three type of predicates describing the current status of each service:

$\mathsf{running}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ is executed by P.

$\mathsf{abort}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ within a workflow aborts.

$\mathsf{success}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ within a workflow successfully executes.

Furthermore, for some additional workflow constraints we need to have some meta-level predicates that specify how many statements are true. We use here a notation borrowed from Niemelä smodels system, but we are substantially using the count predicates defined by Das [4]:

$n \leq \{X.\, Pr\}$ where n is a positive integer, X is a set of variables, and Pr is a predicate, so that intuitively $n \leq \{X.\, Pr\}$ is true in a model if at least n instances of the grounding of X variables in Pr are satisfied by the model. The $\{X.\, Pr\} \leq n$ is the dual predicate.

4 Formal Rules and Semantics

Normal logic programs [1] are sets of *rules* of the form:

$$A \leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{1}$$

where A, B_i and C_i are (possibly ground) predicates among those listed in Sec.3 A is called the *head* of the rule, each B_i is called a *positive literal* and each $not\, C_j$ is a *negative literal*, whereas the conjunction of the B_i and $not\, C_j$ is called the *body* of the rule. A normal logic program is a set of rules.
In our framework, we also need *constraints* that are rules with an empty head.

$$\leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{2}$$

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [8] (see also [1] for an introduction). The intuition is to interpret the rules of a program P as constraints on a solution set S (a set of ground atoms) for the program itself. So, if S is a set of atoms, a rule 1 is a constraint on S stating that if all B_i are in S and none of C_j are in it, then A must be in S. A constraint 2 is used to rule out from the set of acceptable models the situation in which B_i are true and all C_j are false is not acceptable.

Definition 1 (Abduction). *Let P be a logic program, H be a set of predicates (called hypothesis, or abducibles), L be a (positive or negative) ground literal (sometimes called the manifestation), and $\prec$ a p.o. over subsets of H, the* cautious solution *of the abduction problem is a set of ground atoms E such that*

1. *E is a set ground instances of predicates in H,*
2. *$P \cup E \models L$*
3. *$P \cup E \not\models \bot$*
4. *any set $E' \prec E$ does not satisfy all three conditions above*

Remark 1. The choice of the partial order has a major impact in presence of complex role hierarchies. The "intuitive" behavior of the abduction algorithm for what regards the extraction of the minimal set of security credentials is not guaranteed by the straightforward interpretation of H as the set of credentials and by the set cardinality or set containment as minimality orderings.

To understand the problem consider the following logic program:
$\mathsf{assign}\,(\mathsf{User}:U, \mathsf{WebServ}:ws) \leftarrow \mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:U, \mathsf{Role}:R),$
$\mathsf{Role}:R \succ \mathsf{Role}:r_1$

$\mathsf{Role}:r_2 \succ \mathsf{Role}:r_1 \leftarrow$

The request $\mathsf{assign}\,(\mathsf{User}:fm, \mathsf{WebServ}:ws)$ has two $\subseteq$-minimal solutions:
$\{\mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:fm, \mathsf{Role}:r_1)\}$, $\{\mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:fm, \mathsf{Role}:r_2)\}$
Yet, our intuition is that the first should be the minimal one.

So, we need a more sophisticated partial order. For example, if $E \preceq E'$ is such that for all credentials $c \in E$ there is a credential $c' \in E'$ where $c = c'$. We can revise it so that $E \prec E'$ if $c \in E$ there is a credential $c' \in E'$ where c' is identical to c except that it contains a role R' that dominates the corresponding role R in c. This p.o. generates the "intuitive" behavior of the abduction algorithm.

Definition 2. *An access control policy $\mathcal{P}_\mathcal{A}$ is a logic program over the predicates defined in Sec. 3 in which (i) no credential, no role hierarchy atom, and no execution atom can occur in the head of a rule and (ii) for every rule containing and head A which is the (possibly ground instance of) predicate* $\mathsf{forced}\,(P, \mathsf{WebServ}:S)$ *there is the (possibly ground instance of) rule* $\mathsf{assign}\,(P, \mathsf{WebServ}:S) \leftarrow$
$\mathsf{forced}\,(P, \mathsf{WebServ}:S)$.

An access control request is a ground instance of an $\mathsf{assign}\,(i, \mathsf{WebServ}:k)$ *predicate.*

The request r is a security consequence of a policy $\mathcal{P}_\mathcal{A}$ if (i) the policy is logically consistent and (ii) the request is a logical consequence of the policy.

Definition 3. *An interaction policy $\mathcal{P}_\mathcal{I}$ is a logic program in which no credential and no role hierarchy atom can occur in the head of a rule.*

Definition 4 (Fair Access). *Let $\mathcal{P}_\mathcal{A}$ be an access control policy, let $\mathcal{C}_\mathcal{D}$ be the set of ground instances of credentials occurring in $\mathcal{P}_\mathcal{A}$, and let $\prec$ be a p.o. over subsets of $\mathcal{C}_\mathcal{D}$. The access control policy guarantees* $\prec$-fair access *if for any ground request r that is an instance of a head of a rule in $\mathcal{P}_\mathcal{A}$ there exists a set $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ that is a solution of the abduction problem.*

Definition 5 (Fair Interaction). *Let $\mathcal{P}_\mathcal{A}$, and $\mathcal{P}_\mathcal{I}$ be, respectively, an access control policy and an interactive policy, and let $\mathcal{C}_\mathcal{D}$ be the set of ground instances of credentials occurring in $\mathcal{P}_\mathcal{A}$, and let $\prec$ be a p.o. over subsets of $\mathcal{C}_\mathcal{D}$. The policies guarantee* $\prec$-fair interaction *w.r.t. a set of credentials $\mathcal{C}_\mathcal{I}$ if (i) $\mathcal{P}_\mathcal{A}$ guarantees $\prec$-fair access and (ii) for any solution of the abduction problem $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ and any credential $c \in \mathcal{C}_\mathcal{M}$ if it $\mathcal{P}_\mathcal{I} \cup \mathcal{C}_\mathcal{I} \models c$. If the set of initial credentials $\mathcal{C}_\mathcal{I}$ only contains declarations then the access control is* unlimited.

5 The Formal Framework: Reasoning

To allow for an easier grasp of the problem we start with a basic framework.
Traditional Access Control. This approach is the cornerstone of most logical formalization [5].

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \models r$
2. if the check succeeds then grant access else deny access

A number of works has deemed such blunt denial unsatisfactory and therefore it has been proposed by Bonatti and Samarati [3] and Yu et al. [14] to send back to the client some of the rules that are necessary to gain additional access.
Disclosable Access Control. It is revised to allow for the flow of rules and information to users:

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
2. if the check succeeds then access is granted, otherwise select some rule $r \leftarrow p \in \mathcal{P}_{\mathcal{A}}$ and send the rule back to the client

In many cases, this solution is neither sufficient nor desirable. For instance if the policy is not flat, it has constraints on the credentials that can be presented at the same time (e.g., separation of duties) or a more complex role structure is used, these systems would not be complete. Also repeated queries allow for the disclosure of the entire policy, which might well be undesirable. In this case we need the interactive access control solution for Web Services proposed by Koshutanski and Massacci [10] that is described below.

Interactive Access Control for Stateless WS.

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
2. if the check succeeds then access is granted, otherwise
   (a) compute the set of *disclosable credentials* as $\mathcal{C}_{\mathcal{D}} = \{c | c \text{ credential and } \mathcal{P}_{\mathcal{I}} \cup \mathcal{C}_{\mathcal{P}} \models c\}$
   (b) use abduction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}}$ such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$
   (c) if no such set exists then $\bot$ is sent back to the user,
   (d) otherwise the set of missing credentials $\mathcal{C}_{\mathcal{M}}$ is send back to the client and the process re-iterates.

This type of decision is characteristic of most logical approaches to access control [11, 2, 3]: we only look at the policy, the request and the set of credentials.

If the authorization decisions of business processes are stateful, and the the corresponding workflow of the partners has constraints on the execution of future services on the basis of past services, then even this solution is not adequate enough. As we already noted, the problems are the following:

- the request may be inconsistent with some role that the user has taken up in the past;
- the new set of credential may be inconsistent with requirements such as separation of duties;

So, this means that we must have some roll-back procedure by which, if the user has by chance sent the "wrong" credentials, he has some revocation mechanism to drop them.

Access Control for Stateful Business Processes. At this stage, we need all the policy and set of credentials that we have envisaged and indeed the partner expects from the client the set of current credentials $\mathcal{C}_{\mathcal{P}}$ plus the set of revoked $\mathcal{C}_{\mathcal{R}}$. The (logical) access control decision takes the following steps:

1. remove the revoked credentials from the set of active credentials, namely $\mathcal{C}_\mathcal{A} \leftarrow \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{A} \setminus \mathcal{C}_\mathcal{R}$,
2. verify the consistency of the request with the active set of credentials and the history of execution, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{A} \cup \{r\} \not\models \bot$
3. If this check succeeds goes to the next step, otherwise
   (a) derive a subset of *excessing credentials* that must be revoked by the user $\mathcal{C}_\mathcal{E} \subseteq \mathcal{C}_\mathcal{A}$ such that the set $\mathcal{C}_\mathcal{E}$ is minimal w.r.t. the $\prec$ partial order and that by removing it from $\mathcal{C}_\mathcal{A}$ the consistency check would succeed
   (b) if no such set exists then $\bot$ is sent back to the user
   (c) if it exists, this set is send back to the user and the process is re-iterated.
4. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{A} \models r$,
5. if this check succeeds then access is granted
6. if the step fails
   (a) compute the set of *disclosable credentials* as $\mathcal{C}_\mathcal{D} = \{c | c \text{ credential and } \mathcal{P}_\mathcal{I} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{A} \models c\}$
   (b) use abduction to find a minimal set of missing credentials $\mathcal{C}_\mathcal{M}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{A} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{A} \cup \mathcal{C}_\mathcal{M} \not\models \bot$
   (c) if this set exists then $\mathcal{C}_\mathcal{M}$ is send back to the client and the process re-iterates.
   (d) if it does not exists then
      i. generalize the set of disclosable credentials to all credentials occurring in $\mathcal{P}_\mathcal{A}$
      ii. use adbuction to find a minimal set of missing credentials $\mathcal{C}_\mathcal{M}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{M} \not\models \bot$
      iii. if no such set does exist then $\bot$ is sent back to the user,
      iv. if such set do exists then compute the set of revocable credentials $\mathcal{C}_\mathcal{E}$ as the set $\mathcal{C}_\mathcal{A} \setminus \mathcal{C}_\mathcal{M}$, return this set to the client and re-iterate the process

When the request is granted the appropriate grounding of suitable history predicates are added to $\mathcal{H}$.

Remark 2. The step 3a looks the opposite of abduction: rather than adding new information to derive more things (the request), we drop information to derive less things (the inconsistency). It is possible to show that by adding a number of rules linear in the number of potentially revocable credentials the two task are equivalent.

Theorem 1. *If an access control policy guarantees $\prec$-fair access and $\mathcal{H} = \emptyset$ the access control algorithm for stateful business processes never returns $\bot$.*

Theorem 2. *If access and interaction control policies guarantee $\prec$-fair interaction w.r.t. a set of credentials $\mathcal{C}_\mathcal{I}$ and $\mathcal{H} = \emptyset$ the access control algorithm for stateful business processes, there exists a sequence of revocable and missing credentials starting with $\mathcal{C}_\mathcal{I}$ such that the access control algorithm for stateful business processes eventually grant r.*

6 Conclusions

In this paper we proposed a logical framework for reasoning about access control for business processes for web services. Our formal model for reasoning on access control is based variants of Datalog with the stable model semantics and combines in a novel way a number of features: the logic for trust management by Li et al. [11]; the logic for workflow access control by Bertino et al. [2]; and the logic for controlling the release of information by Bonatti and Samarati [3]. We identified the different reasoning tasks (deduction, abduction, consistency checking) that characterize the problem and clarify the problems of temporal evolution of the logical model (updates and downdates of credentials).

References

1. Apt, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.
2. Bertino, E., Ferrari, E., and Atluri, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (1999), 65–104.
3. Bonatti, P., and Samarati, P. A unified framework for regulating access and information release on the Web. *Journal of Computer Security*. (to appear).
4. Das, S. *Deductive Databases and Logic Programming*. Addison-Wesley, Reading, MA, 1992.
5. di Vimercati, S. D. C., and Samarati, P. Access control in federated systems. In *Proceedings of the 1996 workshop on New security paradigms* (1996), ACM Press, pp. 87–99.
6. Eiter, T., Gottlob, G., and Leone, N. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science 189*, 1-2 (1997), 129–177.
7. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., and Ylonen, T. *SPKI Certificate Theory*, September 1999. IEFT RFC 2693.
8. Gelfond, M., and Lifschitz, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88)* (1988), R. Kowalski and K. Bowen, Eds., MIT-Press, pp. 1070–1080.
9. Kang, M. H., Park, J. S., and Froscher, J. N. Access control mechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 66–74.
10. Koshutanski, H., and Massacci, F. An access control system for business processes for Web services. Tech. Rep. DIT-02-102, Department of Information and Communication Technology, University of Trento, 2002.
11. Li, N., Grosof, B. N., and Feigenbaum, J. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 128–171.
12. Shanahan, M. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.
13. Weeks, S. Understanding trust management systems. In *IEEE SS&P-2001* (2001).
14. Yu, T., Winslett, M., and Seamons, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 1–42.

Analysing Topologies of Transitive Trust

Audun Jøsang[1], Elizabeth Gray[2], and Michael Kinateder[3]

[1] Distributed Systems Technology Centre[†]
Queensland University of Technology, Brisbane, Australia
Email: ajosang@dstc.edu.au
[2] Computer Science Department[‡]
Trinity College Dublin, Ireland
Email: Liz.Gray@cs.tcd.ie
[3] Institute of Parallel and Distributed Systems [§]
Faculty of Computer Science, University of Stuttgart, Germany
Email: Michael.Kinateder@informatik.uni-stuttgart.de

Abstract. Transacting and interacting through computer networks makes it difficult to use traditional methods for establishing trust between parties. Creating substitutes by which people, organisations and software agents can derive trust in others through computer networks requires computerised analysis of trust topologies. This paper describes diverse dimensions of trust that are needed for analysing trust topologies, and provides a notation with which to express trust relationships in terms of these dimensions. The result is a simple way of specifying topologies of trust from which derived trust relationships can be automatically and securely computed.

1 Introduction

Modern communication media are increasingly removing us from familiar styles of interacting and doing business which both rely on some degree of trust between the interaction or business partners. Moreover most traditional cues for assessing trust in the physical world are not available through those media. We may now be conducting business with people and organisations of which we know nothing, and we are faced with the difficult task of making decisions involving risk in such situations. As a result the topic of trust in open computer networks is receiving considerable attention in the network security community and e-commerce industry [1–4]. State of the art technology for stimulating trust in e-commerce includes cryptographic security mechanisms for providing confidentiality of communication and authentication of identities. However, merely having a cryptographically certified identity or knowing that the communication

[†] The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).
[‡] This work has been supported in part by the SECURE project (IST-2001-32486), a part of the EU FET Global Computing initiative.
[§] Part of this work has been funded by Hewlett-Packard Limited.

channel is encrypted is not enough for making informed decisions if no other knowledge about a remote transaction partner is available. Trust therefore also applies to for example the reliability, honesty and reputation of transaction partners.

Being able to formally express and reason with these types of trust is needed in order to create substitutes for the methods we use in the physical world, and also for creating new methods for determining trust in electronic environments. The aim of this will be to create communication infrastructures where trust can thrive in order to ensure meaningful and mutually beneficial interactions between players.

In this regard, we intend to describe a notation for specifying topologies of transitive trust, and to discuss ways to reason about trust in such topologies. We first consider properties of trust: diversity, transitivity, and combination. We then propose a notation for describing and reasoning about trust, and illustrate how this notation may successfully and securely be used to correctly analyse different trust scenarios. Finally, we identify several requirements that trust measures and operators should satisfy.

2 Trust Diversity

Humans use trust to facilitate interaction and accept risk in situations where complete information is unavailable. However, trust is a complex concept that is difficult to stringently define. A wide variety of definitions of trust have been put forward [5], many of which are dependent on the context in which interaction occurs, or on the observer's subjective point of view. Deutsch's definition of trust is commonly used as a starting point for understanding:

> If an individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial (Va^+) or to an event perceived to be harmful (Va^-); he perceives that the occurrence of (Va^+) or (Va^-) is contingent on the behaviour of another person; and he perceives that the strength of (Va^-) to be greater than the strength of (Va^+). If he chooses to take an ambiguous path with such properties, I shall say he makes a trusting choice; if he chooses not to take the path, he makes a distrustful choice. [6]

While Deutsch breaks trust down further into several different circumstances in which a trusting choice might be made, he concentrates on the fact that trust "is strongly liked to confidence in, and overall optimism about, desirable events taking place." [7]

A similar description of trust has been expressed by McKnight and Chervany and can be summarised as follows:

> Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible. [5]

This definition illustrates that non-living material or abstract things can also be trusted although they do not have a free will to behave honestly or dishonestly in the way living persons do. McKnight and Chervany also separate between different trust constructs, including *trusting behaviour* which expresses the act of entry into a situation

of dependence, *trusting intention* which is only the intention to do so, and *system trust* which denotes trust in "impersonal structures", either material or abstract.

Thus, we may say that trust is related to belief in the honesty, reliability, competence, willingness, etc. of the trusted entity, it being a person, organisation, system. Trust can also be related to a particular property of material or abstract objects such as a computer system or our legal institutions. Despite this variation in meanings, many researchers simply use and assume a definition of trust in a very specific way, such as *a trusted public key* which refers to the authenticity of that key.

The repeated uses of the word "perceives" in Deutsch's definition implies that trust is a subjective quality individuals place in one another. Additionally, the fact that different entities can have different kinds of trust in the same target entity indicates that trust is subjective. It is also important to notice that trust is related to the purpose and nature of the relationship, e.g. an organisation trusts an employee to deal with financial transactions up to a specific amount, but not above, and that same employee might not be trusted to make public statements about the organisation.

In order for trust to form topologies it needs to be expressed with three basic diversity dimensions [8] where the first dimension represents the trustor or trust origin, the second represents the trust purpose, and the third represents the trustee or the trust target. This is illustrated in Fig. 1 below.

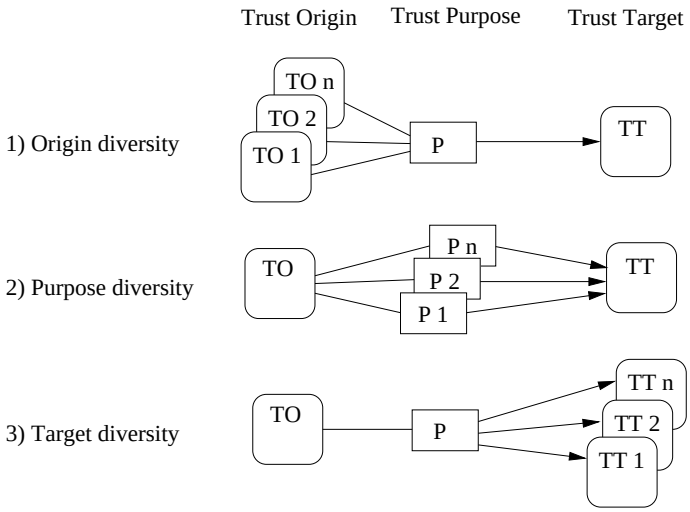

Fig. 1. Basic trust diversity

In addition to the three basic topology dimensions a *measure* can be associated with each trust relationship. The trust measure could for example be binary (trusted, not trusted), discrete (e.g. strong trust, weak trust, strong distrust, weak distrust, etc.)

or continuous in some form (e.g. probability, percentage or belief functions of trustworthiness). The topic of expressing and computing trust measures will be discussed in Sec. 7.

In addition, a fifth important element to a trust relationship is its *time* component. Quite obviously trust of the trustor in the trustee regarding a certain purpose at one point in time might be quite different from this trust after several transactions between these two entities have taken place. This means, that we can model time as a set of discrete events taking place, where both entities trustor and trustee are involved. However, even if no transactions take place, a trust relationship will gradually change with time passing. Therefore, in addition to the discrete changes that are made when events have occurred, we must also take into account continuous changes to trust relationships.

3 Trust Transitivity

It has been shown [9] that trust is not implicitly transitive. However, a recommendation system can be used to allow trust transitivity according to explicit conditions.

Trust transitivity means, for example, that if Alice trusts Bob who trusts Clark then Alice will also trust Clark. This assumes that Bob actually tells Alice that he trusts Clark, and this will typically happen in a recommendation. In this simple example the trust origins and trust targets are easily identifiable, but it does not say anything specific about the trust purposes.

Let us assume that Alice needs to have her car serviced, so she asks Bob for his advice about where to find a good car mechanic in town. Bob is thus trusted by Alice to know about a good car mechanic and to tell his honest opinion about that, whereas Clark is trusted by Bob to be a good car mechanic.

Let us make the example slightly more complicated, wherein Bob does not actually know any car mechanics himself, but he knows Claire whom he believes knows a good car mechanic. As it happens, Claire is happy to recommend the car mechanic named David. The trust origins and targets are again explicit, but it is more tricky to define exactly for what purpose Alice now trusts Bob. The most obvious is to say that Alice trusts Bob to recommend somebody who can recommend a good car mechanic. The problem with this type of formulation is that the length of the trust purpose becomes proportional with the length of the transitive chain, so that the trust purpose rapidly becomes an impenetrable expression. It can be observed that this type of trust purpose has a recursive structure that can be exploited to define a more compact expression for the trust purpose. Trust in the ability to recommend represents indirect trust and is precisely what allows trust to become transitive. At the same time this trust always assumes the existence of a direct trust purpose at the end of the transitive path which in the example above is about being a good car mechanic.

This observation indicates that the trust purpose of the final leg must somehow be part of every leg in the trust path. We will express this by defining the two trust variants *indirect* and *direct* and let the *trust variant* be a parameter in every trust purpose.

Alice would then have *indirect* trust in Bob to be a good car mechanic, and similarly for Bob and Claire. This must be interpreted as saying that Alice trusts Bob to recommend somebody (to recommend somebody etc.) to be a good car mechanic. On

the other hand Claire would have *direct* trust in David to be a good car mechanic. The indirect variant of a trust purpose is recursive so that any transitive trust chain, with arbitrary length, can be expressed using only one trust purpose with two variants.

The examples above assume some sort of absolute trust between the agents in the transitive chain. In reality trust is never absolute, and many researchers have proposed to express trust as discrete verbal statements, as probabilities or other continuous measures. One observation which can be made from a human perspective is that trust is weakened or diluted through transitivity. By taking the example above, it means that Alice's trust in the car mechanic David through the recommenders Bob and Claire can be at most as strong as Claire's trust in David. This is illustrated in Fig. 2 below.

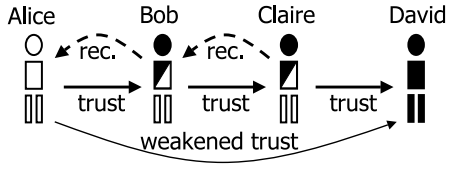

Fig. 2. Trust transitivity

By assuming Alice's trust in Bob and Bob's trust in Claire to be positive but not absolute, Alice's derived trust in David is intuitively weaker that Claire's trust in David.

It could be argued that negative trust in a transitive chain can have the paradoxical effect of strengthening the derived trust. Take for example the case where Bob distrusts Claire and Claire distrusts David whereas Alice trusts Bob. In this situation Alice might actually derive positive trust in David, since she relies on Bob's advice and Bob says: Claire is a cheater, do not rely on her. So the fact that Claire distrusts David might count as a pro-David argument from Alice's perspective. The question boils down to "is the enemy of my enemy my friend". However this question relates to how trust is computed and derived, and we will not go into further detail on this issue here.

We will use the symbol ":" to denote initial trust relationships and "$\overset{\rightarrow}{:}$" to denote derived trust relationships, so that the trust relationships of Fig. 2 can be expressed as:

$$\text{Alice} \overset{\rightarrow}{:} \text{David} = \text{Alice} : \text{Bob} : \text{Claire} : \text{David} \tag{1}$$

where the trust purpose is implicit. Let the trust purpose be defined as P_1; " *trusts X to be a good car mechanic* ". Let the direct variant be denoted by dP_1 and the indirect variant by iP_1. The trust topology of Fig.2 can then be explicitly expressed as:

$$\text{Alice}; [dP_1] \overset{\rightarrow}{:} \text{David} = \text{Alice}; [iP_1] : \text{Bob}; [iP_1] : \text{Claire}; [dP_1] : \text{David} \tag{2}$$

The idea of contstructing transitive trust chains based on a single trust purpose with direct and indirect variants is captured by the following definition.

Definition 1. *A valid transitive trust chain requires that every leg in the chain contains the same trust purpose and that every leg except the last is indirect.*

The transitive path stops when a leg is not indirect. It is of course possible for a principal to have both direct and indirect trust in another principal but that should be expressed as two separate trust legs. The existence of both a direct and an indirect trust leg e.g. from Claire to David should be interpreted as Claire having trust in David not only to be a good car mechanic but also to recommend somebody else for the job.

Let trust measures be denoted by μ_i where i refers to a specific trust measure, and let Alice, Bob and Claire's trust measures be μ_1, μ_2 and μ_3 respectively. Let time stamps be denoted by τ_j where j refers to a specific time, and let the trust measures be time stamped τ_1, τ_2 and τ_3 respectively. Alice's derived trust measure and time stamp are denoted by μ_4 and τ_4. The trust expression of Fig. 2 can then be expressed as:

$$\begin{aligned}\text{Alice};[dP_1,\mu_4,\tau_4]\overrightarrow{:}\text{David} = \text{Alice};[iP_1,\mu_1,\tau_1] : \text{Bob};[iP_1,\mu_2,\tau_2] \ : \\ \text{Claire};[dP_1,\mu_3,\tau_3] : \text{David}\end{aligned} \tag{3}$$

Claire obviously recommends to Bob her opinion about David as a car mechanic, but Bob's recommendation to Alice is ambiguous. It can either be that Bob passes Claire's recommendation unaltered on to Alice, or that Bob derives his own direct trust in David which he recommends to Alice. The latter way of passing recommendations can create problems and it is better when Alice receives Claire's recommendation unaltered. This will be discussed in more detail in Sec. 5.

4 Parallel Trust Combination

It is common to collect recommendations from several sources in order to be better informed when making decisions. This can be modelled as *parallel trust combination*.

Let us assume again that Alice needs to get her car serviced, and that she asks Bob to recommend a good car mechanic. When Bob recommends David, Alice would like to get a second opinion, so she asks Claire for her opinion about David. Intuitively, if both Bob and Claire recommend David as a good car mechanic, Alice's trust in David will be stronger than if she had only asked Bob. Parallel combination of positive trust thus has the effect of strengthening the derived trust. This is illustrated in Fig. 3 below.

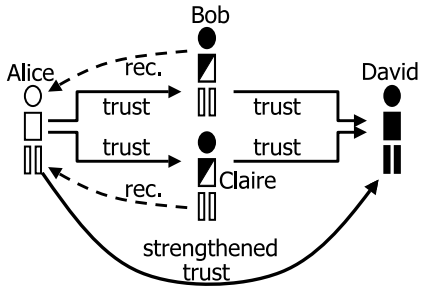

Fig. 3. Parallel trust combination

In the case where Alice receives conflicting recommended trust, e.g. both trust and distrust, then she needs some method for combining these conflicting recommendations in order to derive her trust in David.

We will use the symbol "," to denote combination of trust, so that Alice's combination of the two parallel trust chains from her to David can be expressed as:

$$\text{Alice} \overset{\rightarrow}{:} \text{David} = (\text{Alice} : \text{Bob} : \text{David}), (\text{Alice} : \text{Claire} : \text{David}) \tag{4}$$

In the above expression the trust purpose is implicit, and the following expression makes it explicit with regard to the trust purpose:

$$\begin{aligned} \text{Alice}; [dP_1] \overset{\rightarrow}{:} \text{David} = {} & (\text{Alice}; [iP_1] : \text{Bob}; [dP_1] : \text{David}), \\ & (\text{Alice}; [iP_1] : \text{Claire}; [dP_1] : \text{David}) \end{aligned} \tag{5}$$

5 Topology Analysis

Trust topologies can involve many principals, and capital letters $A, B, \ldots$ will be used to denote principals instead of names such as Alice and Bob.

We will first explain why a recommendation should always be passed in its original form from the recommender to the relying party, and not as secondary derived trust. Fig. 4 shows an example of how not to provide recommendations.

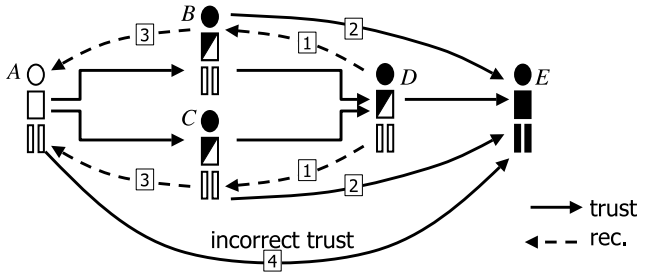

Fig. 4. Incorrect recommendation

In Fig. 4 the trust and recommendation arrows are indexed according to the order in which they are formed whereas the initial trust relationships have no index. In the scenario of Fig.4 D passes his recommendation about E to B and C (index 1) so that B and C are able to derive direct trust in E (index 2). Now B and C pass their derived trust in E to A (index 3) so that she can derive direct trust in E (index 4). As a result A perceives the topology to be $(A : B : E), (A : C : E)$. The problem with this scenario is that A is ignorant about D so that A in fact derives a hidden topology that is different from the perceived topology:

$$\begin{array}{ccc} \text{Perceived topology:} & & \text{Hidden topology:} \\ (A : B : E), (A : C : E) & \neq & (A : B : D : E), (A : C : D : E) \end{array} \tag{6}$$

The reason for this is that B's trust $B \overset{\rightarrow}{:} E$ was derived from $B : D : E$ and C's trust $C \overset{\rightarrow}{:} E$ was derived from $C : D : E$, so when B and C recommend E they implicitly recommend $B : D : E$ and $C : D : E$ [10] but this is hidden from A. It can easily be seen that neither the perceived nor the hidden topology is equal to the real topology, which shows that this way of passing recommendations produces incorrect results.

We argue that B and C should pass the recommendations explicitly as $B : D : E$ and $C : D : E$ respectively, and this is certainly possible, but then A needs to be convinced that B and C have not altered the recommendations from D. If B and C are unreliable they might for example try to change the recommended trust measures. Not only that, any party that is able to intercept the recommendations from B, C, or D to A might want to alter the trust values, and A needs to receive evidence of the authenticity and integrity of the recommendations. Cryptographic security mechanisms can typically be used to solve this problem, and this will be discussed in more detail in Sec.6.

It is thus necessary that A receives all the trust recommendations unaltered and as expressed by the original recommending party. An example of a correct way of passing recommendations is indicated in Fig. 5

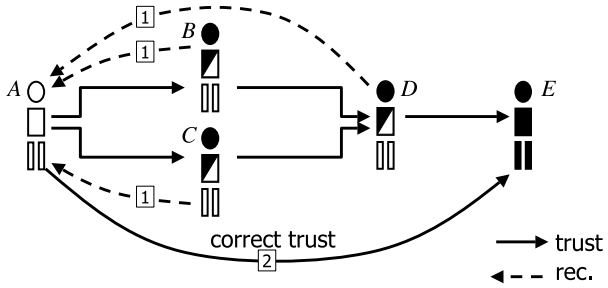

Fig. 5. Correct recommendation

In the scenario of Fig. 5 the perceived topology is equal to the real topology which can be expressed as:

$$A \overset{\rightarrow}{:} E = ((A : B : D), (A : C : D)) : E \tag{7}$$

The lesson to be learned from the scenarios in Fig. 4 and Fig. 5 is that there is a crucial difference between recommending trust in a principal resulting from your own experience with that principal and recommending trust in a principal which has been derived as a result of recommendations from others. We will use the term *primary trust* to denote the former, and *secondary trust* for the latter. Fig. 4 illustrated how problems can occur when secondary trust is recommended, so the rule is to only recommend primary trust [10]. Derived trust is per definition always secondary trust so that for example, A's derived trust in E in Fig. 5 should not be recommended to others.

Expressing transitive chains in the form of Eq. (3) is not always practical, for example when the topology is large, or when only parts of it are known. Instead, each isolated trust relationship can be expressed individually, and an automated parser can establish valid topologies depending on the need.

The initial trust relationships of Fig. 5 can for example be listed as in Table 1 below:

Table 1. Initial trust relationships of Fig.5

Origin	Target	Purpose	Variant	Measure	Time
A	B	P_1	indirect	μ_1	$\tau_3 = 08.09.2003$
A	C	P_1	indirect	μ_2	$\tau_3 = 08.09.2003$
B	D	P_1	indirect	μ_3	$\tau_1 = 03.09.2003$
C	D	P_1	indirect	μ_4	$\tau_1 = 03.09.2003$
D	E	P_1	direct	μ_5	$\tau_1 = 03.09.2003$
D	E	P_1	direct	μ_6	$\tau_2 = 05.09.2003$

A parser going through Table 1 will be able to determine the topology of Fig. 5. The principal A can be called a relying party because she relies on the recommendations from B, C and D to derive her trust in E. We will assume that relying parties will always try to base derived trust on the most recent recommendations. In Table 1 it can be observed that there are two entries for the trust leg $D : E$ and based on the principle of the most recent trust, the parser would select the last entry expressed by $D;[dP_1, \mu_6, \tau_2] : E$. If the relying party A derives her trust in E at time τ_3, then that trust can be expressed as:

$$A;[dP_1, \mu_7, \tau_3] \overrightarrow{:} E = ((A;[iP_1, \mu_1, \tau_3] : B;[iP_1, \mu_3, \tau_1] : D), \\ (A;[iP_1, \mu_2, \tau_3] : C;[iP_1, \mu_4, \tau_1] : D));[dP_1, \mu_6, \tau_2] : E \quad (8)$$

The piece of pseudo-code below represents a parsing algorithm that finds a trust path for a specific origin, target and trust purpose, if it exists, in a set of recommendations. It evaluates all possible trust paths as true or false, and uses binary logic OR to combine parallel trust paths. This simplification assumes that trust measures are binary. As already mentioned, trust can be measured as discrete or continuous values, in which case a more complex algorithm would be needed.

This simple algorithm can be useful to determine if at least one potential trust path exists between two principals, and further analysis can then be done to derive the measure of trust resulting from the topology. The latter must be based on algebraic operators for computing transitive and parallel trust. This issue will be briefly discussed in Sec 7.

```
Pseudo-Constructor for a Recommendation:
========================================

Recommendation(Principal origin, Principal target, Purpose purpose,
               Variant variant) {
  this.origin = origin;
  this.target = target;
  this.purpose = purpose;
  this.variant = variant;
}
```

```
Pseudo-code for a simple evaluation algorithm:
==============================================

Output is binary:
  true  --> there is a trust path
  false --> there is none

Definition of functions:
  transitivity       : --> logical AND
  trust combination  , --> logical OR

boolean ParseTrust(Principal origin, Principal target, Purpose purpose,
                   RecommendationSet recs) {
  IF ((origin, target, purpose, 'direct') IN recs) {
    RETURN true;
  }
  ELSE {
    SELECT rec FROM recs WHERE ((rec.origin == origin) AND
                               (rec.purpose == purpose) AND
                               (rec.variant == 'indirect'));
    IF (RESULTS_FROM_SELECT == empty) {
      RETURN false;
    }
    ELSE {
      Boolean b = false;
      FOREACH rec IN RESULTS_FROM_SELECT DO {
        b = b OR ParseTrust(rec.target, target, purpose, recs\rec);
      }
      RETURN b;
    }
  }
}
```

6 Integrity and Authenticity of Recommendations

Cryptography can be used to provide authenticity and integrity of recommendations. This in turn requires that every participant holds a trusted (i.e. authentic) key. The process of generating, distributing and using cryptographic keys is called key management, and this still is a major and largely unsolved problem on the Internet today.

Public-key infrastructures (PKI) simplify key management and distribution but create trust management problems. A PKI refers to an infrastructure for distributing public keys where the authenticity of public keys is certified by Certification Authorities (CA). A certificate basically consists of the CA's digital signature on the public key together with the owner identity, thereby linking the key and the owner identity together in an unambiguous way. In order to verify a certificate, the CA's public key is needed, thereby creating an identical authentication problem. The CA's public key can be certified by another CA etc., but in the end you need to receive the public key of some CA out-of-band in a secure way. Although out-of-band channels can be expensive to set up and operate they are absolutely essential in order to obtain a complete chain of trust from the relying party to the target public key.

However, there are potential trust problems in this design. What happens if a CA issues a certificate but does not properly check the identity of the owner, or worse, what happens if a CA deliberately issues a certificate to someone with a false owner identity? Furthermore, what happens if a private key with a corresponding public-key certificate is leaked to the public domain by accident, or worse, by intent? Such events could lead

to systems and users making totally wrong assumptions about identities in computer networks. Clearly CAs must be trusted to be honest and to do their job properly and users must be trusted to protect their private keys.

The concept of trusted platforms introduces additional security features to reputation systems in general, and uses cryptographic means to secure recommendations and trust assessments in particular [11]. When including security in the description of our scheme, it must be assumed that every principal has a public/private key pair that can be used for authentication and encryption. We can either assume that the public keys are absolutely trusted (i.e. that the relying party is absolutely certain about their authenticity) or that they too can have various levels of trustworthiness. The easiest is of course to assume absolute trust, because then the authenticity and integrity of the recommendations communicated can be assumed, and trust topologies can be analysed as described in the previous sections.

If on the other hand trust in cryptographic keys can have varying measures, then the trust in every cryptographic key must be determined before the topology in question can be analysed. Trust in public keys can be derived from trust in the various components of a PKI. A method for analysing trust in the authenticity of public keys in a PKI is described in detail in [10] and it broadly follows the same principles as described in the previous sections.

The consequence of having to derive trust in public keys is that the relying party might have to analyse a separate topology for every principal in the topology of interest. The analysis of the topology of Fig.5 which includes 3 recommendations would for example require the derivation of the trust in the public keys of B, D and C before the topology itself can be analysed and the trust in the target entity E can be derived. The analysis of the topology would then have to take the authenticity of the public keys into account in addition to the trust in the principals. With reference to the scenario of Fig.5 the trust relationships that have to be taken into account are illustrated in Fig.6 below.

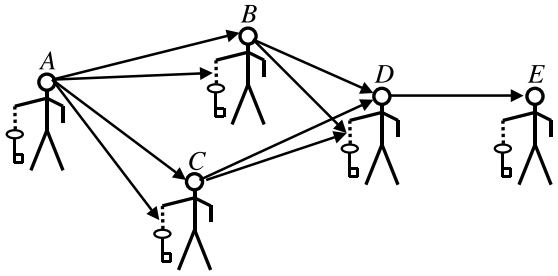

Fig. 6. Trust relationships in a topology with authenticated public keys

The existence of two separate trust legs with different purposes, where the first is targeted at the principal himself and the second at the binding between the public key and its owner requires some method for combining the two together. Various methods can be imagined for this purpose and one possibility is to use conjunction (i.e. logical

AND in the binary case) of the two trust purposes[10]. The purpose of the trust targeted at the public key-to-owner binding can typically be described as P_2; " *trusts the public key to be authentic* ", and it can be associated with a measure and timestamp as in the normal case. We will use the symbol "$\wedge$" to denote conjunction of two trust relationships. A's derived trust in E can then be expressed as:

$$A;[dP_1,\mu_7,\tau_3] \overset{\rightarrow}{:} E \;=$$

$$\begin{array}{l}(\;\; (A;[iP_1,\mu_1,\tau_3] \wedge [dP_2,\mu_8,\tau_3] : B;[iP_1,\mu_3,\tau_1] \wedge [dP_2,\mu_9,\tau_1] : D), \\ \quad\; (A;[iP_1,\mu_2,\tau_3] \wedge [dP_2,\mu_{10},\tau_3] : C;[iP_1,\mu_4,\tau_1] \wedge [dP_2,\mu_{11},\tau_1] : D)\;); \\ [dP_1,\mu_6,\tau_2] : E\end{array} \tag{9}$$

For a parser to be able to derive this topology, it is of course required that the relying party A has received and stored all these trust recommendations for example in the form of a table similar to Table 1. Only the first trust purpose in a conjunctive trust relationship is used by the parser to determine the aactual topology. The second trust purpose is only used when computing the derived trust measure.

To illustrate the role of key authenticity, take for example the case when a principal is recommended to be reliable but that the binding between the principal and his key is broken, e.g. because it is known that the private key has been stolen by an intruder. The result of the conjunction between trust in the principal and the distrust in his key would produce distrust, indicating that a principal identified by this particular public key can not be trusted. This is what intuition would dictate because it is now possible that recommendations that appear to come from the principal in fact originate from the intruder who stole the private key and who is not trusted.

7 Measuring and Computing Trust

In previous sections we have used the term "trust measure" without specifying how it should be expressed or computed, and that is not the topic of this paper. Instead, we have indicated several intuitive principles that trust measures and computational rules should follow. Without going into great detail this section describes additional requirements that trust measures and operators should satisfy.

While trust has no specific measurable units, its value can be measured in a similar manner to other abstract commodities, like information or knowledge [12]. Many trust measures have been proposed in the literature varying from discrete measures [13–17] to continuous measures [10, 18–23].

Typical discrete trust measures are for example "strong trust", "weak trust", strong distrust" and "weak distrust". PGP[13] is a well known software tool for cryptographic key management and email security that for example uses the discrete trust measures "ultimate", "always trusted", "usually trusted", "usually not trusted" and "undefined" for key owner trust. In order to obtain compatibility between discrete and continuous methods it should be possible to interpret such discrete verbal statements by mapping them to continuous measures.

When measuring trust, it is critical that the trust value is *meaningful* to and *usable* by both the origin and the target transacting partners. Otherwise, if trust is subjectively

measured by each party using different methods, the value becomes meaningless and unusable. By explicitly defining P_1 and P_2 in the scenarios above, we ensure that the interacting parties have a common understanding of the trust purpose so that they are deriving meaningful trust values for one another.

The *context*, or purpose, of the interaction must also be satisfied by the trust measure. Again, by explicitly defining P_1, and P_2, the context becomes clear to all parties participating in the interaction.

As mentioned in Sec. 2, *time* is another element that should be captured together with trust measures. This element is necessary not only to demonstrate how trust is evolving, but also in order to enable transaction partners to assess trust based on, for example, the most recent trust value available.

Determining the *confidence* of the trust measure is also a requirement. For example, the weakening of trust through long transitive chains should result in a reduced confidence level. On the other hand, a large number of parallel recommendations should result in an increased confidence level.

Finally, in order to derive trust measures from a topology there must be explicit methods for *combining trust measures in a transitive chain* as in Fig.2, for *combining trust measures in parallel chains* as in Fig.3 as well as for *combining trust measures in a conjunction of trust relationships* as in Fig.6. Various methods and principles for deriving trust from such combinations have been proposed in the literature [10, 13, 14, 16, 18–20]. The validation and suitability assessment of any computational approach should be based on simulations and usability studies in environments equal or similar to those where it is intended for deployment.

8 Conclusion

We have captured the diversity that exists in trust by specifying three basic topology dimensions, that of trust origin, trust target and trust purpose. Additionally, we have incorporated the dimensions of measure and time into the specification which are important for deriving trust measures through computational methods.

We have described principles for recommendation such that transitive trust chains might be formed which capture the basic trust diversity dimensions. In this regard, we found that a trust topology is valid when every leg in the chain contains the same trust purpose with the last leg having direct trust and all previous legs having indirect trust.

We provided a notation with which to express these trust principles and to analyse topologies of transitive trust. In doing so, we proved the rule that only primary trust should be recommended, as recommending secondary trust results in incorrect trust derivation.

We showed also that the parsing of transitive trust chains may be automated such that trust measures might be derived practically and easily in scenarios where the topology is large or where only parts of the topology are known. We presented the pseudocode for such a parser.

Finally, we presented a method with which to ensure the integrity and authenticity of recommendations in transitive trust chains, as well as several requirements for expressing and computing trust measures.

References

1. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proc. of the 17th IEEE Symposium on Security and Privacy, Oakland (1996) 164–173
2. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The KeyNote Trust-Management System. RFC 2704, Network Working Group, IETF (1999)
3. Ellison, C., Frantz, B., Lampson, B., Rivest, R.L., Thomas, B., Ylonen, T.: SPKI Certificate Theory. RFC 2693, Network Working Group, IETF (1999)
4. Rivest, R., Lampson, B.: SDSI - A Simple Distributed Security Infrastructure (1996)
5. McKnight, D., Chervany, N.: The Meanings of Trust. Technical Report MISRC 96-04, Management Informations Systems Research Center, University of Minnesota, (1996)
6. Deutsch, M.: Cooperation and Trust: Some Theoretical Notes. In Jones, M., ed.: Nebraska Symposium on Motivation, Nebraska University Press (1962)
7. Golembiewski, R., McConkie, M.: 7. In: The Centrality of Interpersonal Trust in Group Processes. Wiley (1975) 131–185
8. Jøsang, A.: The right type of trust for distributed systems. In Meadows, C., ed.: Proc. of the 1996 New Security Paradigms Workshop, ACM (1996)
9. Christianson, B., Harbison, W.: Why Isn't Trust Transitive? In: Proc. of the Security Protocols Workshop. (1996) 171–176
10. Jøsang, A.: An Algebra for Assessing Trust in Certification Chains. In Kochmar, J., ed.: Proc. of the Network and Distributed Systems Security Symposium (NDSS'99), The Internet Society (1999)
11. Kinateder, M., Pearson, S.: A Privacy-Enhanced Peer-to-Peer Reputation System. In: Proc. of the 4th International Conference on Electronic Commerce and Web Technologies (EC-Web 2003), Prague, Czech Republic, Springer-Verlag (2003)
12. Dasgupta, P.: Trust as a Commodity (2000)
13. Zimmermann, P.: The Official PGP User's Guide. MIT Press (1995)
14. Abdul-Rahman, A., Hailes, S.: A Distributed Trust Model. In: Proceedings of the 1997 New Security Paradigms Workshop, ACM (1997) 48–60
15. Cahill, V., Shand, B., Gray, E., et al.: Using trust for secure collaboration in uncertain environments. To appear in IEEE Pervasive Computing (2003)
16. Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. Technical Report RS-03-4, BRICS (2003)
17. Manchala, D.: Trust Metrics, Models and Protocols for Electronic Commerce Transactions. In: Proceedings of the 18th International Conference on Distributed Computing Systems. (1998)
18. Jøsang, A.: A Logic for Uncertain Probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9** (2001) 279–311
19. Kohlas, R., Maurer, U.: Confidence valuation in a public-key infrastructure based on uncertain evidence. In: Proceedings of the International Workshop on Theory and Practice of Public-Key Cryptography, Springer (2000)
20. Kinateder, M., Rothermel, K.: Architecture and Algorithms for a Distributed Reputation System. In Nixon, P., Terzis, S., eds.: Proc. of the First International Conference on Trust Management. Number 2692 in LNCS, Crete, Greece, iTrust, Springer-Verlag (2003) 1–16
21. Gray, E., O'Connell, P., Jensen, C., Weber, S., Seigneur, J.M., Yong, C.: Towards a Framework for Assessing Trust-Based Admission Control in Collaborative Ad Hoc Applications. Technical Report 66, Dept. of Computer Science, Trinity College Dublin, (2002)
22. Beth, T., Borcherding, M., Klein, B.: Valuation of Trust in Open Networks. In Gollmann, D., ed.: ESORICS 94, Brighton, UK (1994)
23. Marsh, S.: Formalising Trust as a Computational Concept. Phd thesis, University of Stirling, Department of Computer Science and Mathematics (1994)

Reasoning about Naming and Time for Credential-based Systems

Nathalie Chetcuti[1] and Fabio Massacci[2]

[1] Dep. de Informatique, Univ. Artois - France, chetcuti@cril.univ-artois.fr
[2] Dip. di Informatica e Telecomunicazioni, Univ. di Trento -Italy,
Fabio.Massacci@unitn.it

Abstract. Reasoning about trust management and credential-based systems such as SDSI/SPKI, is one of today's security challenges. The representation and reasoning problem for this (simple) public key infrastructure is challenging: we need to represent permissions, complex naming constructions ("Martinelli's officemate is FAST's PC-Chair's Colleague"), intervals of time and metric time for expiration dates and validity intervals.
Such problem is only partly solved by current approaches. At first because they focus on Lamport and Rivest's SDSI and SPKI, the major goal being to show that the proposed logics and semantics captured exactly SPKI behavior or were better in this or that respect. Second, reasoning about time is missing. Complicated logics and algorithms are put in place for name resolution but it is always assumed that just the valid credentials are evaluated.
What we find missing is what Syverson termed an "independently motivated semantics". Here, we propose such a semantics with annexed logical calculi. The semantics has a natural intuitive interpretation and in particular can represent timing constraints, intersection of validity intervals and naming at the same time.
We also provide a logical calculus based on semantic tableaux with the appealing feature that the verification of credentials allows for the direct construction of a counter-model in the semantics when invalid requests are made. This combines semantic tableau method for modal and description logics with systems for reasoning about interval algebra with both qualitative and metric constraints.

1 Introduction

The security of credential-based systems is one of today's security challenges. This is mostly due to the disappearance of the traditional model of client/server interaction and its replacement by Service Oriented computing [11]: the important data is on some server which knows the clients and let them just have what they deserve. First of all, clients are no longer known by servers: the entire idea behind web services is that requests may come from everybody, provided they have the right credentials . Second, servers themselves are often distributed and their security policies may come from different sources and different administrative domains.

The traditional authentication and authorization questions have been transformed into another one about *trust management*: "Does the set of credentials about identifiers and about permissions proves that the request complies with the set of local policies?"

To perform these tasks without a centralized security infrastructure, a number of proposals have been put forward by security researchers (see the recent survey by Weeks [23]). One of the most cited work is Lampson and Rivest's SDSI (Simple Distributed Security Infrastructure), later refined into an Internet RFC as SPKI (Simple Public Key Infrastructure) [10]. Many later proposals such as Binder [9] build upon the intuition of those works.

Loosely speaking, the appeal of SDSI/SPKI is to have distilled the concept of *local name* and to have reduced the traditional access and authorization decision into a problem of verifying the combination of *credentials linking local names and global names* and *credentials linking names and permissions*. For example, in FAST's Chair, the Chair is a local name, which maps to an individual who is likely to be different from CSFW's Chair. The individual standing for Chair may be linked by some certificate to an individual named Dimitrakos. Dimitrakos' Colleague may also be mapped into more than one individual. Suppose now that any Dimitrakos' Colleague was granted access to Martinelli's Computer, should a claim from an FAST's Chair's Colleague be granted by the server? The reasoning is further complicated by time: one can be PC-chair or colleague for an interval of time and then something may change. Should the claim be granted in 2004?

The SDSI/SPKI proposal has been the subject of an intense debate and a number of researchers have formalized this proposal or its alternatives using logics to analyze and emphasize differences or subtle features. Abadi [1] has used a modal logic, which later Howell and Kotz [14] have modified, Halpern and Van der Meyden [12, 13] have proposed another modal logic to reason about it, Jha, Reps and Stubblebine have used model checking for the verification for the time-free fragment of the logic [21]. Li et al. [16, 19] have used Datalog.

So far, the approaches in Computer Security fora have focussed on using logics for giving "the" semantics to the operational description of SDSI/SPKI [10]. This has a number of drawbacks. For sake of example, to match the certificate treatment of SPKI, Halper and van der Meyden semantics [12, Sec. 3.2] is self-referential: a certificate is valid in a model because it is in the list of valid certificates. The English wording looks almost unacceptable though this is the exact description of the logical definition in the paper. This mixes syntax (the presence of a string in a set) and semantics (validity of the attribution of a key to a principal). The list of "valid" certificates describes what is in a set. In the world, the certificate may no longer describe the situation (for instance the key has been cracked and CRLs may not have been issued yet): syntactic would-be-valid certificates could well not be semantically valid.

A second limitation is the (missing) ability of reasoning about time. Obviously, for reasoning about validity interval one could simply apply the SPKI algorithm. But the same reasoning applies to any 40 pages paper describing a logic for the 4 pages name resolution algorithm. Time is the most intriguing aspect of certificates, in particular if we have name attribution certificates with validity periods that differs from validity periods of privileges attribution certificates. So, if Fabio's co-author is Nathalie since 19/Jan/02, and Fabio's co-author can write chet-mass-03.tex up to 12/Jul/03 this is equivalent to say that the model could (semantically) satisfy also a certificate stating that Nathalie can write the paper from 19/Jan/02 to 12/Jul/03. The certificate may

well not be in the list of valid certificates, as nobody might have actually issued it, but is nevertheless "semantically valid" as it would describe the status of the world. Even more sophisticated models such as Jha, Reps, and Stubblebine general authorization problems [21] cannot reason about such phenomena as their certificates are timeless.

1.1 Our Contribution

Building upon previous formalizations by Abadi, Halpern and van der Meyden, Jha and Reps we provide a general model for reasoning about naming and identifiers, authorization, credentials, *and* time. We hope that this would provide the equivalent of what Syverson termed an "independently motivated semantics" [22]. Last but not least, we provide a method to reason about them. Nobody is interested in formalizing the SDSI/SPKI FAST's PC-Chair's ConfMan policies if we cannot then decide whether the remote user making a request signed with the key 0xF34567 is allowed to see the reviews of paper ES0345.pdf. For the naming and modal part we need to combine features for advanced work in modal and dynamic logics [17, 7]. For the temporal part, as we shall see, this is a challenge where a CSP-based qualitative reasoning proper of Allen's Interval algebra [4] is not sufficient. TCPs (Temporal CSPs) and STPs (Simple Temporal Problem) are necessary to handle metric temporal relations [8].

In the rest of the paper we sketch the intuitions about SDSI/SPKI, we show the semantical model for credential based systems, and the intuitions behind it. We give a sound and complete calculus and conclude the paper with a brief discussion.

2 A Primer on SDSI/SPKI

The idea behind SDSI/SPKI [10] is that servers make access control decision by looking at public key credentials which either link identifiers to known roles or other identifiers or link identifiers with privileges.

Each principal has its set of *local names*[1], denoted by n, possibly with subscript. Name can be composed so that for Lamport, the local name Ron may map into Rivest and the name Ron′Buddy map into what Rivest consider a buddy. In SPKI, *Compound names* are denoted by the tuple construct `(name n1 n2 ...nk)` or by the equivalent expression $n'n_1'\ldots n_k$ where each $n-i$ is a local name, and n is either a local name or a public key. When n is a key we have a *fully qualified name*. The interpretation of a compound name depends on the agent except for fully-qualified names. So that the interpretation of Ron′poker-buddies by one agent depends on its interpretation of Ron, and may be different from another agent's interpretation of Ron and Ron′poker-buddies.

Strictly speaking, we have no agents interpeting names in SPKI but just keys. We may say that Ron's interpretation by the agent Lamport is mapped into the agent Rivest, but what we can only say in practice is that Ron is mapped into the public key k_r. So that the public key k_l (which corresponds to what we call the agent Lamport) takes any credential verifiable with the public key k_r as a statement coming from his fellow Ron.

[1] In absence of a centralized naming authority there are no global names in SPKI whereas SDSI had global names such as DNS.

SPKI has other kinds of principals such as hashes of keys and threshold subject ("any m out of N of the following subjects") for joint signatures, or the reserved word "Self", representing the entity doing the verification. Here, along the same line of Jha, Reps and Stubblebine [21], we only consider compound names.

Credentials are represented by certificates. There are various types of certificates in SPKI: naming certificates, authorization certificates, and certificate revocation lists (CRLs). Here, we only treat the first two but the framework is designed to give a reasonable account of revocation list.

A *naming certificate* has the form of a cryptographically signed message with contents `(cert (issuer (name k n)) (subject p) valid)`, where `k` is a key (representing the issuer, whose signature is on the certificate), `n` is a local name, `p` is a fully-qualified name, and `valid` is an optional section describing temporal validity constraints on the certificate. The valid section describes an interval during which the certificate is valid, expressed by means of a `(not-before date)` and/or a `(not-after date)`. It may also include additional tests for the validation of certificates.

If `k` is a key, `p` is a fully-qualified name, `A` is an authorization (loosely speaking a set of actions), and `valid` is a temporal validity section then an *authorization certificate* is `(cert (issuer k) (subject p) (propagate) A valid)`. Intuitively, the issuer uses such a certificate to grant `p` the authority to perform the actions in `A`. Moreover, if the optional propagation field is present, then the subject is further authorized to delegate this authority to others[2].

The logical syntax is based on the proposals by Abadi [1], Halpern and van der Meyden [12, 13], Jha, Reps et al. [20, 21]. We have a set of actions A, a set of keys K, and a set of names N. A *principal* can be a key $k \in K$, a local name n where $n \in N$, or $p\,'\,q$ where p and q are principals.

The *atomic formulae* of our logic are $p \mapsto q$ and p perm a, where p and q are principals and a is an action. More complex formulae are built by the usual operators of negation, conjunction and disjunction. The intuition behind $p \mapsto q$, which is read "p speaks for q", is that for the current principal any authorization for p can be mapped into an authorization for q.

We have two forms of *certificates*, $\mathsf{cert}_k\,(p \mapsto q\,[t_b, t_e])$ to associate names to other names and $\mathsf{cert}_k\,(p$ perm $a\,[t_b, t_e])$ to associates permissions to names. Here, we allow one to use compound names as subjects and not just local names. The interpretation of $\mathsf{cert}_k\,(n \mapsto p\,[t_b, t_e])$ is that for the principal k the local name n is bound to the fully qualified name p for the validity period between the instant t_b and t_e. The interpretation of $\mathsf{cert}_k\,(p$ perm $a\,[t_b, t_e])$ is that k permits action a to p for the validity period of the certificate. It is possible to have also open ended certificates. Delegation can be treated by generalizing permission certificates replacing action a with recursive permissions.

With respect to the calculus in [1, 2] we have eliminated the $says$ operator because it is subsumed by the certificate operator. See [17] for an automated reasoning method for some fragments of Abadi's *et al.* calculus.

[2] This unbounded delegation of powers has been mitigated by other authors. For instance, Li et al. [16, 19] introduce the notion of delegation up to n steps.

3 Semantics

The motto of any credential-based security religion could be "Extra public-keys nulla salus" and we build upon this intuition by making a model where the basic domain is a set of keys, and where names connect keys with each other and with permissions.

Let's first give a model *without* time. So a *model* is a generalized Kripke structure that is a triple $\langle \mathbf{K}, \mathbf{LN}, \mathbf{A} \rangle$ where $\mathbf{K}$ is a set of real keys (such that for all $k \in K$ there is a $\mathbf{k} \in \mathbf{K}$ plus possibly additional keys). The naming relation $\mathbf{LN}$ is an indexed family of partial mappings from keys to subset of keys $N \longrightarrow \mathbf{K} \rightarrow 2^{\mathbf{K}}$. The grant relation $\mathbf{A}$ is a function mapping actions into mappings of keys into subset of keys $\mathbf{A} \longrightarrow \mathbf{K} \rightarrow 2^{\mathbf{K}}$. In the sequel we always use X as the syntactic value and $\mathbf{X}$ as the semantic counterpart.

The intuition behind the set $\mathbf{K}$ should be obvious. The naming relation associates to each local name a mapping: who uses this name for whom. So $\mathbf{k}' \in \mathbf{LN}_n\mathbf{k}$ means that the principal associated to key $\mathbf{k}$ associates to the name n at least the key $\mathbf{k}'$. We have a set because the same name n may refer to many individuals as in $Ron\,'\,poker - buddies$. Furthermore note that the mapping may be partial because a principal may not associate anybody to a name. In the sequel, for simplicity we use the relation-oriented notation to describe the mapping: $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{LN}_n$.

The grant relation associates to each key the set of actions that the agent holding the key is willing to permit to other principals. So $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{A}(a)$ means that the principal associated to the key $\mathbf{k}$ is willing to permit action a to $\mathbf{k}'$.

Names can be lifted to compound names by giving a semantics for the $'$ operator:

- $\mathbf{N}(n) = \mathbf{LN}(n)$ where $n \in N$
- $\mathbf{N}(k) = \bigcup_{\mathbf{k}' \in \mathbf{K}} \{\langle \mathbf{k}', \mathbf{k} \rangle\}$ where $k \in K$
- $\mathbf{N}(p\,'\,q) = \bigcup_{\mathbf{k_p} \in \mathbf{K}} \{\langle \mathbf{k}, \mathbf{k_{pq}} \rangle \mid \langle \mathbf{k}, \mathbf{k_p} \rangle \in \mathbf{N}_p \text{ and } \langle \mathbf{k_p}, \mathbf{k_{pq}} \rangle \in \mathbf{N}_q\}$

The second rule says that syntactic keys are mapped into the corresponding semantics keys. The last rule is just composition: if $\mathbf{k}$ associate $\mathbf{k_p}$ to the name p and $\mathbf{k_p}$ associates $\mathbf{k_{pq}}$ to the name q then $\mathbf{k}$ should associate $\mathbf{k_{pq}}$ to the name $p\,'\,q$.

Proposition 1. *A principal $p_0\,'\,\ldots\,'\,p_l$ is equivalent to some principal $q_0\,'\,\ldots\,'\,q_m$ where q_0 is either a key or a name and $q_1, \ldots, q_m$ are names.*

So from now on we shall consider only principals in the latter form.

Figure 1 shows an example of a Kripke structure for credential systems without time. It describes the relation between two co-authors and a PC chair. Keys are represented as nodes of the graph and the permissions to read or write the paper labels each node. The key k_1 corresponds to Fabio Massacci, whereas the key k_3 is Fabio Martinelli which is $k_3\,'\,\mathsf{FAST\text{-}03}\,'\,\mathsf{co\text{-}chair}$. Note how the name Fabio is mapped into different keys.

To lift our structure to time we introduce the concept of a *trace* $\mathcal{M}$ that is a mapping from time to models. A trace associates to each instant of time a given model $\mathcal{M}@t = \langle \mathbf{K}@t, \mathbf{LN}@t, \mathbf{A}@t \rangle$ where $t \in \mathbb{R}$. Then $\mathbf{K}@t : \mathbb{R} \longrightarrow \mathbf{K}$ is the set of real keys which are associated to the named keys at time t, $\mathbf{LN}@t$ is the local naming relation with the additional time parameter $\mathbb{R} \times N \longrightarrow \mathbf{K} \rightarrow 2^{\mathbf{K}}$ at time t and $\mathbf{A}@t$ is the grant relation: $\mathbb{R} \times A \longrightarrow \mathbf{K} \rightarrow 2^{\mathbf{K}}$.

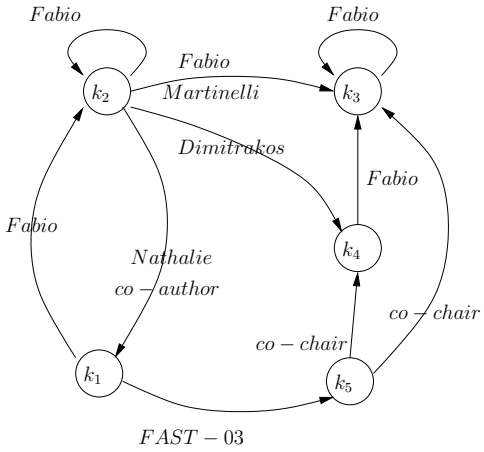

Fig. 1. Timeless Model

The extension of the semantics to compound names *principals* is identical, except for the t subscript. In the sequel, we merge the principal and the time in the same subscript. So we write $\mathbf{N}_{p@t}$ instead of $\mathbf{N}@t(p)$, and similarly for other operators.

Intuitively, we can view the same formal structure from two different perspectives. At first glance, we have a timeline and at each time the entire world may change. This is hardly appealing: we are used that the world stays the same and something in it changes. The second perspective is more appealing: the model is composed by parts (keys, naming relations, and permission relations) that change over time.

In Figure 3 we show a timed version of the model shown in Figure 1 in which we have added the validity interval of each naming relation.

The timed model allows for fine grained distinctions that are not possible in the untimed model. If we consider key security properties like dynamic separation of duties, the timed models allows for a natural description of such constraints and situations.

For instance in the untimed model k_3 is always $k_1\,'\,\mathsf{FAST}\,'\,\mathsf{Co\text{-}chair}$, and it is at the same time $k_2\,'\,\mathsf{Fabio}$ and $k_2\,'\,\mathsf{Martinelli}$. In the timed modelthe concatenation of the validity periods is such that $k_1\,'\,\mathsf{FAST}\,'\,\mathsf{Co\text{-}chair}$ only maps into k_3 for the period $[7/03, 9/03]$ which is by far shorter then each validity period of the component naming relations. Equally, k_3 has never been at the same time $k_2\,'\,\mathsf{Martinelli}$ and $k_2\,'\,\mathsf{Fabio}$ (as in the untimed model), but either only one or the other. Thus k_3 has never got the permissions associated with both identifiers.

It is possible to add more security constraints on the model. For instance, should we have persistence of global keys, i.e. should $\mathbf{K}@t = \mathbf{K}@t'$ hold? Basically this says that all keys have been already invented, just not used. Another possibility is that the only possible keys are those listed in a given set of syntactic certificates, and thus $\mathbf{K}@t = K$ for all t.

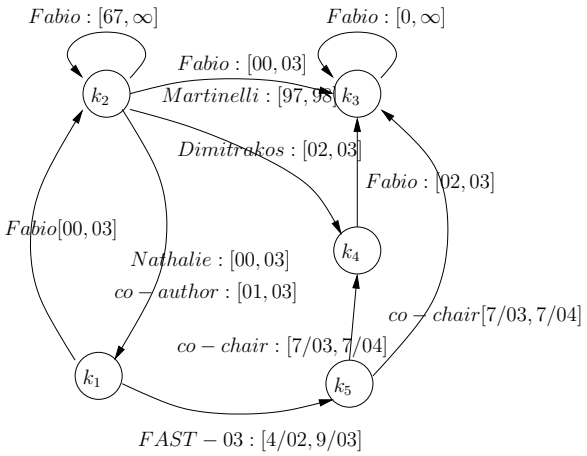

Fig. 2. Timed Model

Another question is whether the validity period of keys should always be a connected interval. Suppose that we have $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{N}_{\mathsf{FAST'Co\text{-}chair}@12/07/03}$, that is the real key $\mathbf{k}$ associates the key $\mathbf{k}'$ to the FAST′Co-chair. After an year the certificate expires and we have $\langle \mathbf{k}, \mathbf{k}' \rangle \notin \mathbf{N}_{\mathsf{FAST'Co\text{-}chair}@12/07/04}$. Do we want to impose that for all $t \geq 11/07/2004$ we have $\langle \mathbf{k}, \mathbf{k}' \rangle \notin \mathbf{N}_{@tFAST'\mathsf{Co\text{-}chair}}$? If the answer is yes, this means that after a certificate is expired we would not accept another re-validation certificate for the same key. This is one possible scenario and there might be cases when this is desirable and cases when it is not.

All such possibilities can be captured with suitable axiom schemata. Different certificate theories (for instance persistence-of-syntactic-keys, no-revalidation) can be characterized by different axiom schemata.

We have now all the necessary material to give a semantics to *formulae*.

- $\mathcal{M}@t, \mathbf{k} \models p \mapsto q$ iff for all $\mathbf{k}' \in \mathbf{K}@t$, if $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{N}_{q@t}$ then $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{N}_{p@t}$
- $\mathcal{M}@t, \mathbf{k} \models p \ \mathsf{perm}\ a$ iff $\langle \mathbf{k}, \mathbf{k}' \rangle \in \mathbf{N}_{p@t}$ implies $a \in \mathbf{A}@t(\mathbf{k}')$

As for *certificates*, we evaluate them as follows:

- $\mathcal{M}@t \models \mathsf{cert}_k\ (p \mapsto q\ [t_b, t_e])$ if $t \in [t_b, t_e]$ implies $\mathcal{M}@t, \mathbf{k} \models p \mapsto q$
- $\mathcal{M}@t \models \mathsf{cert}_k\ (p\ \mathsf{perm}\ a\ [t_b, t_e])$ if $t \in [t_b, t_e]$ implies $\mathcal{M}@t, \mathbf{k} \models p\ \mathsf{perm}\ a$

Remark 1. Comparing with Halpern and van der Meyden proposals we have no syntactic list of valid certificates. A model is an independent entity from certificates. It has its properties and satisfies some formulae. If it satisfies the appropriate formulae, then it also satisfies some certificates. In this way, as we said, a particular certificate theory characterizes a particular set of models.

Another characteristics of our model is that we worry about validity of certificates in model only if they refer to the current time. If a certificate is not at all applicable

there is little interested in knowing whether it describes the current state of the world or not.

Then we can define the notion of *satisfaction by a trace* $\mathcal{M}$, that is the notion of satisfaction for all time instant.

– $\mathcal{M} \models \mathsf{cert}_k\,(c\,[t_b, t_e])$ iff $\forall t$, $\mathcal{M}@t \models \mathsf{cert}_k\,(c\,[t_b, t_e])$

As Jha and Reps [20, 21] we have a notion of consequence for chains of certificates:

Definition 1. *A boolean combinations of certificates* χ *is a* logical consequence *of a set of certificates* $\mathcal{C}$ *if any trace which satisfies all certificates* $\mathcal{C}$ *also satisfies* χ.

If χ is a single certificates then we have exactly Jha and Reps notion of consequence. This generalized notions is more useful for deriving interesting consequences. For instance we can ask whether an invalid certificate implies that another certificate is also implicitly invalidated.

4 Semantic Tableaux

Once the model is in place, how do we know that a certificate is a logical consequence of a a set of physical certificates? Furthermore, if this is not the case, how can we get a counter-example?

We propose a calculus based on *semantic tableaux*. Intuitively, to prove that a certificate χ is a logical consequence of a set of other certificates $\mathcal{C}$ (see Def. 1) we instead try to construct a model that falsifies χ and satisfies $\mathcal{C}$. If we succeed, then we have the counter example. If we fail and we used a fair and systematic procedure, we are sure that no such countermodel exists and the formula is valid.

If we drop the timed information, our calculus could be a tableaux sibling of the model checking procedure of Jha and Reps. The additional dificulty is that for each pair of constraints on the attempted counter-model we must check that

– either there is no temporal interaction and then we simply impose that the temporal constraints are not overlapping,
– or we update the untimed information with the required constraints during the overlapping validity intervals.

The construction starts from the formulae and then try to build the entire model and the trace by incrementally constructing the naming associations, by attributing permission and by determining temporal informations.

For tableaux we shall use the usual terminology. For instance, see De Giacomo and Massacci [7] for the naming and permission part and Kautz and Ladkin [15] for the temporal part. So a *tableau* is a collection of branches, each intuitively corresponding to some potential counter-model. A *branch* has three components for *qualitative information* (such as naming relations), for *qualitative temporal information* and for *quantitative temporal information.*

For the naming and permission information we have a triple $\langle(\mathsf{K}), \mathsf{N}, (\mathsf{F}, \mathsf{A})\rangle$ where

– K are the syntactic keys plus possibly some new keys

– a function $\mathsf{N} : N \longrightarrow 2^{\mathsf{K}\times\mathsf{K}}$ are the naming relation constructed so far,
– a function $\mathsf{A} : A \longrightarrow 2^{\mathsf{K}\times\mathsf{K}}$ are the permissions assigned so far
– $\mathsf{F} : \mathsf{K} \longrightarrow 2^{Fml}$ are the formulae (labelled with validity intervals) which we try to satisfy

If no contradiction is found at the end of the construction then the countermodel would be simply $\mathbf{K} := \mathsf{K}$, $\mathbf{LN} := \mathsf{N}$ and $\mathbf{A} := \mathsf{A}$.

For the qualitative temporal information about validity of certificates we have an *interval network* $\langle TI, \mathsf{E}\rangle$ where

– TI is a set of variables representing temporal intervals
– a function $\mathsf{E} : TI \times TI \longrightarrow 2^{Allen's\ relations}$ corresponds to the qualitative temporal relations that we have forced so far

If v is an interval variable by we represent its beginning and end point as v^- and v^+

Allen's interval relations are the following: before, after, meets, $\mathsf{met-by}$, $\mathsf{overlaps}$, $\mathsf{overlapped-by}$, starts, $\mathsf{started-by}$, during, $\mathsf{contains}$, $\mathsf{finishes}$, $\mathsf{finished-by}$, equals. Their interpretation is intuitive and we refer to Allen's work for additional explanations [3, 4], and to Dechter and Meiri [8, 18] for reasoning procedures. For instance v_1 meets v_2 means that when v_1 ends, v_2 starts.

For the metric temporal information we have a point network $\langle TP, \mathsf{E}_T\rangle$ where

– TP is a set of variables representing time points
– $\mathsf{E}_{TP} : TP \times TP \longrightarrow 2^{Intervals}$ represents the metric constraints between the time points that we contructed so far.

For example $E_{TP}(t, t') = \{[1, 5], [100, 201]\}$ means that t is distant by t' either for a value on the range $1 - 5$ or the range $100 - 201$. Using linear inequalities we would have $t + 1 \leq t' \leq t + 5$ or $t + 100 \leq t' \leq t + 201$

Initialization At the very start, K is the set of keys appearing in $\{\chi\} \cup \mathcal{C}$. The sets F, N, A, TI, and TP are empty.

For each certificate $\mathsf{cert}_k\ (c\ [t_b, t_e])$ in $\mathcal{C} \cup \{\chi\}$ a new interval variable v_c is added to TI and v_c^- and v_c^+ are added to TP. Then E, resp. E_{TP}, is enriched with the constraints existing between v_c, resp. v_c^- and v_c^+ , and the remaining interval, resp. point, variables. The intuition is that the interval variable v_c is used to define the validity period of the certificate.

Finally if χ, the certificate we are trying to disprove, has the form $\mathsf{cert}_{k_i}\ (c\ [t_{ib}, t_{ie}])$, we let $\mathsf{F}(k_i) \leftarrow \{\neg c : v_i\}$ where v_i is the interval corresponding to $[t_{ib}, t_{ie}]$. Intuitively, we want a model where the certificate is not valid in the given interval.

Remark 2. At this stage one may ask why do we need at all such validity intervals and such cumbersome notation of end points: we already know that a certificate spans an interval of time by simply looking at it. This is actually correct except for one small but essential point: revocation certificates! The validity period is the maximal potential period in which a certificate *can* be valid. If certificates can be revoked, even if temporarily, we no longer knows what validity interval a certificate has just by looking at the certificate.

There is no space here to introduce revocation certificates but the machinery is necessary to scale up to a reasonable semantic account of revocation[3]. In particular the tricky bit is the validity period of revocation certificates. In a general model, there is no obligation[4] for revocation certificates to have validity period outlasting the validity period of the revoked certificate.

Suppose that cert_{k_3} ($\mathsf{Dimitrakos}'\mathsf{Fabio} \mapsto \mathsf{Fabio}\ [1/1/02, 31/12/03]$) is revoked by a certificate whose validity interval is only $[1/1/03, 1/7/03]$. It is clear what happens in the interval $[1/1/02, 31/12/02]$: the speaks-for naming relation between Fabio and $\mathsf{Dimitrakos}'\mathsf{Fabio}$ is valid. It is also clear that in the period $[1/1/03, 1/7/03]$ this certificate is n longer valid[5]. What happens during the period $[2/7/03, 31/12/03]$ in which the revocation certificate is no longer valid? The natural semantical interpretation is that the original certificate is still valid.

4.1 Rules for Tableau Construction

After the initialization step the construction proceeds step-wise by the application of a rule and the checking of the consistency of the temporal information. It stops either when all the expressions were processed, or when an inconsistency is found.

We now consider the processing of atomic formulae only. The boolean operators are handled in the usual way: if a key must satisfy the conjunction of two formulae this means that the key must satisfy both formulae and thus both formulae are added to the branch at the appropriate key.

The only tricky bit for the read not familiar with tableau methods is the treatment of disjunction. Since disjunction means that either one or another formula must be satisfied we split the current branch in two. Formally this means that we duplicate every sets that we have constructed so far K, F, N, A, TI, and TP, and add one disjunct to one instance of F and the second disjunct to the second instance of F. Then the search continues by pushing one branch on the stack and by exploring the remaining branch. If the search for counter models is unsuccessful in the first branch, the search resumes the other alternative. Of course in any tableau implementation the structure is not duplicated and pointers are used.

To simplify the rules for formulae we need some abbreviations that describe possible relations between validity intervals of certificates.

Definition 2. *Let v, v' and v^* be interval variables.*

no overlapping validity: $v \cap v' = \emptyset$ *when the following constraint is satisfied:*

$$v\{\mathsf{before}, \mathsf{meets}, \mathsf{met}-\mathsf{by}, \mathsf{after}\}v'$$

[3] Again in Halpern and Van der Meyden [13] we have a syntactic account of revocation, mostly because they have not considered time. In Jha et al, revocation is not treated.

[4] A particular certificate theory may instead impose such obligation. There might be axiom schemata forcing properties of revocation certificates. Such schemata would be mapped into conditions for inconsistency in our tableaux setting.

[5] The relation may still be valid is there is some other valid certificate.

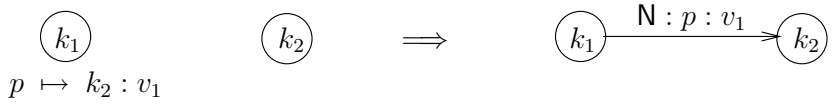

Fig. 3. Illustration of first rule for $\mapsto$ (the temporal graphs remain unchanged)

overlapping validity: $v \cap v' \neq \emptyset$ *when the following constraint is satisfied:*

$$v \left\{ \begin{array}{c} \mathsf{starts}, \mathsf{started}-\mathsf{by}, \mathsf{finishes}, \mathsf{finished}-\mathsf{by}, \\ \mathsf{overlaps}, \mathsf{overlapped}-\mathsf{by}, \mathsf{during}, \mathsf{contains}, \mathsf{equals} \end{array} \right\} v'$$

containment: $v' \subseteq v$ *when the following constraint is satisfied:*

$$v' \{\mathsf{starts}, \mathsf{during}, \mathsf{finishes}, \mathsf{equals}\} v$$

The intuition of the first rule is that two intervals have no intersection if either one interval is before the other, or one interval just finishes when the other just starts.

Proposition 2. *To compute the overlap of validity periods we let* $v^* = v \cap v'$*. Then the following conditions hold (1) if* $v\{\mathsf{starts}, \mathsf{during}, \mathsf{finishes}, \mathsf{equals}\}v'$ *then* $v^* = v$*, (2) if* $v\{\mathsf{started}-\mathsf{by}, \mathsf{contains}, \mathsf{finished}-\mathsf{by}\}v'$ *then* $v^* = v'$*, (3) if* $v\ \mathsf{overlaps}\ v'$ *then* $v^*\ \mathsf{finishes}\ v$ *and* $v^*\ \mathsf{starts}\ v'$*, (4) if* $v\ \mathsf{overlapped}-\mathsf{by}\ v'$ *then* $v^*\ \mathsf{starts}\ v$ *and* $v^*\ \mathsf{finishes}\ v'$*.*

Certificate. The rule for certificates is the simplest: if $\mathsf{cert}_k\ (c\ [t_b, t_e]) \in \mathcal{C}$ then for the corresponding $v_c \in TI$, add $\{c : v_c\}$ to $\mathsf{F}(k)$. In words:if the certificate $\mathsf{cert}_k\ (c\ [t_b, t_e])$ is valid then the corresponding formula must be true for the corresponding key during the validity interval of the certificate.

Speaking for. The rules consider both positive and negative cases.

- if $p \mapsto k' : v \in \mathsf{F}(k)$ then add $\{p : v\}$ to $\mathsf{N}(k, k')$
- if $p \mapsto k'\,'\,q : v \in \mathsf{F}(k)$ then $\forall k^* \in \mathsf{K}$ such that $q : v' \in \mathsf{N}(k', k^*)$ either one has $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{p : v^*\}$ to $\mathsf{N}(k, k^*)$
- if $p \mapsto n\,'\,q : v \in \mathsf{F}(k)$ (where q can be null) then $\forall k' \in \mathsf{K}$ such that $n\,'\,q : v' \in \mathsf{N}(k, k')$ either one has $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{p : v^*\}$ to $\mathsf{N}(k, k')$
- if $\neg(p \mapsto q) : v \in \mathsf{F}(k)$ then for a new action a^* we add $\{p\ \mathsf{perm}\ a^*) : v, \neg(q\ \mathsf{perm}\ a^*) : v\}$ to $\mathsf{F}(k)$

The intuition behind the first rule is that if p is associated to k' for the key k then we add the labelling to the relation, tagged with the appropriate validity interval v. The graphical representation is shown in Figure 3

The intuition behind the second rule is the following: suppose that you have a claim that for the key k the name p has been associated to the name $n\,'\,q$ for a certain validity interval v. We can have a look at all naming relations between keys that have q as their name. These naming relations will also have their validity period, say v'. Now we have two possibilities. The first one is that the validity periods do not overlap (that is $v \cap v' = \emptyset$) and therefore there is nothing that we need to do. The second one is

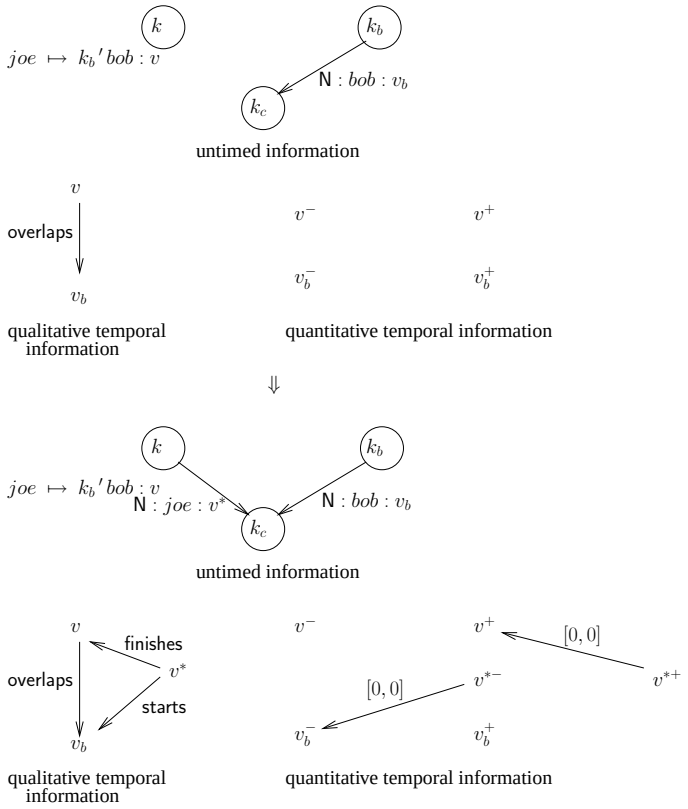

Fig. 4. Illustration of one of the rules for $\mapsto$ (where v overlaps v')

that the validity periods do overlap and then we must chain the two certificates for the overlapping periods, namely $v^* = v \cap v'$.

We illustrate the second rule in the special case where v overlaps v' in Figure 4. As we can see from the figure we have added the link between k and k^* but only for the overlapping interval v^*. The temporal information says that when v finishes then v^* also finish and when v' starts then v^* also starts.

Permission. These rules have the same flavour of the speaks for rules, excpet that they add permitted actions to each key rather then connecting keys with a naming relation.

- if k' perm $a : v \in \mathsf{F}(k)$ then add $\{a : v\}$ to $\mathsf{A}(k, k')$
- if $k'\,'\,q$ perm $a : v \in \mathsf{F}(k)$ then $\forall k^* \in \mathsf{K}$ such that $q : v' \in \mathsf{N}(k', k^*)$ then either $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{a : v^*\}$ to $\mathsf{A}(k, k^*)$
- if $n\,'\,q$ perm $a : v \in \mathsf{F}(k)$ (where q can be null) then $\forall k' \in \mathsf{K}$ such that $n\,'\,q : v' \in \mathsf{N}(k, k')$ then either $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{a : v^*\}$ to $\mathsf{A}(k, k')$

We also have a rule for negated permissions.

- if $\neg k'$ perm $a : v \in \mathsf{F}(k)$ then for a new interval $v^* \subseteq v$, add $\{\neg a : v^*\}$ to $\mathsf{A}(k, k')$
- if $\neg k' \, ' \, q$ perm $a : v \in \mathsf{F}(k)$ then for a new $k^* \in \mathsf{K}$ and a new interval $v^* \subseteq v$, set $\mathsf{N}(k', k^*)$ to $\{q : v^*\}$ and $\mathsf{A}(k, k^*)$ to $\{\neg a : v^*\}$
- if $\neg n \, ' \, q$ perm $a : v \in \mathsf{F}(k)$ (where q can be null) then for a new $k^* \in \mathsf{K}$ and a new interval $v^* \subseteq v$, set $\mathsf{N}(k, k^*)$ to $\{n \, ' \, q : v^*\}$ and $\mathsf{A}(k, k^*)$ to $\{\neg a : v^*\}$

Rules for principals. These rules refine the naming relations eliminating compound names when validity intervals allow to do that.

- if $k^* \, ' \, q : v \in \mathsf{N}(k, k')$ then add $\{q : v\}$ to $\mathsf{N}(k^*, k')$
- if $n \, ' \, q : v \in \mathsf{N}(k, k')$ then $\forall k^* \in \mathsf{K}$ such that $n : v' \in \mathsf{N}(k, k^*)$ either $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{q : v^*\}$ to $\mathsf{N}(k^*, k')$
- if $q : v \in \mathsf{N}(k, k')$ and $q' : v' \in \mathsf{N}(k', k^*)$ then either $v \cap v' = \emptyset$ or let $v^* = v \cap v'$ and add $\{n \, ' \, q : v^*\}$ to $\mathsf{N}(k, k^*)$.

Rule for consistency. A principal associated to a key cannot be permitted and forbidden the same action during overlapping periods.

- if $a : v \in \mathsf{A}(k, k')$ and $\neg a : v' \in \mathsf{A}(k, k')$ then

$$\mathsf{E}(v, v') = \mathsf{E}(v, v') \cap \{\mathsf{before}, \mathsf{after}, \mathsf{meets}, \mathsf{met} - \mathsf{by}\}$$

Finally, we must guarantee the consistency of the qualitative and the metric temporal information. To this extent we need to solve the corresponding constraint network and if the empty interval is present then the original information is inconsistent [15]. This algorithm is sound but not complete unless the interval network is at least *preconvex* [6]. To get completeness of the processing of the temporal information, we modify slightly some rules: whenever we need to add $v \cap v' = \emptyset$ we actually split it into $v\{\mathsf{before}, \mathsf{meets}\}v'$ or $v\{\mathsf{met} - \mathsf{by}, \mathsf{after}\}v'$.

Definition 3. *A branch is* saturated *when no new information can appear through the application of a rule. A branch is* closed *if an inconsistency is found ; it is* open *if it is saturated and not closed. A tableau is closed when all its branches are closed, it is open if one of its branches is such.*

Theorem 1. *If a boolean combinations of certificates χ has a closed tableau given a set of certificates $\mathcal{C}$ iff χ is a logical consequence of $\mathcal{C}$.*

5 Discussion and Conclusions

In the security literature there has been a number of proposals for the right logical account for SDSI/SPKI features. Abadi [1] has used the DEC-SRC calculus for access control [2]. However a number of problems have been found in the DEC-SRC calculus by other authors [17, 12]. Howell and Kotz [14] have proposed an alternative semantics, but their solution is logically rather awkward (for instance it is not closed under the usual boolean operators) and does not give a reasoning procedure. Halpern and Van

der Meyden [12, 13] have refined the semantics of Abadi and their proposal is the basic reference for all subsequent works (including this one). They have proposed two modal logics to reason about it but their proposal is fairly tailored on the SDSI/SPKI framework. Jha, Reps and others [20, 21] have given a pushdown automata procedure for SPKI certificates but have only focused on the normal trust relationship assuming time interval fixed. this is the most comprehensive treatment, and it is a decision procedure for time-free fragment of Halpern and van der Meyden logic. Dropping time from our framework it is a sibling of our model. However, in all papers the treatment of time is either absent (Abadi, Howell and Kotz, Jha and Reps) or refers essentially to the SPKI algorithms (Halpern and Van der Meyden).

With respect to the trust management systems of Li et al. [16, 19] we only have propositional rules. In contrast, Li et al. constructions based on logic programming and Datalog allows for quantification. Since term creating functions are absent, quantification is a just a more efficient and compact representation of the propositional version but does not introduce new possibilities. However, the semantics is just the term algebra of the Datalog programs representing a policy. It could be intriguing to have a independent semantics for the bounded delegation framework.

Conceptually, it is possible to lift our framework to first order reasoning over objects and it would be interesting to derive syntactic restrictions on quantification in certificates that would allow for the the same decidability results based on Datalog in Li el al [19]. The lifting of permissions can be done without difficulties: one could simply import techniques from first order modal logics. Basically, we could have certificates like $\mathsf{cert}_k\ (p\ \mathsf{perm}\ a(o)\ [t_b, t_e])$ when permission to perform action a is only granted on object o and $\mathsf{cert}_k\ (p\ \mathsf{perm}\ \lambda x.a(x)\ [t_b, t_e])$ when permission is granted on all objects.

The lifting of naming association is the tricky bit, as there is no simple security intuition on the meaning of "key k is associated to name n for the object o" (though it is possible to write formulae linking them). A convincing semantics could then be given only for a convincing security intuition of the above concept. Even looking at neighbor field, such as grammar logics or description logics we don't have anything remotely resembling such ternary relation.

Once this framework were lifted to first logic would then be interesting to analyze the relations between the two frameworks. In particular one could then try to derive syntactic restriction on the general model that would make possible to use the Datalog representation and inference engine. Building on this result one could then show the relation with other works on credential based systems that are based on logic and that cope with the proper aspect of trust negotiation (which is neglected in the original SPKI proposal) such as the work of Yu et al. [24] and Bonatti and Samarati [5]

An intriguing subject of future research is the usage of *symbolic validity intervals*. For instance, if we had $\mathsf{cert}_k\ (\mathsf{FAST'co\text{-}chair} \mapsto \mathsf{Martinelli}\ [12/07/04, \mathsf{NextFAST}])$ and then have symbolic constraints on intervals such as NextFAST is non-overlapping with 2004 and overlaps with part of 2002 and starts after [appointment-day,end-of-conference]. The calculus we have present can indeed cope with such constraints. The lifting of the SPKI framework to symbolic validity intervals looks promising for some distributed applications.

Building upon previous attempt of formalizations we provided a general model for reasoning about naming and identifiers, authorization, credentials, and time. We show how to construct a general reasoning method for the logic that combines advanced tableaux methods for modal and description logics [7] with systems for reasoning about the interval algebra by Allen [4] and advanced proposals that exploit both qualitative and metric constraints [18, 15].

References

1. M. Abadi. On SDSI's Linked Local Name Spaces. *JCS*, 6(1-2):3–21, 1998.
2. M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A Calculus for Access Control in Distributed Systems. *TOPLAS*, 15(4):706–734, 1993.
3. J.F. Allen. An Interval-Based Representation of Temporal Knowledge. In *IJCAI'81*, 1981.
4. J.F. Allen. Maintaining Knowledge about Temporal Intervals. *CACM*, 26(11):832–844, 1983.
5. Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the Web. In *ACM CCS-00*, pages 134–143. ACM Press, 2000.
6. J.-F. Condotta. The Augmented Interval and Rectangle Networks. In *KR'2000*, 2000.
7. G. De Giacomo and F. Massacci. Combining Deduction and Model Checking into Tableaux and Algorithms for Converse-PDL. *Inf. and Comp.*, 162(1–2):117–137, 2000.
8. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *AIJ*, 49(1–3):61–95, 1991.
9. J. DeTreville. Binder, a logic based security language. In *IEEE S&P-2002*, 2002.
10. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. *SPKI Certificate Theory*, September 1999. IEFT RFC 2693.
11. D. Georgakopoulos and M. Papazouglou. Service-oriented computing. *CACM*, 2003. To appear.
12. J. Halpern and R. van der Meyden. A Logic for SDSI's Linked Local Name Spaces. *JCS*, 9(1/2):105–142, 2001.
13. J. Halpern and R. van der Meyden. A Logical Reconstruction of SPKI. In *CSFW'01*, 2001.
14. J. Howell and D. Kotz. A Formal Semantics for SPKI. In *ESORICS 2000*, 2000.
15. H. A. Kautz and P. B. Ladkin. Integrating Metric and Qualitative Temporal Reasoning. In *AAAI-91*, 1991.
16. N. Li. Local names in spki/sdsi. In *IEEE CSFW 2000*, pages 2–15, 2000.
17. F. Massacci. Tableaux Methods for Formal Verification of Multi-Agent Distributed Systems. *JLC*, 8(3):373–400, 1998.
18. I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *AAAI'91*, 1991.
19. J. Feigenbaum Ninghui Li, B. N. Grosof. A practically implementable and tractable delegation logic. In *IEEE S&P 2000*, pages 27–42, 2000. Full version appears in ACM TISSEC.
20. T. Reps S. Jha. Analysis of spki/sdsi certificates using model checking. In *IEEE CSFW-2002*, 2002.
21. S. Schwoon, S. Jha, T. Reps, and S. Stubblebine. On generalized authentication problems. Technical report, Univ. of Winsconsin, 2003. To appear in CSFW 2003.
22. Paul Syverson. The use of logic in the analysis of cryptographic protocols. In *IEEE S& P-91*, pages 156–170, 1991.
23. S. Weeks. Understanding trust management systems. In *IEEE S&P-2001*, 2001.
24. Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM TISSEC*, 6(1):1–42, 2003.

Revocation in the privilege calculus ⋆

Babak Sadighi Firozabadi[1] and Marek Sergot[2]

[1] Swedish Institute of Computer Science (SICS)
babak@sics.se
[2] Imperial College of Science, Technology and Medicine
mjs@doc.ic.ac.uk

Abstract. We have previously presented a framework for updating privileges and creating management structures by means of authority certificates. These are used both to create access-level permissions and to delegate authority to other agents. In this paper we extend the framework to support a richer set of revocation schemes. As in the original, we present an associated calculus of privileges, encoded as a logic program, for reasoning about certificates, revocations, and the privileges they create and destroy. The discussion of revocation schemes follows an existing classification in the literature based on three separate dimensions: resilience, propagation, and dominance. The first does not apply to this framework. The second is specified straightforwardly. The third can be encoded but raises a number of further questions for future investigation.

1 Introduction

A public key certificate (PKC) is a data record digitally signed by the private key of its issuer, a certificate authority (CA). A PKC can be seen as a statement by its issuer to certify that there is a binding between an identity (a distinguished name) and a certain public key. Revocation of a PKC can be seen as another statement saying that from a certain time point—the time-stamp of the revocation or the time-stamp of the revocation list containing the revocation—the binding given in the PKC no longer holds. The usual reason for revoking a PKC is that the private key matching the public key given in the certificate is lost or stolen.

Sometimes, in addition to the identity of the key holder, a PKC contains other information, e.g., the key holder's affiliation. As a consequence, a PKC sometimes has to be revoked not because the private key is lost, but because the affiliation of the key holder has changed. However, with the introduction of new types of digital certificates, i.e., *attribute certificates* (AC), there will often be cases in which one has to revoke a certificate because the attribute (privilege) given in the certificate needs to be deleted. For example, a certificate containing an access permission may get revoked because the permission given

⋆ This research was supported in part by Policy Based Management Project funded by the Swedish Agency for Innovation Systems, and by AMANDA project funded by Microsoft Research in Cambridge, UK.

in that certificate does not hold any longer and not because the private key used to sign the certificate has been exposed.

In an earlier paper [FSB01] we presented a framework for updating privileges in a dynamic environment by means of *authority certificates* in a Privilege Management Infrastructure. These certificates can be used to create access-level permissions but also to *delegate* authority to other agents, thereby providing a mechanism for creating management structures and for changing these structures over time. We presented a semantic framework for privileges and certificates, and an associated calculus, encoded as a logic program, for reasoning about them. The framework distinguishes between the time a certificate is issued or revoked and the time for which the associated privilege is created. This enables certificates to have prospective and retrospective effects, and allows us to reason about privileges and their consequences in the past, present, and future. The calculus provides a verification procedure for determining, given a set of declaration and revocation certificates, whether a certain privilege holds.

In our earlier presentation we restricted attention to the simplest form of revocation only. The present paper expands the framework for managing authorities to support a richer set of revocation schemes. The ideas in this paper has earlier been presented in the position paper [FS02]. The current paper extends the ideas and incorporate them in the privilege calculus.

1.1 Revocation (deletion) Classification

In [HJPW01], the authors classify revocation schemes based on three dimensions—*resilience*, *propagation*, and *dominance*—which can be combined in various ways to provide several distinct categories. Although these authors do not consider revocation of *certificates*, but rather deletion of privileges that have been granted in a delegation chain, and although their way of representing privileges and delegations is not the same as ours, we nevertheless find it instructive to consider how that classification might apply to the classification of revocation schemes in our framework. The three dimensions are:

1. **Resilience**: This concerns how a privilege may be revoked in such a way that its effects are persistent, that is to say, in such a way that no other agent may subsequently re-create it. For example, this is the effect that is obtained by creating a 'negative privilege' that will always override its positive counterpart. In this way, the subsequent granting of the positive privilege will never have an effect, since it will always be overriden by the negative one.
2. **Propagation**: This concerns how revocation of a privilege may have indirect effects on other privileges stemming from it in a delegation chain.
3. **Dominance**: This concerns how an agent may be allowed to revoke privileges that are not directly delegated by him. For example, the case considered by [HJPW01] is one in which an agent retains the ability to revoke not only his own delegations, but those of any other agents whose privileges were obtained in a delegation chain stemming from him.

Of these three, the first, resilience, is not applicable to our framework. We do not support the granting of negative privileges. Put another way, our framework deliberately separates privileges from the means by which they are created (here, the issuing of certificates). It is meaningful in our framework to revoke *certificates*, and thereby indirectly delete privileges, but it is not meaningful to revoke privileges directly.

The other two dimensions, however, do apply, and are examined in the body of the paper.

2 Managing Authorities

In ITU-T Recommendation X.509 (2000) [X.500], the standard is extended to include Privilege Management Infrastructure (PMI) as well as Public Key Infrastructure (PKI). PMI is similar to PKI, but its purpose is to give an infrastructure for issuing and managing attribute certificates for assigning and conveying privileges.

The main components of a PMI are: Attribute Certificates (AC), Sources of Authority (SoA), Attribute Authorities (AA), and Attribute Certificate Revocation Lists (ACRL). An attribute certificate is, like a public key certificate, a digitally signed statement (in the form of a data structure) certifying the binding between the holder of the certificate and some of his privileges. A privilege in an attribute certificate can be, for example, an access permission, a group membership, or a role assignment. A SoA is an agent who is recognised by the users as initially empowered to create and delegate certain privileges. An AA is an agent who has been delegated, by the SoA, the authority to create a privilege. A number of AAs can create an *authority management structure* in which authorities have been delegated from the top AA, e.g. the SoA, to subordinates. An ACRL is a list of references of attribute certificates that are no longer to be considered valid.

The framework introduced in our earlier paper [FSB01] provides a means for creating and updating authority management structures such as an AA structure in the PMI model of X.509 (2000). In that paper we distinguish between an access level permission and a management level authority. $perm(s, a, o)$ represents an *access level permission* which says agent s is permitted to perform action a on object o. $auth(s, p)$ represents an *management level authority* which says agent s has the authority to bring about privilege p. Here, p can either be an access-level permission or another management-level authority. We use the term *privilege* to refer both to an access level permission and a management level authority. Note that having the authority to create a privilege does not necessarily mean that one has or can enjoy that privilege oneself; nor that one has the authority to create that privilege for oneself.

A certificate stating that its holder has the authority to create a privilege by means of issuing another certificate is called here an authority certificate. Neither in PMI [X.500] nor in other digital certificate frameworks there is a direct support for encoding management authorities in certificates, however one

can see an authority to create a privilege as an permission to delegate that privilege which can be encoded in attribute certificates.

In [FSB01], we considered only the simplest type of revocation, one that makes it possible to invalidate a certificate from the time the revocation occurs for all times in the future. In the current paper, we generalise this revocation type, allowing the revoker to disable a certificate in the past, present, or future, permanently or just for some specified period of time. This means that the time at which the revocation occurs and the time for which the revoked certificate becomes invalidated are independent. Furthermore, in the previous framework we allowed only the issuer of a certificate to revoke it (though it was possible to get other effects indirectly). In this paper we consider a number of alternatives, under the heading of 'dominance'.

2.1 The Framework of Authority Management

The framework contains only two type of actions: the issuing of certificates and the revoking of certificates.

– Certificates are represented as:

$$\mathit{certifies}(\mathit{issuer}, p[I], \mathit{time\text{-}stamp}, \mathit{id}).$$

The intended reading is as follows: the *issuer* makes an attempt, at time *time-stamp* to bring about that privilege p holds for the time interval $[I]$. We say that the certificate *certifies* (or sometimes 'contains') the privilege p, and we call $[I]$ the *validity interval* of the certificate. If p is a permission then the action in p is an access level action, e.g. read or write a file. If p is a management level authority, of the form $\mathit{auth}(s, q)$, then the action in p is the creation of a privilege q for s. The *id* is the unique identifier of the certificate.

For simplicity we leave out all the parts of a certificate management system that do not have any impact on the reasoning required to determine whether a given privilege holds at a given time. In particular, validation of signatures is of course an essential component of verifying a certificate, but signatures are not part of the reasoning process for verifying that a privilege holds, and for this reason signatures do not appear in our representation of certificates.

– Revocations are represented as:

$$\mathit{revokes}(\mathit{issuer}, \mathit{id}, [I], \mathit{time\text{-}stamp}).$$

Revocations, like certificates, are seen as time-stamped statements. In contrast to certificates, a revocation does not have an *id*, but it contains the *id* of the certificate which is the subject of the revocation. The interval $[I]$, called the *disabling interval*, represents the period of time for which the revocation disables the certificate with the id *id*. By specifying the disabling interval, revocation can be used to disable the particular instance of the privilege in that certificate either temporarily or permanently. Note that revocation works on *certificates*: if the

same privilege is created by several different certificates, revoking only one of them will not necessarily disable the privilege itself. All of the certificates would have to be revoked for that to happen.

Given a historical database of certificates and revocations, it is possible to determine whether a privilege p holds at a given time. Informally: a privilege p holds at a time-point t when there is a certificate c declaring that p holds for some interval I containing t; the certificate c moreover must be 'effective' at t, in the sense that it was issued by s at a time when s had the authority to declare p to hold for interval I. The authority of s, in turn, requires a certificate that was effective at the time c was issued — and so on, in a chain of effective certificates back to some source whose authority can be accepted without certification (as determined by the organisational structure).

Now, we give a number of definitions to make these ideas more precise. We assume that there is a historical database recording the issued certificates and their revocations. This database may be stored in a distributed form: the only requirement is that the reasoning engine for determining whether a certain privilege holds has access to the information.

Definition 1. *A certificate c_1 **directly supports** another certificate c_2 if*

1. *the privilege given in c_1 is the authority for the issuer of c_2 to bring about the privilege given in c_2 at the time of issuance of c_2, and*
2. *c_1 is not disabled at the issuance time of c_2.*

*If condition 1 holds we say that the privilege given in c_1 **validates** the certificate c_2.*

Note that this definition refers only to the time point at which c_2 is issued. If c_1 becomes disabled at any other time, that is to say, for some time period not containing the issuance time of c_2, the support of c_1 for c_2 will not be affected.

Definition 2. *A set of certificates $c_1 \dots c_n$ is a **chain** if each c_i directly supports c_{i+1}, for $1 \leq i < n$.*

A chain represents an authority management structure created by a number of authority certificates. A chain usually, but not always, ends in an end-entity attribute certificate containing an access level permission.

Definition 3. *A certificate c_i in a chain $c_1 \dots c_n$ **indirectly supports** a certificate c_j, if $i + 1 < j$ and $1 \leq i < n$.*

In any application there should be a way of defining who is a source of authority and in what circumstances. For example, in many applications the owner of an object is considered to be the source of authority for any privilege concerning that object. We assume that there is a specified way of recognising sources of authorities, either because the SoA relation between an agent and a privilege is given explicitly, or by means of a set of rules which defines this relation.

Definition 4. *A certificate is called* ***rooted*** *if it is issued by the source of authority of the particular privilege given in the certificate.*
A chain of certificates is ***rooted*** *if the first certificate in the chain is rooted.*

In this framework, anyone can issue a certificate at any time with or without having the necessary authority to make it effective. Without the necessary authority, the certificate has no effect. However, it is possible for a certificate c_1 to become supported, retrospectively, by another certificate c_2 issued at a time later than c_1. This happens when the validity interval of c_2 contains the issuance time of c_1. Examples of the use of such retrospective mechanisms are provided in [FSB01].

Definition 5. *A chain that is not rooted is called a* ***dormant chain****.*
A certificate is called ***dormant*** *at time t if it is part of a dormant chain at time t.*

Since a certificate can be revoked permanently or for a specified time period only, we say that a revoked certificate is disabled.

Definition 6. *A certificate c_1 is* ***disabled*** *at time t if there is a revocation for c_1 and t is contained in the disabling interval of c_1.*

Definition 7. *A certificate c is* ***effective*** *at time t if it is rooted at time t, t is at or after the time of issuing of c, and c is not disabled at time t.*

Definition 8. *A privilege P* ***holds*** *at time t if there is a certificate c that certifies P, c is effective at time t, and t is contained in the validity interval of c.*

2.2 Privilege Calculus

Here we present a logic program which implements the framework given above, encoding in an executable form the definitions given in the previous section. It provides a means of evaluating queries of the form

holds(P, T)

to determine whether a privilege P holds at time-point T given a database of time-stampled certificates and revocations. The program presented below can be executed as a Prolog program as it stands, once the symbol ':' is declared as an infix functor. It can also be executed in other logic programming systems to give the same results but with different computational behaviour. We use a term of the form $[T_1, T_2]$ to represent the closed-interval $[T_1, T_2]$, and a term of the form $since(T_1)$ to represent the open-ended interval of all time points later than or equal to T_1.

Rather than the *holds(P, T)* program, it is very straightforward to produce a generalisation, which is the version we present here. The 3-ary predicate

$holds(P, T, T_D)$ represents that, according to the certificates and revocations issued up to and including time T_D, privilege P holds at time T. This generalized form allows one to query not only the current state of the certificates and revocations database, but all past states as well. This can very useful for auditing purposes, for example, or for determining the effects of retroactive certificates and revocations. (The simpler, less general version of the program, may be obtained by deleting all occurrences of the parameter T_D, and all conditions in which it appears.)

[PC 0.] T *during_interval* $[T_s, T_e] \quad \leftarrow T_s \leq T \leq T_e.$
T *during_interval since*$(T_s) \leftarrow T_s \leq T.$

[PC 1.] $holds(P, T, T_D) \leftarrow C = certifies(S, P{:}I, T_0, ID),\ T_0 \leq T_D,$
$effective(C, T, T_D),$
T *during_interval* $I.$

[PC 2.] $effective(C, T, T_D) \leftarrow C = certifies(S, Priv, T_0, ID),\ T_0 \leq T_D,$
$T_0 \leq T,$
$rooted(C, T_D),$
not disabled$(C, T, T_D).$

[PC 3.] $rooted(C, T_D) \leftarrow chain(C1, C, T_D),$
$C_1 = certifies(S, Priv, T_0, ID),\ T_0 \leq T_D,$
$sourceOfAuthority(S, P).$

[PC 4.] $chain(C, C, T_D).$

[PC 5.] $chain(C1, C2, T_D) \leftarrow supports(C1, C2, T_D).$

[PC 6.] $chain(C1, C2, T_D) \leftarrow supports(C1, C3, T_D),\ chain(C3, C2, T_D).$

[PC 7.] $validates(auth(S, P){:}I, C, T_D) \leftarrow C = certifies(S, P, T, ID),\ T \leq T_D,$
T *during_interval* $I.$

[PC 8.] $supports(C1, C2, T_D) \leftarrow C1 = certifies(S_1, Priv, T_1, ID_1),\ T_1 \leq T_D,$
$C2 = certifies(S_2, Priv', T_2, ID_2),\ T_2 \leq T_D,$
$validates(Priv, C2, T_D),$
not disabled$(C1, T_2, T_D).$

[PC 9.] $disabled(C, T, T_D) \leftarrow C = certifies(S, Priv, ID, T_0),$
$revokes(S, ID, I, T_1),$
$T_0 < T_1 \leq T_D,$
T *during_interval* $I.$

3 Revocation schemes in the privilege calculus

In this framework we are concerned with the revocation of *certificates*, and for reasons already explained, the resiliance dimension of [HJPW01] does not apply. We now consider how the propagation and dominance dimensions apply to the revocation schemes of ourframework.

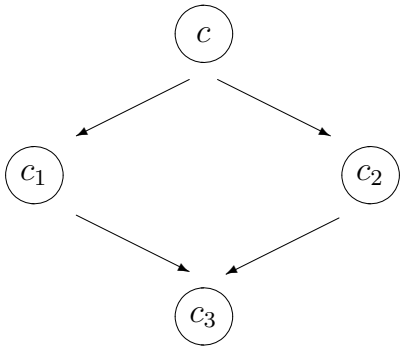

Fig. 1. c, c_1, c_2, and c_3 are certificates. An arrow between two certificates represents the support relation between the two in the direction of the arrow. For example, the arrow between c and c_1 represents that c supports c_1. A deliberate feature of the framework is that the issuer of a certificate does not have to identify the certificate that grants him the necessary authority to issue a new certificate. It is possible therefore that the same certificate can be supported by more than one certificate, as shown here. Both c_1 and c_2 could have created a privilege that made the issuing of certificate c_3 effective.

3.1 Simple Revocation

The simplest revocation scheme is the one in which an agent revokes one of his own issued certificates simply in order to withdraw the privilege given in that certificate from the time of revocation onwards. In the case where the privilege in the revoked certificate is an access-level permission, the effect of the revocation is that this particular instance of the permission does not hold at any time after the issuance of the revocation. In the case where the privilege in the revoked certificate is a management level authority the effect is that the certificate can no longer support any new certificate issued after the issuance time of the revocation. However, revocation of this certificate will not affect any other existing certificates—any certificates created by a delegation chain from the revoked certificate will continue to be effective (unless revoked directly).

This simple scheme is easily specified in the privilege calculus by a revocation with a disabling interval that starts at the revocation time and extends (open-ended) into the future: the revocation has the form $revokes(issuer, id, since(ts), ts)$.

This implies that from the time of the revocation and for any time after that, the particular instance of the privilege given in the revoked certificate is deleted.

In the diagram above, if the issuer of c revokes c at any time t after the issuance of c_1, c_2, and c_3, the revocation will not affect certificates c_1, c_2, c_3, but c cannot be used to support any new certificate issued after time t.

3.2 Propagation of revocations

Sometimes one needs to revoke a certificate in such a way that it affects the validity of some other certificates. The typical scenario is when one discovers that an agent has abused his authority. If this (perhaps fraudulent) agent is in a management role in which he has delegated privileges to others, then often one wants to delete not only his authority but also all those privileges that were delegated by him, and by his delegatees in turn.

This kind of propagation of revocations can be specified in the privilege calculus by disabling the certificate that made the fraudulent agent an authority. The certificate has to be disabled in such a way that its support for other certificates vanishes. This can be done by a revocation that has a disabling interval containing all the time points, in the past and in the future, at which that certificate supports other certificates. This is similar to: I say now that what I said in the past was not true, hence every other statement based on what I said then does not hold either. The framework we are presenting is explicitly designed to support such retrospective effects. Note that we are reasoning in two different time lines: one is the time of the certificate database, and the other is the time at which a privilege in a certificate holds.

Example: In the above picture, certificate c, issued by agent a, *directly supports* certificates c_1, issued by a_1 at time t_1, and c_2, issued by a_2 at time t_2. Further assume that a_1 and a_2 are different agents both included in the group of agents who receive the privilege given in c and that $t_1 \neq t_2$.

If at a certain time t_x, a finds out that a_1 is a fraudulent agent, the authority given to a_1 has to be deleted, immediately. But not only does the authority given to a_1 have to be deleted, the authorities delegated by a_1 also have to be deleted. Beside revoking c from time t_x, a must disable c such that its support for c_1 disappears. a can do this by first revoking c at t_x such that c cannot be used by a_1 at any time after t_x. This is a simple revocation of c at time t_x by its issuer. In order for a to delete privileges delegated by a_1, he has to revoke c again, but this time in such a way that the support relation between c and c_1 disappears. Of course, one could consider a revocation format that allows several disabling intervals to be specified in one revocation statement. One can similarly devise other kinds of 'macros' for commonly occurring patterns of revocation statements.

Note that, if c does not support c_1 any longer then automatically c_1's support for c_3 also disappears. Further, the support of c for c_2 remains the same if the disabling interval does not include t_2. Hence, the authority management structure created by a_2 remains untouched. However, for a_2 to be able to exercise,

at any time after t_x, the same authority he once received in c, a has to issue a new certificate c', at time t_x, that gives a_2, but not a_1, the same privilege that was given to him in c.

3.3 Dominance

In the framework presented so far, it is only the issuer of a certificate that has the possibility to revoke it. Relaxing this constraint will complicate the framework as well as the privilege calculus, but it is necessary if we want to support revocation schemes based on the dominance dimension. But why do we need this?

It is a deliberate feature of our framework that agent x who issues a certificate does not have to identify the certificate which grants him the necessary authority: the certificate C does not have any record of which other certificate is being invoked when certificate C is issued. This is by design. It is not realistic in our view to require that agents say "I am issuing this certificate by the authority given to me by certificate X". Agent x may not know the identifier of X, and in the case of a dormant chain, there is no such X (yet).

But now consider. Bjorn goes on holiday for the weekend and we are unable to revoke any of the certificates he issued until he gets back. The only way is to try to discover who has issued certificates to Bjorn, and then to ask each of them to revoke their certificates so Bjorn's privileges are revoked. That is clearly ridiculous. Some of the problem can be mitigated by introducing some kind of representation mechanism, allowing Bjorn to specify who can act for him in his absence. But that does not solve all the problems. It does not deal with the case where Bjorn has forgotten to appoint a representative, or done so deliberately.

The most general approach would be to decouple entirely the authority to grant privileges from the authority to revoke certificates, i.e., by introducing separate predicates $auth^+(s, p)$, which says "s has the authority to bring about p", and predicate $auth^-(s, p)$, which says "s has the authority to delete p". This would enable us to delegate each of these authorities separately. We would be able to give the authority to create a privilege to one agent and the authority to delete that privilege to a different agent.

Although this general approach would cover many interesting scenarios, and has great flexibility, we believe that it would also make the framework too powerful and too difficult to manage. Therefore we consider a less general approach, similar to that discussed under the heading of 'dominance' by [HJPW01]. Here, an agent is given the authority to revoke any certificate issued by him, and any certificate that is issued on the basis of a delegation chain stemming from one of the certificates issued by him.

The rationale is this. When an agent delegates some authority to another agent, he retains some responsibility for the actions of the delegatee. And then it seems only natural to say that the delegator should therefore retain some measure of control over how the delegated authority is used. In a certificate-based framework, this suggests the delegator should be allowed to revoke certificates issued on the basis of his original delegation acts. This also seems to be the reasoning behind the dominance mechanism discussed in [HJPW01].

The framework and the associated calculus of privileges can be modified straightforwardly. One simply replaces [PC 9.] in the privilege calculus with a more elaborate version. An example is given presently.

There are a number of outstanding points of detail to be examined, however. In particular, in a framework such as ours which supports the issuing of certificates with retrospective effects, it is easy to produce undesirable effects. For example, if a wishes to revoke a certificate C issued by b, a could simply issue a certificate granting himself the authority to create the privilege certified by C. a's certificate is not effective, but it is the start of a dormant chain supporting the certificate C. If we are not careful in specifying the dominance relation, a may be allowed the authority to revoke C in these circumstances.

One solution is to restrict attention to *rooted* chains, on the grounds that their validity rests ultimately on a source of authority whose actions need not be questioned. A second check is to compare the times of issuance of certificates in a chain, blocking the retrospective revocation of certificates through the dominance mechanism.

Such questions remain to be explored more systematically. Here we illustrate how the calculus of privileges can be modified to support the form of dominance just discussed. Investigation of other forms of dominance and their properties is a topic of current research for us.

$$\begin{aligned}[\text{PC 9x.}]\quad \mathit{disabled}(C,T,T_D) \leftarrow\ & C = \mathit{certifies}(S_1,\mathit{Priv}_1,\mathit{ID}_1,T_1),\ T_1 \le T_D,\\ & \mathit{revokes}(S_r,\mathit{ID}_1,I,T_2),\ T_2 \le T_D,\\ & T\ \mathit{during\_interval}\ I,\\ & T_1 < T_2,\\ & \mathit{dominant}(S_r,C,T_D).\end{aligned}$$

$$\begin{aligned}[\text{PC 10.}]\quad \mathit{dominant}(S_2,C,T_D) \leftarrow\ & C = \mathit{certifies}(S_1,\mathit{Priv}_1,\mathit{ID}_1,T_1),\\ & \mathit{certifies}(S_2,\mathit{Priv}_2,\mathit{ID}_2,T_2),\\ & C' = \mathit{certifies}(S_2,\mathit{Priv}_2,\mathit{ID}_2,T_2),\\ & \mathit{chain}(C',C,T_D),\\ & \mathit{rooted}(C',T_D).\end{aligned}$$

It is possible to derive equivalent but computationally more efficient formulations. However, there are other implementation issues and options to be considered. We leave detailed discussion for another paper dealing with the practical aspects of the framework.

Example: In the above picture, the issuer of c wants to delete the privilege given in c_3 which is supported by both c_1 and c_2 without deleting the privilege given in c_1 nor the privilege given in c_2. Now, what the issuer of c can do is to revoke c_3 directly without touching the validity of c_1 or c_2.

3.4 Related work

In this paper we only focus on the issue of privilege revocations in terms of certificate revocations. Hence, we will only consider the related work that deals with

the privilege revocation issues. This means that, currently, we do not consider the issues of secure and reliable revocation mechanism for distributed certificates as it is the case in PKI and PMI systems.

Most of the earlier work in revocation of privileges has focused on the consequences of revocations in terms of propagation issue. The revocation schemes discussed above are similar to those addressed in the databases literature. The *grant option* presented in [GW76] is similar to the delegations in our privilege calculus with the main distinction that in the privilege calculus one can delegate an authority to create a privilege to someone without creating the privilege for that person and/or without delegating the authority to that person to create the privilege for himself. In the framework given in [GW76] or its extension given in [Fag78], the grant capability can only be given together with the privilege itself.

In the original authorisation mechanism given in [GW76] if the same privilege is granted by the same grantor to the same subject but at different time-points then only the first one is recorded in a table and the second one is ignored. In [Fag78], the author shows that, because of how the authorisations are recorded in the authorisation mechanism given in [GW76], its revocation mechanism does not perform as it should. Hence the authorisation mechanism is extended such that similar authorisations with different time-stamps are recorded. For more details we refer the reader to [Fag78]. Note that in these authorisation mechanisms one does not revoke a particular exercise of a grant option, but the actual granted privilege, as grants themselves have no identifiers. This is different from our approach in which each delegation has its own identifier in terms of the i.d of its certificate.

A very similar approach to ours is given in [BSJ97], in which a framework for authorisations with a grant option is developed. The framework uses the grant option given in [GW76,Fag78] for further delegation of authorisations to support a decentralised management of authorisations. In this framework, negative authorisations are used for blocking positive authorisations for limited time periods. This is different from the framework given in this paper in which we use revocations with blocking time. The authors also give definitions for cascaded and non-cascaded revocations that are similar to our simple and propagation schemes for revocations.

In [BBFS97] the authors develop a framework for temporal authorisations using time intervals and temporal operators for specifying the time interval during which an authorisation is valid. In the same way a revocation may also have a time-interval that specifies the interval that the revoked authorisation becomes invalid which is similar to the disabling interval in revocations in our framework. However, this framework is also based on the same grant model as the authorisation mechanism in [GW76,Fag78], and it does not support the retrospective authorisations as in the Privilege Calculus.

4 Conclusion

We have presented an extension of our framework for updating privileges by means of authority certificates in order to provide a richer variety of revocation schemes than was originally supported. As in the original, we provide an associated calculus, encoded as a logic program, for reasoning about what privileges hold at which times given a database (or access to such a database) of time-stamped attribute certificates and revocations.

We find the classification scheme for revocation mechanisms introduced in [HJPW01], and the dimensions of resilience, propagation, and dominance identified there, extremely illuminating and helpful in structuring our own investigations. However, by focussing as we do on *mechanisms* for the creation and revocation of privileges, and on *time-dependent* effects, we find that there are a number of further distinctions that can be drawn. The concept of dominance, in particular, seems deserving of further examination, in that we have encountered a number of further points of detail that need to be resolved. We are currently undertaking a more systematic exploration of these possibilities.

A distributed privilege management system requires an architecture for secure and reliable revocation of digital certificates.

References

[BBFS97] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Decentralized Administration for a Temporal Access Control Model. *Information Systems*, 22:223–248, 1997.

[BSJ97] E. Bertino, P. Samarati, and Sushil Jajodia. An extended authorization model for relational databases. *IEEE Transaction on Knowledge and Data Engineering*, 9(1):85–101, 1997.

[Fag78] R. Fagin. On an authorization mechanism. *ACM Tansactions on Database Systems*, 3(3):310–319, Sept 1978.

[FS02] B. Sadighi Firozabadi and M. Sergot. Revocations Schemes for Delegated Authorities. In *proceedings of IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, June 2002.

[FSB01] Babak Sadighi Firozabadi, Marek Sergot, and Olav Bandmann. Using Authority Certificates to Create Management Structures. In *Proceeding of Security Protocols, 9th International Workshop*, Cambridge, UK, April 2001. Springer Verlag. In press.

[GW76] P.P. Griffiths and B.W. Wade. An authorozation mechanism for a relational databse systems. *ACM Transactions on Databases Systems*, 1(3):242–255, 1976.

[HJPW01] Åsa Hagström, Sushil Jajodia, Francesco Parisi.Persicce, and Duninda Wijesekera. Revocation - a Classification. In *The Proceeding of the 14th Computer Security Foundation Workshop*. IEEE press, 2001.

[X.500] ITU-T Recommendation X.509: The Directory - Public-Key and Attribute Certificate Frameworks. published by International Telecommunication Union, 2000.

Information Integrity Policies

Peng Li Yun Mao Steve Zdancewic*

University of Pennsylvania

Abstract. Information integrity policies are traditionally enforced by access control mechanisms that prevent unauthorized users from modifying data. However, access control does not provide end-to-end assurance of integrity. For that reason, integrity guarantees in the form of noninterference assertions have been proposed. Despite the appeals of such information-flow based approaches to integrity, that solution is also unsatisfactory because it leads to a weaker notion of integrity than needed in practice.
This paper attempts to clarify integrity policies by comparing and contrasting access control vs. information flow, integrity vs. confidentiality policies, and integrity vs. availability policies. The paper also examines data invariants as a way to strengthen integrity. The result is a better classification of information-integrity policies.

1 Introduction

Information integrity is a critical issue in computer security, and integrity policies that seek to prevent accidental or malicious destruction of information have long been recognized as important. However, the concept of integrity is difficult to capture and context dependent. Pfleeger's textbook, *Security in Computing*, [26] defines integrity to mean that data is:

- precise
- accurate
- unmodified
- consistent
- modified only in acceptable ways
- modified only by authorized people
- modified only by authorized processes
- meaningful and correct

Traditionally, information integrity has been supported by security models based on *discretionary access control* via access control lists or capabilities. These models mainly provide the authorization component of integrity requirements. However, such enforcement mechanisms are not adequate to deal with situations in which important data in the system may be affected by untrusted sources of information.

For example, the programmer may use a format string provided by the user to print some information, possibly creating a vulnerability to the well-known *buffer overflow attack*. Other examples include using a value received from the network as an array index, or executing a piece of code downloaded from untrusted Web

* Email: {lipeng, maoy, stevez}@cis.upenn.edu

sites. Prior to using dangerous data, the programmer must carefully verify it against possible attacks to assure safety.

Because these potential integrity violations stem from how untrustworthy data propagates through the system, it is tempting to generalize access-control models of integrity to *information-flow* models, which have been widely applied to the problem of protecting confidential information [28]. In fact, it has been observed that information-flow based integrity can be treated as the formal dual to confidentiality [2]. Yet it is also known that doing so yields a somewhat unsatisfactory (and potentially inadequate) definition of security [6, 22].

The goal of this paper is to investigate integrity policies and their relationship to other existing information security policies. The paper is intended to serve as map of the technical landscape surrounding integrity policies; it suggests that integrity be classified into *program correctness*, *noninterference*, and *data invariant* policies. The main contributions of the paper include:

- A critical comparison between confidentiality and integrity when treated as formal duals. This comparison is carried out in the context of Myers and Liskov's *decentralized label model* (DLM) [22].
- A comparison of information-flow style integrity policies as enforced by static program analysis to data invariants (in the style of Clark and Wilson [6]). This comparison is motivated because treating integrity as purely dual to confidentiality leads to weak guarantees.
- A comparison of integrity policies to availability policies.
- Some suggestions for possible future research directions.

2 An aside on program correctness

Ideally, integrity, like any other program property, would be described by a specification and enforced by proving that the implementation satisfies the specification. Such an approach captures the "meaningful and correct" data requirement of Pfleeger's definition. These specifications are especially difficult to pin down and can perhaps best be formalized by requiring *program correctness*, which can be defined as the following [3]: Let π denote (a specification of) a computational decision or search problem. For x an input to π, let $\pi(x)$ denote the output of π. For a deterministic program f, we say that

- The program f is *correct*, if and only if $\forall x, f(x) = \pi(x)$.
- The output $f(x)$ is *correct*, if and only if $f(x) = \pi(x)$.

Program verification [15, 4] aims at proving the correctness of programs with respect to their specifications. *Program correctness checkers* [3] are programs that verify the correctness of the program output.

Due to their expense, program verification and correctness checkers have not been widely applied in practice. This paper instead focuses on formal definitions of integrity policies based on *noninterference* and *data invariants*. These definitions can cover almost all of the meanings of integrity given by Pfleeger and have more tractable enforcement strategies.

3 Information flow

3.1 Access control vs. information flow

In computer security, access control has been widely used to manage the permissions of users to access the objects. *Capabilities* [8, 36] and *access control lists* [17] are often used to implement such policies. In *discretionary access control* (DAC) [17, 13], the owner of an object controls the access to the object by granting and revoking access to other users. *Stack inspection* [35, 11] is a more recently developed dynamic mechanism for enforcing access control checks to sensitive system calls from untrusted code. Both DAC and stack inspection are insufficient in the context of a running system—they do not control how the data is used after granting access to the user.

Information-flow policies [12, 20, 28] are end-to-end security policies that provide more precise control of information propagation than access control models. With access control, a file is accessible to some principal only if the appropriate read permission is specified in the access control list. However, once the file is read, the data can be used in any arbitrary way. Information-flow policies aim to solve this problem by granting file access to only those processes that will not leak confidential data. The policy enforcement is thus extended to the end systems, providing more security than access control alone.

Information flow can be defined as a set of *noninterference* assertions [12, 20]. Intuitively, noninterference requires that high-security information does not affect the low-security observable behavior of the system. Such policies are especially useful to protect data confidentiality, where the goal is to ensure that secret data does not influence public data.

Formally, noninterference for confidentiality can be defined as the following[1]:

- Suppose there are two security levels of data: *secret* and *public*. Let the program f take as input (i_{secret}, i_{public}) and produce the output (o_{secret}, o_{public}).
- Define *noninterference* as a property of the program: f is noninterfering iff

$$\forall a, a' : \ f(a, b) = (c, d) \ \Rightarrow \ f(a', b) = (c', d)$$

That is, nothing about i_{secret} can be learned by only observing o_{public}.

Recent studies have shown that language-based techniques are a promising approach to enforcing noninterference. In particular, static program analysis [7, 33, 14, 21, 27] has demonstrated advantages of little run-time overhead, the capability of managing implicit information flows, and provable guarantees.

3.2 Decentralized label model

As a concrete point of comparison between information-flow definitions of confidentiality and integrity policies, we use Myers and Liskov's *decentralized label*

[1] This simple definition of noninterference is suitable for our discussion—many other more refined variants of noninterference exist in the literature. See the survey by Sabelfeld and Myers [28] for a summary.

model (DLM) [22]. The DLM addresses weaknesses of previous information-flow control models in a distributed setting containing untrusted code or users. It allows users to precisely control information flows, and it also accommodates mutual distrust and selective declassification.

Principals and Labels To address privacy[2] for mutually distrusted users and groups, the DLM uses principals as a central concept. *Principals* are the authority entities such that information is owned, read and written by them. For example, each user or group account in a Windows/Unix machine is a principal.[3]

Labels are the confidentiality requirements that the principals state about their data. A label consists of several policies, each of which is enforced by the decentralized label model. A *policy* is written as $\{o : r_1, r_2, \cdots, r_n\}$, where o represents the *owner* principal who owns the policy; $r_1, \cdots, r_n$ are the *readers*, meaning that the owner o gives $r_1, \cdots, r_n$ permissions to read the data.

A *composite label* consists of zero, one or several policies, e.g. $\{o_1 : r_1, r_2; o_2 : r_1, r_3\}$. This label says the data is owned by both o_1 and o_2. Owner o_1 permits r_1 and r_2 to read it, and owner o_2 permits r_1 and r_3 to read it. To satisfy both requirements, the only effective reader is r_1.

Labels represent different security levels. It is easy to see that label $L_1 = \{o_1 :\}$ represents a stricter policy than label $L_2 = \{o_1 : r_1\}$ because o_1 does allow r_1 to access the data in L_2, but does not in L_1. Such a label relationship is written as $L_2 \sqsubseteq L_1$, and is read as L_1 is more restrictive than L_2. It is shown by Myers and Liskov that all the labels form a distributive lattice, and have a partial order relation in the lattice [22].

When a program tries to compute a result from two values labeled with L_1 and L_2, the result should have the least restrictive label L that enforces all the privacy concerns specified by L_1 and L_2. Namely, $L_1 \sqsubseteq L, L_2 \sqsubseteq L$ and $\forall L'$ such that $L_1 \sqsubseteq L'$ and $L_2 \sqsubseteq L'$ we have $L \sqsubseteq L'$. This least restrictive label is the *least upper bound* or *join* of L_1 and L_2, written as $L_1 \sqcup L_2$. The *greatest lower bound* or *meet* of L_1 and L_2, written as $L_1 \sqcap L_2$ is defined to be the largest label less than both L_1 and L_2.

Declassification During computation, the labels on program results become increasingly restrictive, making the data unreadable. Consequently, the owners of the data sometimes need to relax their policies so that other parties can read it. This kind of label relaxation is called *declassification* [39, 38].

To make declassification reasonable, the decentralized label model permits it only when the current process is authorized to act on behalf of the principals whose policies are weakened. Because a principal can weaken only his own policy, the other owners in the label are safe—their policies are still enforced. No centralized declassification process is needed, making the DLM well suited to distributed, heterogeneously trusted systems.

[2] In this paper, the terms *privacy* and *confidentiality* are considered synonyms.

[3] For the purposes of this paper, we ignore the DLM *actsfor* relationship, which allows one principal to delegate rights to another.

4 Integrity and noninterference

We have seen noninterference policies for protecting data confidentiality. Such policies constrain: (1) Who can *read* the secret data. (2) Where the secret data will flow *to* (in the future).

Dually, integrity policies constrain: (1) Who can *write* to the data. (2) Where the data is derived *from* (in the past, the history of the data).

The analog of "public" is "untainted" and the analog of "private" is "tainted". This style of integrity policy can be defined formally using the same definition of noninterference used for confidentiality:

- Suppose there are two security levels of data: *tainted* and *untainted*. Let the program f take the input $(i_{tainted}, i_{untainted})$ and produce the output $(o_{tainted}, o_{untainted})$.
- Define *noninterference* as a property of the program: f is noninterfering iff

$$\forall a, a' : \ f(a, b) = (c, d) \ \Rightarrow \ f(a', b) = (c', d)$$

  That is, the value of $i_{tainted}$ does not affect the value of $o_{untainted}$.

Integrity policies based on noninterference are useful when we want to control the source of important data. For example, in an encryption algorithm, we may want to generate randomized numbers from trusted sources and to make sure that the adversary cannot affect the value of these numbers. Information-flow control and program analysis can be used to enforce such policies based on noninterference.

Biba [2] first observed that integrity and confidentiality are duals in this way. Consequently, both can be enforced by controlling information flows.

A related concept for integrity policies is the notion of *separation of duties* (see for example Clark and Wilson's paper [6]). The idea of separation of duties is to increase data integrity by requiring that multiple principals collaborate to produce the data—forging a data item is thus more difficult. Such separation is only useful if the two parties are noninterfering in some sense. Shockley [31] and Lee [19] showed how the enforcement parts of the Clark-Wilson paper could be implemented using Bell and LaPadula [1] noninterference mechanisms. They used the Bell and LaPadula model with "partially trusted subjects" to represent the trusted program components.

More recent work by Foley gives a framework [9] for describing label-based policies that supports separation of duties. Foley [10] has also shown how to define integrity in terms of a refinement-based notion of dependability, which is closely related to standard definitions of noninterference.

5 Extending the DLM for integrity

This section discusses an integrity variant of the DLM and compares it to other integrity label models.

5.1 Integrity label models

Before introducing the full decentralized label model for integrity, we first give several simpler integrity label models for comparison and to help readers have a gradual introduction to the label models.

1. **Binary Model** In this model, there are only two possible labels: {tainted} and {untainted}. The meanings of these two labels are clear, and the order relation is {untainted} $\sqsubseteq$ {tainted}. Although the binary model is very simple, it has been used to detect format string exploits [30].
2. **Writer Model** In this model, labels are sets of principals: $\{p_1, p_2, \cdots, p_n\}$. This label means that every principal p_i in the label may have modified the data, and principals not in the set have not affected the data. Therefore, the partial order relation can be defined as: $L_1 \sqsubseteq L_2$ if and only if $L_1 \subseteq L_2$.
   For example, L_1={Alice} and L_2={Alice, Bob}, we have $L_1 \sqsubseteq L_2$ because data with label L_2 could be tainted by Bob, who is not a writer in L_1.
3. **Trust Model** Here, labels are again sets of principals: $\{p_1, p_2, \cdots, p_n\}$, but the meaning differs from the writer model. The interpretation is that a principal *trusts* the data with the label if and only if it is in the label set. Therefore, the partial order relation can be defined as: $L_1 \sqsubseteq L_2$ if and only if $L_2 \subseteq L_1$. The trust model is used in secure program partitioning [40, 41] to ensure data quality and to meet the robust declassification [39] requirement.
4. **Distrust Model** The distrust model is very similar to the trust model. In the distrust model, a label, being a set of principals, means that principals in the set do not trust the data. Consequently, the partial order relation is the opposite to the trust model: $L_1 \sqsubseteq L_2$ if and only if $L_1 \subseteq L_2$

Some of these models can be reduced to others. For example, to represent the binary model in the in the trust model, the label {} can represent {tainted}, and {root} can represent {untainted}. Because root should be trusted by every principal, everyone trusts that the data is untainted. It is not hard to see that trust model and distrust model can be reduced to each other.

5.2 The DLM for integrity

Label Definition Myers and Liskov [22] show how the DLM extends to support integrity in addition to confidentiality. Like a privacy label, an *integrity label* also consists of several policies. Each *policy label* is written as $\{o : w_1, w_2, \cdots, w_n\}$. Here o represents the *owner* principal, meaning that he owns the data; $w_1, \cdots, w_n$ are the *writers*, meaning that the owner o believe that only $w_1, \cdots, w_n$ could have modified the data. The owners and writers are principals drawn from the same set as in privacy labels.

A *composite label* consists of zero or more policies, e.g., $\{o_1 : w_1, w_2; o_2 : w_1\}$. This label says that the data is owned by both o_1 and o_2. Owner o_1 believes that w_1 and w_2 have written to it, and owner o_2 believes that w_1 have written to it. To satisfy both integrity constraints, the only effective writer is w_1.

We intentionally choose this integrity label definition to use the same representation syntax as that of privacy labels to make the duality between them explicit. To avoid confusion, unless the readers can easily tell from the context, we use the convention that L denotes representations, L^P denotes privacy labels and L^I denotes integrity labels. We define $\mathcal{L}$ as the set of all possible label representations. For convenience, we also define a function $R(L)$ to convert an integrity label or a privacy label to its representation, namely, $\forall L \in \mathcal{L}, R(L^P) = R(L^I)$.

Integrity Label Ordering Integrity labels have a partial order relation that expresses their relative security. For privacy labels, $L_1 \sqsubseteq L_2$ if and only if L_2 is as restrictive as L_1 or more restrictive than L_1. For integrity labels, we define $L_1 \sqsubseteq L_2$ if and only if L_2 is as tainted as L_1 or more tainted than L_1. For example, $L_1 = \{o : w_1, w_2\}$ is dirtier than $L_2 = \{o : w_1\}$ because w_2 may have tainted the data in L_1, but not in L_2. Therefore, $L_2 \sqsubseteq L_1$. If we treat L_1 and L_2 as privacy labels, and w_1, w_2 as reader principals, then $L_1 \sqsubseteq L_2$. To summarize: $\forall L_1, L_2 \in \mathcal{L}, L_1^P \sqsubseteq L_2^P$ iff $L_2^I \sqsubseteq L_1^I$ [22].

By this order definition, integrity labels also form a distributive lattice that is exactly the dual of the privacy lattice. That is, if we turn integrity lattice up side down, the two lattices will perfectly match. This property can help us to derive the integrity relabeling rules and computation rules.

Label Computation To keep track of the integrity information-flow property in the program, it is necessary to define the least upper bound operation $\sqcup$ and greatest lower bound operation $\sqcap$ in terms of integrity labels. Because of the dual relation of integrity labels and privacy labels, $R(L_1^I \sqcup L_2^I) = R(L_1^P \sqcap L_2^P)$ and $R(L_1^I \sqcap L_2^I) = R(L_1^P \sqcup L_2^P)$. For composite label $L = \{P_1; P_2; \cdots; P_n\}$, where $L_1 = \{P_1\}, L_2 = \{P_2\}, \cdots, L_n = \{P_n\}$, we have $L^P = L_1^P \sqcup L_2^P \sqcup \cdots \sqcup L_n^P$, and $L^I = L_1^I \sqcap L_2^I \sqcap \cdots \sqcap L_n^I$

Representation Power of the DLM The DLM for integrity is able to represent all the other models we have given. For example, {root: Alice, Bob} in the DLM represents {Alice, Bob} in the writer model. {Alice: ; Bob:} in the DLM represents {Alice, Bob} in the trust model. Because the distrust model is representable by the trust model, it is also representable by the DLM.

Endorsement An analog to declassification also exists for integrity labels. We call this integrity security relaxation *endorsement*. The motivation is that along with the information flow in the program, more and more data may be tainted, and the final results may be useless. It is desirable to cast tainted data to untainted data, perhaps after performing some check on that data. For example, the program may read data from the Internet, a very tainted data source. Enforcing pure noninterference would prevent such data from affecting any untainted data in the program, which is usually too restrictive in practice. Endorsement is intended for such situations. If the program can verify that the data from the network is safe and not tainted, e.g. the digital signature is correct, the program may endorse the data and allow it to flow to untainted places.

In the extended DLM, the endorsement operation is a set of relabeling rules to change the integrity labels. Because we can express mutual distrust integrity constraints, it is important that the endorsement can only weaken the integrity

constraints of the current authority. In particular, if the policy of a principal does not exist in the label, he has the right to add to it.

The endorsement rule for integrity labels can be formalized as follows: If $L_2 \sqcap \{a :\} \sqsubseteq L_1$ and a is a current authority then L_1 can be endorsed to L_2.

Examples We have described how to modify the DLM to support integrity constraints. Now we present several examples in Jif [23], which is a variant of Java language that implements the DLM, to help readers understand how the model works. To distinguish integrity labels from privacy labels, we put "!" in front of labels to represent integrity labels.

```
Example 1                          Example 2
1: boolean !{Alice: Bob} x=true;   1: public int !{Alice:} foo(int !{} in)
2: boolean !{Alice:} y = false;    2:      where authority(Alice)
3: x = y;                          3: {
4: y = x; //this is wrong!         4:  int result = (in>0) ? in:0;
5: if (x) {                        5:  return(endorse(result, !{Alice:}));
6:    y = true;  //wrong,  too!    6: }
7: }
```

In Example 1, variables `x` and `y` are given two different labels. By our integrity label definition, `!{Alice: } ⊑ !{Alice: Bob}`. Therefore, `x` is considered tainted data while y is untainted. Thus, assigning `y` to `x` is legal (line 3) but assigning `x` to `y` is illegal (line 4). Line 6 illustrate the implicit information flow case. Although `y` is not directly modified by any tainted data, `y`'s value depends on the condition value of the control statement in line 5. In fact, if `y` is false before line 5, then line 5 and 6 is equivalent to `y=x`. The compiler rejects such programs by adding another constraint that `y` must be dirtier than the current program-counter (`pc`) label, and in line 5 the `pc` label is augmented to `x`'s label so that the compiler can detect the fault.

In Example 2, the function `foo` takes an argument, tests whether the value is larger than 0, and returns a non-negative result. The interesting part is that the input data could be written by anyone. The program will be rejected without endorsement. With the endorsement statement, the return value of the function is fully trusted by Alice. Note that the endorsement requires Alice to be a current authority. If the current authority is Bob or someone else, the endorsement will not be granted.

6 Comparison of label models

The DLM can be symmetrically defined for both confidentiality labels and integrity labels, but they are not completely symmetric in applications. The problem lies in the motivation to use the DLM: to enforce security policies in presence of mutual distrust. Figure 1 is a comparison of the label models for both confidentiality and integrity; the details are explained in the following subsections.

	Confidentiality Labels				Integrity Labels		
	Trusted Code		Untrusted Code		Trusted Code		
	RM	DLM	RM	DLM	WM	TM/DM	DLM
Mode 1	√	√	√	√	×	×	×
Mode 2	√	√	√	√	√	√	√
Mode 3(a)	×	×	×	√	×	×	×
Mode 3(b)		√		√		√	√

RM = Reader Model TM/DM = Trust/Distrust Model
WM = Writer Model DLM = Decentralized Label Model

Fig. 1. Comparison of confidentiality and integrity label models under various failure modes (see Sections 6.1 and 6.3).

6.1 Revisiting confidentiality

To better understand the motivation of using the DLM to protect confidentiality, we compare the DLM with the *reader model*. The reader model is the dual of the writer model: each label is simply a list of principals allowed to access the information. We consider both trusted (self-written) code, and untrusted (downloaded) code, assuming all the code is typechecked before execution.

We compare the two models in the following failure modes (see the summary for confidentiality labels in the left half of Figure 1):

1. *Wrong Computation*: This is the case where the code performs some wrong computation. For the trusted, self-written code, it may be a bug in the program. For the untrusted code, it may be some malicious statements trying to leak secret information. As long as the code is annotated and typechecked, both the DLM and the reader model are safe, because the typechecking guaranteed that no information will be leaked. In fact, performing the wrong computation on the secret data only make it safer. The secret data only becomes more useless after the wrong computation.
2. *Violating Flow Constraints*: This is the case where the flow of information violates the annotated constraints. Both models are safe in this aspect, because the typechecker will detect such problems.
3. *Wrong Declassification*: This is the case where the code declassifies some secret information that should not be leaked. The reader model is not safe in this case, because there is no relationship between the authority of the code and the principals in the label. If we allow declassification here, there is no way to restrict the declassification in the code that we do not trust. For example, anyone can declassify root's password to the public.
   The DLM works better here, because the authority of the code is associated with the owners in the policies, and each principal may only weaken his/her own policy. There are two cases:
   (a) *Wrong Declassification with sufficient authority*: If a principal has the authority to weaken a policy, such a wrong declassification is not safe. For example, if root wants to declassify his own password to the public, this is a serious mistake and the typechecker will not detect it.

(b) *Wrong Declassification without sufficient authority*: If a principal does not have the authority to weaken a policy, it is safe because such a mistake can be detected by the typechecker. For example, if an applet wants to declassify root's password, such a program will be rejected because the applet does not have root's authority.

This comparison gives a clear view of the motivation to use the DLM. In the reader model, it is not safe to allow declassification when the code is not trusted. In the DLM, declassification is safe as long as each principal does not compromise his own confidentiality.

6.2 Code must be trusted to assure integrity

We know that the decentralized label model and information-flow control can be used to assure data confidentiality, even when part of the code is not trusted. As long as the program is typechecked, it is guaranteed that no secret information can be leaked.

However, when untrusted code exists, the information-flow control approach is not immediately sufficient to assure integrity. Consider the following examples:

```
Example 1
1: int !{untainted} add(
2:    int !{untainted} a,
3:    int !{untainted} b )
4: {
5:    return a+b;
6: }
```

```
Example 2
1: class Evil {
2:  int !{untainted} fileHandle;
3:  void setFileHandle(int !{untainted} fh){
4:       fileHandle = fh - fh; // Evil here
5:  }
6:  int !{untainted} getFileHandle() {
7:       return fileHandle;
8: }}
```

In the first example, the method `add` does not violate any information-flow control constraints. It takes two `untainted` arguments and calculate the sum of them, so that the result can also be annotated as `untainted`. But if the code is downloaded from untrusted sources, the integrity of the result still cannot be guaranteed, because it is easy to compromise data integrity without violating information-flow constraints. For example, the `a+b` can be modified to `a-b`, or even `a-a`, so that the result can always be garbage. More generally, we can infer that, if the code is not trusted, then we have no integrity guarantee on any information returned from it, because the adversary can manipulate the data in many ways without violating information-flow control constraints. In the second example. The `Evil` class was written by the adversary. It was supposed to store the value of a file handle and later return it. Apparently, we want to assure the integrity of that file handle, but the direct information-flow control approach cannot provide such a guarantee. The `Evil` class can be typechecked, but the value returned from the `getFileHandle()` method is completely garbage.

Therefore, the information-flow control approach must be augmented to assure the integrity of information returned from untrusted code. If such untrusted

code exists, it may taint any data flowing out of it. One solution is to annotate these data as tainted by the authority of the code, which is equivalent to treating the untrusted code as tainted input channels.

6.3 Comparison of integrity label models

Now we compare the three integrity label models described earlier in this paper: the *writer model*, the *trust/distrust model*, and the *decentralized label model*. We exclude the untrusted code cases from our comparison, because none of these models can guarantee the integrity of data coming from untrusted code. We assume that all the code is written by a trusted person.

We compare the two models in the following failure modes (see the summary for integrity labels in the right half of Figure 1):

1. *Wrong Computation*: This is the case where the code performs some wrong computation. Clearly, such a mistake will break the integrity of the results, and it is not safe for any model.
2. *Violating Flow Constraints*: Just as for confidentiality, all models are safe.
3. *Wrong Endorsement*: This is the case where the code endorses some data that is actually tainted.

   The writer model is not safe in this case, because there is no relationship between the authority of the code and the principals in the label. If the programmer misuses a endorsement, the typechecker cannot find this mistake.

   In both the trust/distrust model and the DLM, the authority of the code is associated with the principals in the policies, and each principal may only weaken his/her own policy. There are two cases:

   (a) *Wrong Endorsement with sufficient authority*: If a principal has the authority to weaken a policy, such a wrong endorsement is still not safe. For example, if root wants to endorse an arbitrary integer and use it as the index to access an array, the typechecker will not detect it.

   (b) *Wrong Endorsement without required authority*: If a principal does not have the authority to weaken a policy, both the DLM and the trust/distrust models are safe. Such mistakes can be detected by typechecking.

This comparison shows that the writer model is not as powerful as the DLM, because the authority of the code is not associated with the principals in the label. However, the comparison does not show any difference between the trust model and the DLM. In fact, it seems that the trust model is sufficient to use in practical applications.

We have shown that although the DLM can be defined, manipulated and implemented in symmetric ways for integrity as well as for confidentiality, it does not provide more benefits than simpler models as the trust model or distrust model. The motivation and the power of DLM is significantly weakened by the assumption that the code is trusted.

7 Integrity and data invariants

This section considers data invariants as a means of strengthening information-flow based definitions of integrity.

7.1 Limitations of noninterference

Noninterference is not strong enough to represent the integrity policies needed in practice. In our case study of the integrity extension to the Jif language, we simply ruled out the information-flow analysis for untrusted code. We have seen such examples: An untrusted applet may have a method f that takes an untainted argument a and returns an untainted value $f(a)$. This method can be noninterfering, i.e. the output $f(a)$ only depends on a, the untainted input value. But in fact, even if f is noninterfering, $f(a)$ can be anything. It can be $a + a$ or any arbitrary constant. This will lead to serious problems that hurt the integrity of the system. What will happen if we use $f(a)$ as the index to access an array or use it as the format string for `printf()`?

The problem is that noninterference can be used only to control how the data flows in the system, but it cannot be used to control how the data is manipulated. For confidentiality, this suffices, because destroying the data only make the secrets safer, just like shredding a piece of paper with secrets on it only makes it harder to reveal these secrets. For integrity, we must consider both the data flow and its contents.

Consider the Unix file system as an example. Besides noninterference policies, which state that only authorized users can write to the file, we also want to make sure that the file system is manipulated in correct ways. The data structure of the file system should be kept consistent, i.e. the user cannot create cross-linked files or directories, the date and time of each directory item are valid (not June 32nd). Such requirements are far beyond the scope of noninterference policies, yet they are usually considered as integrity requirements.

7.2 Integrity as invariants

For a better understanding of integrity policies beyond noninterference, we need to study the concept of integrity of the contents of data. In practice, there are many examples: array access (indexes not exceeding the boundaries), message authenticity (digital signature or MAC is valid), and file systems (the data structures are consistent).

Depending on the context of the application, the integrity of data may have very different meanings. Nevertheless, they all require some property of the data, which is either "good", i.e. it satisfies the constraint or meets the specification, or "bad", i.e. the data is invalid or inconsistent. To be enforceable, such properties must be computable, so that given a specific value, we can decide whether it is good or bad. Generally, for a piece of data a, we can define the *quality* of a as a computable predicate φ on it:

$$\varphi(a) \equiv a \text{ has good quality}$$

An integrity policy on the value of data can be defined as: *an invariant φ on the quality of data under program execution.* We use the function f to represent a fragment of program such as a method or an instruction. For an integrity policy with invariant φ, we can define the quality of the program as a predicate ψ_φ on f, by checking whether it always maintains the invariant:

$$\psi_\varphi(f) \iff \forall a, \varphi(a) \to \varphi(f(a))$$

We would expect that such integrity policies be enforced by checking $\psi_\varphi(f)$. For example, if the quality of data φ is easy to compute, a simple approach to enforcing the invariant policies would be to rewrite the program f in the form

$$g(a) = \begin{cases} f(a) & \text{when } \varphi(f(a)) \\ b & \text{when } \neg\varphi(f(a)) \wedge \varphi(b) \end{cases}$$

where b is a value with good quality that satisfies φ. Suppose the invariant is a range [1..10] on variable x, then, after the execution of f, x may be 11, which is out of the range. In such a case, g could either return a default value 1 that falls within the range, or raise an exception, indicating the failure. This is exactly the approach people have used in software-based fault isolation [34], where mandatory runtime monitors and assertions are inserted into the programs. In particular, if $\varphi(a)$ holds and we choose $b = a$, the function g has the atomic effect of a transaction—the data a will either persist unchanged or become $f(a)$ if and only if $f(a)$ has good quality.

If we treat the data during program execution as traces, a security policy based on the invariant φ is a safety property [29], specifying that bad things never happen during any execution. Therefore, various enforcement mechanisms for safety properties can be used to assure integrity, such as execution monitoring with security automata [29], software-based fault isolation [34], and proof carrying code [24].

7.3 Enforcing the invariants

Clark and Wilson's model [6] used an access-control style enforcement mechanism for preserving data invariants. The model introduced the concepts of CDI (Constrained Data Item), referring to data with good quality, and TP (Transformation Procedure), referring to programs preserving the invariants on CDI. The invariants are enforced with two kinds of rules: certification rules, which state that all the TPs are certified so that they always preserve the invariants on the CDIs, and enforcement rules, saying the CDI can only be accessed via the TP and the accesses are controlled according to a certified relation (User, TP, CDI).

The certification of TP is exactly the problem of checking whether a program preserves a data invariant. In Clark and Wilson's model, the certification is usually a manual operation performed by the security officer or the system custodian. For large systems, solely relying on the manual proofs would be inefficient and error-prone. Moreover, the emerging application of mobile code brings more challenges on the integrity model: it is impractical to manually check the

safety of a downloaded applet before executing it. Therefore, we need automated mechanisms to enforce the data invariants for programs.

There are two kinds of enforcement mechanisms—static and dynamic mechanisms. One tradeoff between them is that they have different information to work with—dynamic mechanisms have access to the run-time state, but static mechanisms typically have access to the full program text. Another tradeoff is that dynamic mechanisms potentially introduce more runtime overhead. The typical dynamic runtime mechanisms include software-based fault isolations [34, 32] and reference monitors [29, 18].

Static mechanisms, such as type systems and program analyses are also a promising way to specify integrity invariants. The difficulty is in verifying rich integrity invariants—properties involving arithmetic, for instance, quickly become intractable.

An important research direction is to apply these technologies to integrity invariants, potentially by combining static and dynamic approaches: type systems that ensure dynamic checks have been inserted into the system appropriately.

8 Availability vs. integrity

Recently, the Denial-of-Service (DoS) attack, emerging as one of the most serious security problems, has brought availability issues to the security community. Because the idea behind DoS attack is to make the computing resources *unavailable* to break down the system, not confidentiality, integrity or anonymity, but availability policies are violated. However, there are some similarities between integrity and availability policies.

The well-known availability definition is: Availability $= \frac{\text{MTTF}}{\text{MTTF+MTTR}}$.

MTTF stands for Mean Time to Failure and MTTR stands for Mean Time To Repair. One problem of this definition is that it tries to measure the failure of the whole system. In a large, distributed system, even when some hardware or software fails, parts of the system are still functioning. Thus, we need a finer grained definition. We propose that the system may have multiple inputs and outputs, each of which has a different availability concern. This requirement is almost inevitable for the large-scaled distributed system. Now, it is natural to make the analogy of the noninterference notion from integrity to availability. That is, just as we need that untainted data is not affected by tainted data for integrity, we need that highly available data (dependable data) does not depend on the low-available data (undependable data) during the computation.

The formal definition of noninterference for availability is as follows:

- Suppose there are two security levels of data: *undependable* and *dependable*. Let the program f take the input ($i_{undependable}$, $i_{dependable}$), and produce the output ($o_{undependable}$, $o_{dependable}$).
- Define *noninterference* as a property of the program: f is noninterfering iff

$$\forall a, a' : \ f(a, b) = (c, d) \ \Rightarrow \ f(a', b) = (c', d)$$

That is, the value of $i_{undependable}$ will not affect the value of $o_{dependable}$.

Given this formal definition, we also need to formalize the model of the multiple inputs/outputs system, and precisely answer what we mean by availability for an output. First, we simplify the system as a real-time function f: f takes a vector $\langle i_1, i_2, \ldots, i_n \rangle$ as input, and outputs another vector $\langle o_1, o_2, \ldots, o_m \rangle$. Assume that for all $\langle i_1, i_2, \ldots, i_n \rangle \in I^n$, RunningTime($f(\langle i_1, \ldots, i_n \rangle)$) $\leq t_0$. That is, the running time of f, no matter what its inputs are, is bounded by time t_0. All the data the system depends on should be specified by the input parameters or have universal availability. If the input channel for i_j is available, the data can be read by the system in time t_j, which should be negligible compared to t_0, i.e. $t_j = o(t_0)$.

Secondly, in this model, we define the availability as a property on each output element of the system. The availability for an output is the a vector of probabilities, each element of which is the probability for the system to get the corresponding output element successfully in time $O(t_0)$. By this definition, in a system one can have different levels of availability for different output elements. Recent research shows that such multi-level tunable availability is useful in system design [37].

The noninterference of this model is useful at least in following two scenarios:

1. There is a well-known trade-off that making data highly available sacrifices performance. For example, in storage systems, making more replicas increases availability, but maintaining consistency degrades the performance. Therefore, noninterference is useful to guarantee that all critical data is dependable while other data may not be, permitting improved performance.
2. One possible way to fight against DoS attack is to allocate some privileged users separate, highly available resources from ordinary users [5]. When some DoS attack happens, the behavior of privileged users is not affected.

There are limitations to this noninterference definition of availability.

1. The system model above is significantly simplified from reality. System running time is typically not bounded and varies according to many factors, such as logic inside the program, input, network conditions, etc. It is unpredictable in general. Strong noninterference, which takes timing channels into account, may have application to availability research, i.e., the undependable data cannot interfere with the timing effects of dependable data.
2. The computing infrastructure, including OS, CPUs, chips, storage devices, power supplies, is heavily influential in availability issues. For confidentiality and integrity, it is natural to put the infrastructure into the trust computing base. For availability, however, how dependable the infrastructure is becomes much more important [25, 16]. One interesting question is how to practically specify the availability of underlying infrastructure.
3. A probabilistic model of availability brings difficulties to both the program analyzer and end-users who specify policies. For program analyzer, it is harder to calculate least upper bounds of security levels because they may or may not correspond to independent events. For end users, it is harder to specify availability of resources in terms of percentages than it is to specify who can see/modify the data in confidentiality/integrity.

Therefore, enforcing noninterference for availability is sometimes necessary for some applications, but definitely not sufficient.

These observations suggest a number of possible future research directions:

- Specifying availability concerns inside programs to enable static program analysis, perhaps using type systems to provide the specifications. We call for concrete specification models to address this problem. In particular, how to specify the underlying hardware availability is an important open question.
- Developing a better metric model for availability than the model defined above. We need to incorporate timing effects, the probability that the service is available, the corresponding distribution, and correlated events.
- Exploring ways to better enforce timing sensitive strong noninterference.

9 Conclusion

We examined similarities and differences between access-control and information-flow based definitions of confidentiality, integrity, and availability policies, focusing mainly on integrity.

Confidentiality, integrity, and availability can be defined as *noninterference* policies in *information-flow control* with varying degrees of success. The *decentralized label model* exhibits the duality between confidentiality and integrity. However, treating integrity as the formal dual of confidentiality leads to a weak notion of security, because integrity demands additional assurance on the quality of the data. This paper suggests *invariants* on quality of the data under program execution as a way to strengthen noninterference based integrity policies.

These observations lead us to propose a definition of integrity as *program correctness, noninterference*, and *data invariant* conditions, which yields the following categorization:

- *Program correctness*: program output is precise, accurate, meaningful and correct with respect to a specification.
- *Noninterference*: data is modified only by authorized people or processes either directly, or indirectly by means of information flow.
- *Data invariants*: data is precise or accurate, consistent, unmodified, or modified only in acceptable ways under program execution.

References

1. D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, MITRE Corp. MTR-2997, Bedford, MA, 1975. Available as NTIS AD-A023 588.
2. K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977.
3. M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.
4. R.S. Boyer and J.S. Moore. *The Correctness Problem in Computer Science*. Academic Press, London, 1981.

5. Jos Brustoloni. Protecting electronic commerce from distributed denial-of-service attacks. In *Proceedings of the eleventh international conference on World Wide Web*, pages 553–561. ACM Press, 2002.
6. David Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–194, 1987.
7. Dorothy E. Denning and Peter J. Denning. Certification of Programs for Secure Information Flow. *Comm. of the ACM*, 20(7):504–513, July 1977.
8. J. B. Dennis and E. C. VanHorn. Programming semantics for multiprogrammed computations. *Comm. of the ACM*, 9(3):143–155, March 1966.
9. Simon N. Foley. The specification and implementation of 'commercial' security requirements including dynamic segregation of duties. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 125–134. ACM Press, 1997.
10. Simon N. Foley. A nonfunctional approach to system integrity. *IEEE Journal on Selected Areas in Communications*, 21(1):36–43, January 2003.
11. Cedric Fournet and Andrew Gordon. Stack inspection: Theory and variants. In *Proc. 29th ACM Symp. on Principles of Programming Languages (POPL)*, pages 307–318, 2002.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, April 1982.
13. M. A. Harrison, W. L Ruzzo, and J. D. Ullman. Protection in operating systems. *Comm. of the ACM*, 19(8):461–471, August 1976.
14. Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Proc. 25th ACM Symp. on Principles of Programming Languages (POPL)*, pages 365–377, San Diego, California, January 1998.
15. C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. of the ACM*, 12(10):576–580, October 1969.
16. James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.
17. Butler W. Lampson. Protection. In *Proc. Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, Princeton University, March 1971. Reprinted in Operating Systems Review, 8(1), January 1974, pp. 18–24.
18. I. Lee, H. Ben-Abdallah, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Monitoring and checking framework for run-time correctness assurance. In *Proceedings of the Korea-U.S. Technical Conference on Strategic Technologies*, 1998.
19. Theodore M. P. Lee. Using mandatory integrity to enforce "commercial" security. In *IEEE Symposium on Security and Privacy*, pages 140–146. IEEE Computer Society Press, 1988.
20. John McLean. Security models and information flow. In *Proc. IEEE Symposium on Security and Privacy*, pages 180–187. IEEE Computer Society Press, 1990.
21. Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. 26th ACM Symp. on Principles of Programming Languages (POPL)*, pages 228–241, San Antonio, TX, January 1999.
22. Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, 2000.
23. Andrew C. Myers, Nathaniel Nystrom, Lantian Zheng, and Steve Zdancewic. Jif: Java information flow. Software release. Located at http://www.cs.cornell.edu/jif, July 2001.

24. George C. Necula. Proof-carrying code. In *Proc. 24th ACM Symp. on Principles of Programming Languages (POPL)*, pages 106–119, January 1997.
25. David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conf. on Management of Data*, pages 109–116. ACM Press, 1988.
26. Charles P. Pfleeger. *Security in Computing*, pages 5–6. Prentice-Hall, 1997. Second Edition.
27. François Pottier and Sylvain Conchon. Information flow inference for free. In *Proc. 5th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 46–57, September 2000.
28. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.
29. Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 2001. Also available as TR 99-1759, Computer Science Department, Cornell University, Ithaca, New York.
30. Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, and David Wagner. Detecting format string vulnerabilities with type qualifiers. In *Proceedings of the 10th USENIX Security Symposium*, 2001.
31. W. Shockley. Implementing the clark/wilson integrity policy using current technology. In *Proceedings of NIST-NCSC National Computer Security Conference*, pages 29–37, 1998.
32. Chris Small. MiSFIT: A tool for constructing safe extensible c++ systems. In *Proceedings of the Third Usenix Conference on Object-Oriented Technologies*, June 1997.
33. Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
34. R. Wahbe, S. Lucco, T. Anderson, and S. Graham. Efficient software-based fault isolation. In *Proc. 14th ACM Symp. on Operating System Principles (SOSP)*, pages 203–216. ACM Press, December 1993.
35. Dan S. Wallach, Andrew W. Appel, and Edward W. Felten. The security architecture formerly known as stack inspection: A security mechanism for language-based systems. *ACM Transactions on Software Engineering and Methodology*, 9(4), October 2000.
36. W. A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: The kernel of a multiprocessor system. *Comm. of the ACM*, 17(6):337–345, June 1974.
37. Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services. In *Proc. 18th ACM Symp. on Operating System Principles (SOSP)*, Banff, Canada, October 2001.
38. Steve Zdancewic. A type system for robust declassification. In *Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics*. Electronic Notes in Theoretical Computer Science, March 2003.
39. Steve Zdancewic and Andrew C. Myers. Robust declassification. In *Proc. of 14th IEEE Computer Security Foundations Workshop*, pages 15–23, Cape Breton, Canada, June 2001.
40. Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *Transactions on Computer Systems*, 20(3):283–328, 2002.
41. Lantian Zheng, Stephen Chong, Steve Zdancewic, and Andrew C. Myers. Building secure distributed systems using replication and partitioning. In *IEEE 2003 Symposium on Security and Privacy*. ieee, 2003.

I'm Not Signing That!

James Heather[1] and Daniel Hill[2]

[1] Department of Computing, University of Surrey, Guildford, Surrey, GU2 7XH, UK.
Email: j.heather@eim.surrey.ac.uk
[2] Department of Philosophy, University of Liverpool, 7 Abercromby Square,
Liverpool, L69 3BX, UK. Email: djhill@liverpool.ac.uk

Abstract. In which we demonstrate that, under eminently reasonable assumptions, Ben-Or, Goldreich, Micali and Rivest's contract-signing protocol has a curious weakness: rational agents—that is, agents who are unwilling to concede any advantage if there is no chance at all of recouping the loss in the long run—will never be willing to run the protocol at all.

1 Introduction

Often it is desirable to have a procedure to allow two parties to sign a contract simultaneously. The obvious danger of having one party sign before the other is that the second party may refuse to sign until it becomes to his advantage to do so. For instance, suppose the contract is for Alice to buy 1,000 shares in Fudge Labs, Inc., from Bob, at £1 per share, in a year's time. If Alice signs first, Bob may then delay signing until the end of the year. If the price has fallen, he will sign, the contract will be binding on both parties, and he will make a profit; if the price has risen, he will not sign. A similar problem will arise if Bob signs first. Because the market changes over time, anything that allows one party the option of delaying the decision on whether to commit will cause problems.

Simultaneity in message transmission is, in most practical situations, impossible to achieve, making cryptographic contract-signing protocols difficult to realize. A trusted third party can be employed to receive the two signatures and then distribute them, but it is clearly advantageous to construct a protocol that does not require action on the part of a third party unless there is suspicion of foul play by one of the parties involved in the contract.

In this paper, we examine a cryptographic contract-signing protocol that aims to solve these problems by making the probability that one party can abuse the protocol arbitrarily small. We demonstrate that the protocol has a curious weakness: two rational agents whose rationality is common knowledge will refuse to run the protocol at all.

Most protocol properties that have been studied and analysed in the literature to date—secrecy and authentication in particular—are ones for which the two protocol participants are considered to have the same goal: secrecy of some value from other agents, or authentication of the initiator to the responder, for instance. In such cases, the agents are either both satisfied or both disappointed.

A contract-signing protocol, however, does not fit this mould: the very existence of such a protocol is predicated on the fact that Alice would be better off if Bob were to commit while she did not.

These considerations suggest that the protocol should be analysed within the general framework of *game theory*, which provides, among other things, ammunition for reasoning about interaction between two parties who may assign different levels of satisfaction (different *pay-offs*) to the same outcome. It is this framework that we use here.

This paper is not the first to apply such techniques to security protocols. Kremer and Raskin used game theory to conduct formal analysis of a contract-signing protocol in [1], though their method is rather different from that employed here.

The paper is structured as follows: Sect. 2 describes the protocol in question; in Sect. 3 we set out and demonstrate the problem; we then in Sect. 4 consider whether our argument is related to two famous game-theoretic problems; and finally we sum up in Sect. 5.

2 The Protocol

Ben-Or, Goldreich, Micali and Rivest in [2] propose a protocol for approximating simultaneous contract signing on a network in which simultaneous message transmission is impossible. Interest in this protocol has been recently revived by Norman and Shmatikov's analysis in [3].

The protocol operates by means of allowing agents A and B to commit to the contract with a certain probability. As the messages go back and forth between the agents, the probability of each being committed to the contract increases gradually from 0 to 1. The protocol takes two parameters: a measure of fairness $\epsilon > 0$, and a measure of negligibility $v > 0$.

If either agent should abort the protocol, the probability that A will not be committed given than B is committed will be less than ϵ, provided that the probability that B is committed is at most v; and similarly for A committed and B not. These parameters ϵ and v can be made as close to zero as one wishes; however, the smaller the values, the more messages will need to be exchanged to complete the protocol and have both parties committed to the contract.

The protocol makes use of a *judge*—but only if the protocol run does not reach completion. If everything goes according to plan, the protocol run terminates with each party having committed to the contract with probability 1.

If either party fails to send the next message of the run, the other party can send the last message received to the judge. The judge, on receiving a message with the meaning

$$A \text{ commits to contract } C \text{ between } A \text{ and } B \text{ with probability } p$$

will choose a value for a random variable x with a uniform distribution on $[0, 1]$. If $x \leqslant p$ then the judge will declare C binding on A; otherwise, the judge will

declare it not binding. If another message is later received by the judge for the same contract C, he uses the *same* value of x as before.

The purpose of the judge in this protocol is rather unlike that of the trusted third party in most other protocols. The judge here is not performing any cryptographic service, but simply making a ruling on whether the contract is binding or not.

2.1 Protocol Messages

A general protocol message k will be of the form

$$\text{Message } k. \quad a \to b \ : \ \{a, b, c, p\}_{SK(a)}$$

meaning that a commits to contract c between a and b with probability p. The precise mechanism used for signing the messages is not important for the purposes of this paper.

The agents decide, by some undisclosed mechanism, who is to 'go first'; we shall take it that a goes first. They also decide on a factor $\alpha > 1$ by which to increase the probability of commitment at each turn[3], such that the probability of an agent not being committed given that the other agent is committed will never exceed ϵ (except when the probability of commitment is at most v). The first message sent is

$$\text{Message 1.} \quad a \to b \ : \ \{a, b, c, v\}_{SK(a)}$$

Thereafter, message sending alternates between a and b. When b is to send a message, he takes the probability p committed to by a in the previous message m, and sends out

$$\text{Message } (m+1). \quad b \to a \ : \ \{a, b, c, \min(\alpha \cdot p, 1)\}_{SK(b)}$$

Agent a, on receiving a commitment by b with probability p' in message m', sends out

$$\text{Message } (m'+1). \quad a \to b \ : \ \{a, b, c, \min(\alpha \cdot p', 1)\}_{SK(a)}$$

The protocol continues, with the probability of commitment by the agent sending the message increasing by a factor of α at each stage (except at the very end). The protocol terminates when each agent has committed to the protocol with probability 1.

[3] The original paper allows for the agents to choose different factors—or, in fact, for either agent or both agents to get away with never increasing the probability at all! This introduces the possibility of infinite 'ping-pong' between the two agents, with no progress ever made.

Provided that, as in any real-world implementation, the probability space is discrete, so that there is an upper bound on the length of an eventually terminating protocol run, these considerations do not affect the results presented here.

3 Problems

We now set out the problem with the protocol. Section 3.1 gives the assumptions required for our analysis; then in Sect. 3.2 we consider what will happen when two agents look to sign a contract using the protocol.

3.1 Assumptions

In what follows, we make two important assumptions about the two agents A and B who wish to sign the contract.

Assumption 1. The agents are *rational*; that is, they always follow strategies that are optimal based on the knowledge that they have at the time. This results in a certain type of selfishness: an agent will never give anything away for nothing. If it is certain that he will not gain (in the long run) from sending a message, he will not send it.

Assumption 2. There is *common knowledge* of rationality between the agents; that is,

(i). A knows that B is rational.
(ii). B knows that A is rational.
(iii). A knows that B knows that A is rational.
(iv). B knows that A knows that B is rational.
(v). A knows that B knows that A knows that B is rational.
(vi). B knows that A knows that B knows that A is rational.
etc.

These assumptions are not in any way unusual: they are standard fare in game theory (see, for example, [4, 5]).

3.2 Stalemate

Consider what happens when Alice and Bob attempt to run the protocol to sign contract C, using a negligibility of v and a factor of α. Let us assume, without loss of generality, that Bob is to send the last message of the run, and that this will be Message n. We shall further assume that $v \cdot \alpha^d = 1$ for some natural number d. This last assumption is not necessary, but it makes the notation easier, and does not otherwise affect the argument.

The last few messages of the run will, therefore, if all goes according to plan, be

$$\begin{array}{lrcl}
\text{Message } (n-3). & \mathit{Alice} \to \mathit{Bob} & : & \{\mathit{Alice}, \mathit{Bob}, C, \alpha^{-2}\}_{SK(\mathit{Alice})} \\
\text{Message } (n-2). & \mathit{Bob} \to \mathit{Alice} & : & \{\mathit{Alice}, \mathit{Bob}, C, \alpha^{-1}\}_{SK(\mathit{Bob})} \\
\text{Message } (n-1). & \mathit{Alice} \to \mathit{Bob} & : & \{\mathit{Alice}, \mathit{Bob}, C, 1\}_{SK(\mathit{Alice})} \\
\text{Message } n. & \mathit{Bob} \to \mathit{Alice} & : & \{\mathit{Alice}, \mathit{Bob}, C, 1\}_{SK(\mathit{Bob})}
\end{array}$$

Suppose that Bob has just received Message $(n-1)$, and that he is therefore due to send out Message n. What is his motivation for doing so? It is immediately clear that he has none whatsoever: Alice has already committed with probability 1, and Bob knows that his final message cannot elicit any further commitment from Alice. To send out Message n would lose him his small probability of $1-\alpha^{-1}$ of getting out of the contract at a later date, and for no gain. If Bob is rational (in the sense of Assumption 1) then he will certainly not send out Message n.

Now consider Alice's position when she has received Message $(n-2)$ and is preparing to send Message $(n-1)$. She knows, by Assumption 2, that Bob is rational, and so she knows that if she sends Message $(n-1)$ then she will not get a response. This puts her in a position similar to Bob's discussed above: she is being asked to give something for nothing. She knows that, whatever she does, she will not get any further commitment from Bob; so she has nothing to gain by continuing with the protocol. By sending Message $(n-1)$, she would increase the probability of her being bound to the contract from α^{-2} to 1, for no profit. She will not comply.

But then, of course, Bob has no reason to send out Message $(n-2)$. Why should he? To do so would commit himself further—with probability α^{-1}, rather than the α^{-3} of Message $(n-4)$—with no hope of a response. His rationality, his knowledge of Alice's rationality, and his knowledge of Alice's knowledge of his rationality, forbid it.

The rather startling conclusion now becomes clear: Alice will not start the protocol at all! The argument above can be tracked right back to the beginning of the protocol. If Message $(k+1)$ will not be sent, then Message k will not be sent either. And since Message n will not be sent, we see by induction that Message 1 will not be sent.

In summary: if the agents are rational, and have common knowledge of rationality, then they will be unwilling to run the contract-signing protocol.

4 The Validity of the Argument

We now turn to consideration of whether the problems that we have identified in the protocol are similar to those exemplified in a couple of well-known puzzles that are much discussed in the literature: the 'paradox' of the unexpected hanging, and the problem of the prisoner's dilemma. We argue in Sect. 4.1, drawing on Quine's work in [6, 7], that the 'paradox' of the unexpected hanging is no paradox at all, but in any case is related to the weakness in the protocol only in virtue of being another instance of weak backwards mathematical induction. We then turn in Sect. 4.2 to the prisoner's dilemma, where we accept the surprising result that two rational agents may each be worse off than two irrational agents. We argue that this is replicated in the protocol: two rational agents, in not running the protocol, and, consequently, not signing the contract, can each be worse off than two irrational agents that run it and, consequently, sign the contract.

4.1 The 'Paradox' of the Unexpected Hanging

At first sight, it looks as if the contract-signing protocol suffers from a weakness related to that suffered by the well-known 'paradox' of the unexpected hanging.

The story runs as follows. A judge tells a prisoner that he will be hanged one afternoon in the next week. Further, he tells him that, to make his punishment worse, he will not know on the morning of the day of the hanging that that is indeed the day of the hanging. The prisoner is not subdued by this dread announcement, but, on the contrary, is jubilant. He reasons thus:

> Suppose I am to be hanged on Friday. Then I shall know on Friday morning that I am to be hanged on Friday, since I shall obviously know that I have not been hanged on preceding days. So clearly I cannot be hanged on Friday as that would violate the judge's decree. But since I now know that I cannot be hanged on Friday, I cannot be hanged on Thursday either. Suppose I were to be hanged on Thursday; then I'd know on Thursday morning that I was to be hanged on Thursday, since I should know first that I have not been hanged on previous days, and, secondly, by my previous argument, that I cannot be hanged on Friday. Hence I cannot be hanged on Thursday.

By a similar argument the prisoner rules out every day of the week and concludes that he cannot be hanged.

The story comes in many different versions, the surprise examination being the most common. It was first discussed in print in [8], though, Gardner tells us, it apparently circulated by word of mouth in the early 1940s (see, *inter alia*, Chapter 1 of [9]). Gardner thinks that the origin of the 'paradox' lies in a Swedish Broadcasting Company announcement in 1943 or 1944 that a civil-defence exercise would be held in the following week and that no one would know before the actual day on which day of the week it would take place. For further references we advise the reader to consult [10].

What is wrong with the prisoner's reasoning? We are convinced by the simple argument of Quine in [6, 7]. The prisoner errs right at the start—when considering whether he could be hanged on Friday he considers only two possibilities:

(i). that he would be hanged on Friday afternoon and would know it then;
(ii). that he would have been hanged before Friday afternoon.

The prisoner rejected (i) because of its non-fulfilment of the judge's decree, leaving (ii) as the remaining possibility, and then reasoned inductively to the absurd conclusion that he would have had to be hanged before the start of the week. He did not consider at the start of his reasoning the further possibilities:

(iii). that he would not be hanged at all;
(iv). that he would be hanged on Friday afternoon but would not know it then.

It is clear that (iii) contravenes the decree, but it is neither more nor less than the prisoner's own conclusion at the end of all of his reasoning, and (iv) is another

live option. Indeed, it is perfectly possible that (iv) will actually be realized, and that the unfortunate prisoner, still confidently believing (iii), will be surprised by the arrival of the hangman on Friday afternoon. Since the prisoner is first surprised and then hanged, the decree is fulfilled.

The question arises as to whether the weakness of the protocol is related to the weakness, as outlined above, of the unexpected hanging 'paradox'. At first sight, the prisoner's reasoning is very similar to our reasoning about the protocol in Sect. 3.2. In each case, the last possibility (hanging on the last day, or sending of the final message) is ruled out; the penultimate is consequently discarded too; and the argument proceeds inductively to rule out every possible day on which to be hanged or every possible message to be sent. If the prisoner's reasoning is fallacious, one must ask whether our argument concerning the protocol is beset with similar problems.

A little thought convinces one that it is not, however. The essence of the prisoner's fallacy, as Quine makes clear, is to confuse two fundamentally different thoughts: the assumption that the judge's decree will be fulfilled, and the assumption that he *knows* that the judge's decree will be fulfilled. It is as a result of this that the prisoner fails to distinguish between (i) and (iv) in his reasoning. This fallacy does not make an appearance, however, in our treatment of the protocol. The key to this is the assumption of common knowledge of rationality (Assumption 2). If we were to drop this, but keep Assumption 1, the reasoning would no longer be sound: we could still infer that Bob would not send the last message of the protocol, but we could go no further. Alice would no longer have a clear reason to hold back with the penultimate message, because she would not know that Bob was rational, and hence not know that Bob would not send the final message.

Without the common knowledge assumption, we would be making an invalid leap from Bob's rationality to Alice's knowledge of Bob's rationality; this is, in essence, the mistake that the prisoner makes in his reasoning. With the assumption in place, however, the argument is sound. Assumption 2 plugs the gap that the prisoner needs somehow to fill to get his argument to work.

4.2 The Prisoner's Dilemma

We have argued above that no rational player will participate in the protocol under discussion, and that, consequently, it will never get off the ground with rational players. We then have the surprising conclusion that two irrational players may be jointly better off than two rational players. This is because if they are both irrational then they may run the protocol to conclusion and thus get the contract signed, whereas if they are rational then they will not run the protocol and they will not sign the contract. Under the reasonable assumption that each is better off if both sign than if neither signs—for otherwise, they would not be looking to sign the contract at all—it follows that each may be better off if both are irrational than if both are rational. This surprising result is reminiscent of the well-known problem of the prisoner's dilemma.

The basic form of the prisoner's dilemma, according to Kuhn [11], originated with Merrill Flood and Melvin Dresher in the 1950s, though the title 'Prisoner's Dilemma' and the precise format we use below are originally due to Albert Tucker. For further information, we refer the interested reader to the exhaustive and annotated bibliography of Robert Axelrod [12].

In the prisoner's dilemma we are asked to imagine that two prisoners have been captured and are being held separately from each other, such that each has no means of communication with the other. Each is offered a deal by the police: if he confesses and implicates the other, he will get off scot-free, while the other will get a heavy sentence (say, ten years in prison); but if he stays silent and the other confesses, he will then get the ten-year stretch, while the other will get off scot-free. If both confess then each will get five years, but if both stay silent then they will get only one year apiece, since the courts will be able to convict only on a lesser charge. For each prisoner the dominant or optimal strategy is to confess: if the other confesses, he will do better to confess; and if the other does not confess, he will do better to confess. The surprising thing is that if each prisoner is rational then both will confess and the overall result will be worse for each of them than if both were to remain silent. It is clear that the prisoner's dilemma is closely related to the weakness in the protocol: rational agents can be worse off than irrational agents.

One may then ask: what if Alice and Bob had more than one contract to sign? What if they had a pile of k contracts to work through, one by one, with each signing each contract? One might wonder whether in this case it would be worthwhile for Alice to sign the first few contracts, so as to encourage Bob to sign some, and only refuse to sign near the bottom of the pile. She might be better off with, say, ten signed contracts and a risk of Bob refusing to co-operate on the eleventh, than no signed contracts at all.

This reasoning calls to mind the iterated prisoner's dilemma. Here there are k rounds, for some fixed (large) k, in each of which each prisoner is given the option to confess and the option to remain silent. In each round the prisoner knows what he and what the other prisoner did in previous rounds. The suggestion is that this 'solves' the problem, as a prisoner can be punished in later rounds for having confessed in earlier rounds.

However, this line of reasoning is not sound (see Selten's work in [13]). The kth round would function exactly as would a non-iterated prisoner's dilemma, and since each prisoner would realize this, he would not take it into account in his deliberations over how to play the $(k-1)$th round, which would then in turn be reduced to a standard non-iterated prisoner's dilemma. By induction, this would apply to every round, and so the whole problem would collapse into a normal, non-iterated prisoner's dilemma, in which each prisoner's dominant strategy is to confess at every turn, thus ensuring that each is worse off (under the assumption that both are rational) than had both stayed silent.

This, of course, applies directly to Alice and Bob's pile of contracts. They will not sign the kth contract, for the reasons outlined in Sect. 3.2; and, by Assumption 2, each will know in advance that this will be the case. They will,

therefore, have no motive to sign the $(k-1)$th contract, or the $(k-2)$th, and, by induction, will not sign any of the contracts at all. The fact that they have a large, fixed number of contracts to sign makes no difference at all to the practicality of the protocol.

Note, however, that the number of contracts must be known to the agents in advance in order for this argument to work. If there were an infinite number of contracts to sign, or a finite but unknown number, the situation would be less than clear. One might argue that issues of reputation and trust come into play: an agent might not want to get a reputation for being a non-signer. Two considerations make gaining a reputation less significant than one might think, however: in the first place, many contracts are of such importance and value (for instance, exchanging contracts for sale of a house) that concerns about reputation are likely to be overshadowed by concerns about what is to be gained from this particular contract; and in the second place, one would hope to be able to use such a protocol over an anonymous channel (where the signature is tied to an anonymous identifier for a particular electronic payment method, for instance, rather than to a named individual), and reputation makes no sense with anonymous signing. In any case, there is no indication from the designers of the protocol under consideration that a desire for a good reputation is necessary to get the protocol to function adequately.

5 Conclusions

In this paper, we have demonstrated that Ben-Or, Goldreich, Micali and Rivest's contract-signing protocol suffers from a serious and surprising weakness: two rational agents whose rationality is common knowledge will not be prepared to use the protocol at all.

Four important points should be noted about what we have concluded here.

(i). The result does not in any way depend on the value the agents may place on the contract itself. Even if each is desperate to have the other committed to the contract, neither will get anywhere. Once Alice is fully committed, Bob has everything he wants, and has no reason at all to send the last message, regardless of how pleased he may be to have Alice's commitment. And however much Alice may want Bob to send the last message and commit with probability 1, she knows that nothing will induce him to do so; and so, regardless of the value she places on the contract, she will not send out the penultimate message...

(ii). Nor does the result hinge on the agents' risk profiles. The argument we have given is not one that requires probabilistic analysis on the part of Alice and Bob, but one based on certainties: for Alice to send the penultimate message in the hope of a response from Bob would not be to take a risk but to guarantee herself a loss. She may be as risk-averse or as risk-inclined as she wishes; but she will still be unwilling to concede her advantage when she has no chance whatsoever of a response from Bob.

(iii). It should be noted that these results apply only in cases where each party's best result is to get the other party to commit to the contract while remaining uncommitted himself. There may be cases where both parties need to commit in order to make any subsequent progress; and, in this case, it will be to Bob's advantage to send the last message, and the argument will not hold. However, simultaneous contract-signing protocols are not designed with such situations in mind! It is precisely to deal with the possibility of one party signing and the other not signing that contract-signing protocols were invented.

(iv). This is not the end of the story. It would be interesting to investigate the effects of other more advanced game-theoretic models on this and other contract-signing protocols. Selten's 'trembling hand' model (see [13]), for example, allows for consideration of game participants who, on each turn, fail with some small probability to act on their rational decision. This would allow us to specify agents who might at each point continue the protocol in spite of their belief that they have nothing to gain. Incorporating this into the analysis will be the topic of a further paper.

We suspect that this problem is fundamentally insoluble without involving a more traditional form of trusted third party. However the protocol is organized, someone will have to send the last message; and since it is never possible, without the aid of a trusted third party, to prove that one has sent a message, it follows that the agent sending the last message of the protocol has nothing to gain by doing so. The same argument as that presented in Sect. 3 will then apply: the last message will not be sent; therefore, by the same reasoning, the penultimate message will not be sent; therefore the antepenultimate message will not be sent; and so on.

There are protocols in existence that make use of a trusted third party, in a way that is not vulnerable to the problem outlined above, to perform cryptographic functions in cases where one party or the other fails to complete the protocol. The work by Asokan, Shoup and Waidner [14], for example, makes use of *verifiable key escrow* to ensure that the third party can force the agents to play fair. The weight on the shoulders of the third party here is much less than if he were involved in every run. However, it would clearly be advantageous to have a protocol that does not ever involve a third party; our belief is that this is impossible without running into the sort of problem laid out in this paper. Even in the Ben-Or, Goldreich, Micali and Rivest protocol, in which the third party is active but plays a purely judiciary role, rational agents are unable to make any progress.

Acknowledgements

We wish to thank Peter Ryan and the anonymous referees for their helpful comments on this paper.

References

1. Kremer, S., Raskin, J.F.: Game Analysis of Abuse-free Contract Signing. Proceedings of 15th IEEE Computer Security Foundations Workshop (2002)
2. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A Fair Protocol for Signing Contracts. IEEE Transactions on Information Theory **36** (1990) 40–46
3. Norman, G., Shmatikov, V.: Analysis of Probabilistic Contract Signing. In Abdallah, A., Ryan, P., Schneider, S.A., eds.: Formal Aspects of Security, Royal Holloway, University of London (2002)
4. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press (1995)
5. Syverson, P.F.: Logic, Convention, and Common Knowledge: A Conventionalist Account of Logic. CSLI Publications (2003)
6. Quine, W.v.O.: On a so-called paradox. Mind **62** (1953) 65–67
7. Quine, W.v.O.: 2. In: The Ways of Paradox, and Other Essays. Harvard University Press, Cambridge, Mass. (1976)
8. O'Connor, D.J.: Pragmatic Paradoxes. Mind **57** (1948) 358–359
9. Gardner, M.: The Unexpected Hanging and Other Mathematical Diversions: With a New Afterword and Expanded Bibliography. University of Chicago Press, Chicago (1991)
10. Chow, T.Y.: The Surprise Examination or Unexpected Hanging Paradox. American Mathematical Monthly **105** (1998) 41–51 Available with supplementary bibliography at `http://xxx.lanl.gov`.
11. Kuhn, S.T.: Prisoner's dilemma. Entry in the Stanford Encyclopedia of Philosophy (2003) Available at `http://plato.stanford.edu/entries/prisoner-dilemma`.
12. Axelrod, R.: Annotated Bibliography on the Evolution of Cooperation (1994) Available online at `http://pscs.physics.lsa.umich.edu/RESEARCH/Evol_of_Coop_Bibliography.html`.
13. Selten, R.: The Chain-Store paradox. Theory and Decision **9** (1978) 127–159
14. Asokan, N., Shoup, V., Waidner, M.: Optimistic Fair Exchange of Digital Signatures. IEEE Journal on Selected Areas in Communications **18** (2000) 593–610

The Attacker in Ubiquitous Computing Environments: Formalising the Threat Model*

Sadie Creese[1], Michael Goldsmith[3,4], Bill Roscoe[2,3], and Irfan Zakiuddin[1]

[1] QinetiQ Trusted Information Management, Malvern, UK.
{S.Creese,I.Zakiuddin}@eris.QinetiQ.com
[2] Oxford University Computing Laboratory
Bill.Roscoe@comlab.ox.ac.uk
[3] Formal Systems (Europe) Ltd
michael@fsel.com
WWW home page: http://www.fsel.com
[4] Worcester College, University of Oxford.

Abstract. The ubiquitous computing paradigm will involve both the storage of increased amounts of valuable or sensitive digital data, and an increase in potential interfaces to networks and devices where such data is stored. Such information systems will be relied upon to provide adequate data security. It is imperative that techniques be developed to assure the trustworthiness of the technologies from which they are built. Formal verification provides the tools and techniques to assess whether systems are indeed trustworthy, and is an established approach for security assurance. When using formal verification for security assessment one of the most important concerns should be to be precise about the threat model. In the now extensive field of the formal analysis of crypto-protocols, the Dolev-Yao threat model is the *de facto* standard. This paper discusses how and when this threat model may be inappropriate for use in the ubiquitous computing environment and that it may be appropriate to assume the existence of multiple channels exposed to different threat models. We then present a selection of crypto-protocols for ubiquitous computing scenarios that require this heterogeneous approach.

1 Introduction

1.1 Ubiquitous computing environments

Future ubiquitous computing environments will consist of computational devices embedded in and pervading all parts of our environment. These devices will be capable of transmitting and receiving data, and providing varying degrees of computational processing power. Such huge numbers of communicating devices will provide and enable multiple dynamic networks at any one location. The

* This research is being conducted as part of the FORWARD project which is supported by the U.K. Department of Trade and Industry via the Next Wave Technologies and Markets programme. www.forward-project.org.uk

types of these networks will range from standard client-server architectures to dynamic ad-hoc peer-to-peer networks, including a spectrum of hybrids in between. In order to utilise future services users will be surrounded by intelligent and intuitive interfaces, some consciously controlled and others operating more-or-less autonomously, capable of providing information and communication facilities efficiently and effectively. These bespoke interfaces will be accessible from diverse geographic locations much like the Internet of today. The interaction of such bespoke interfaces with local dynamic networks will enable sophisticated ambient intelligence systems.

As ubiquitous computing technologies become common-place in our societies, companies, organisations and individuals will increasingly depend on them. It will be challenging to navigate through the copious amounts of information and services on offer. Various research initiatives such as the EU Framework 5 PAM-PAS project [10] and research funded by the U.K's Department of Trade and Industry T-SAS project [14], predict that autonomous computing agents will act and negotiate on users' behalf. This would further increase the range of critical and sensitive transactions.

The unobtrusive nature we expect from this new generation of computing, and its very ubiquity (almost unavoidable presence in every part of life), will make it easy for users to forget security requirements yet make these especially essential. This imposes an obligation on the engineers who create these systems to produce and satisfy appropriate security specifications, and to argue this persuasively to users and, hopefully, to standards authorities.

Whereas in some circumstances ubiquitous computing applications may behave like traditional ones and have readily available links to trusted third parties and a usable PKI, we believe that this will often not be the case. This might be through the use of these systems in isolated circumstances or simply because of problems with scalability, lack of resources, interoperability or lack of trust. In this paper we therefore concentrate on the more difficult case and see what can be achieved without a traditional PKI.

1.2 Security assurance and formalisation

The difficulty of building secure, trustworthy systems in this world of vastly more complex information relationships will be greater than ever. This highlights an increased need to be able to verify (or at least validate) that proposed designs and implementations are indeed sound. However, the increased complexity of ubiquitous computing environments will exacerbate the difficulty of achieving bolt-on information assurance. See [11] for a discussion on the difficulties of bolting security onto a system implementing certain fault-tolerance techniques.

Basic security properties such as authentication and secrecy are frequently achieved via key agreement protocols; these protocols are traditionally designed for an environment which differs from what we expect in ubiquitous computing in two vital respects:

1. Processes identify each other by long-term names, either because they have *a priori* knowledge of the partners they wish to do business with, or per-

haps in some cases because there is no better way to identify them. In a ubiquitous environment we are most unlikely to know the name of a process we encounter (the latter probably being a consequence of locality), and mass-produced processes may well not have names or unique public-key certificates. Even if we did discover a process's name, this would not in many cases be a basis for trust: see [6] for a discussion of attribute versus identity authentication.

2. Protocols often rely on the presence of trusted third parties, for example to judge the validity of partners in some public key infrastructure. This may not always be a realistic prospect as argued above.

Formal methods and formal assurance techniques provide the proven capability to verify systems, in order to support high assurance design, implementation or accreditation. A fundamental component of any formal analysis of security properties is to construct an appropriate threat model for the environment in which the system is designed to be working. If the threat model represents too weak an attacker then it may be that security can be proven in the formal model, but does not hold in the real implementation. Conversely, it may be that a stronger attacker than necessary is modelled, which could then place constraints on the implementation which could have been avoided.

In this paper the security service that we study is authenticated key agreement. Currently, the *de facto* standard threat model for analysing key agreement protocols is the Dolev-Yao model.

1.3 Is the Dolev-Yao threat model appropriate?

The bulk of classic crypto-protocol analysis is predicated upon an almost omnipotent attacker, limited only by (more-or-less perfect) cryptography. The Dolev-Yao model [7] supposes an intruder who effectively controls the communications network, and is therefore capable of overhearing messages between legitimate principals; of intercepting messages and preventing their delivery to their intended recipient; and – within the limitations of the cryptographic mechanisms involved – of synthesising messages from data initially known to him together with fragments of any previous messages and delivering them, apparently originating either from an identity under his control or indeed from any other principal.

This is clearly a worst-case scenario for a protocol to cope with, although it may unfortunately be realistic for the interned and similar environments. In any case, it is arguably best to err on the safe side: a protocol which exhibits no flaws under these assumptions will *a fortiori* be secure against a less potent attacker. But it is undoubtedly easier to come up with protocols which achieve their goals of confidentiality or authentication if one is reasonably able to assume a more restricted threat model: Broadfoot and Lowe [4], for example, have shown the security of transaction protocols relative to assumed properties of a secure transport layer. Equally, there may be intrinsic properties of the network under consideration which justify cutting back the Dolev-Yao capabilities.

The main thrust of this paper is to develop a more flexible threat model which allows us to handle ubiquitous applications better. We will demonstrate this with a number of examples and show that it helps to explain some existing literature.

1.4 Paper outline

In the rest of this paper we first provide a brief overview of the threat models assumed by some of the extant literature on ubiquitous computing security[1]. Then in Section 3 we present a range of restricted threat models and describe example use scenarios. In Section 4 we discuss how these revised threats may formally be modelled for automated verification. Finally, in Section 5 we present our conclusions and some ideas for future work.

2 A Brief Overview of Threat Models

A superficial survey of the rapidly growing literature in this area reveals that, from a formal perspective, the threat model is not well- specified. One reason for this might be that the extant work is dominated by approaches to *devising* security mechanisms and policies, while the problems of verification have, as yet, received little attention. Our brief look is based on three items:

1. Balfanz *et al.*'s work on authentication in ad-hoc networks, [2].
2. Bootstrapping security in mesh networks [1].
3. Proxy based security protocols [5].

The first two also provide added context for the presentation of our protocols; the three together provide a cursory sample of the subject. A commonality of all three is that it is difficult to distinguish the capabilities of the attacker (in other words what the attacker is able to do to attempt to compromise their mechanisms) from the constraints on the attacker (which are in effect the security properties achieved by their mechanism, against an attacker of uncertain capability). Below we attempt to outline the threat model utilised by each.

2.1 The Threat Model when Trusting Strangers

The problem addressed by Balfanz *et al.* in [2] is of a user wishing to print a confidential document, residing on a PDA, on a public printer – the printer may, for instance, be in an airport. Communication between the the printer and the user's PDA is via a wireless connection of some type (the exact communications protocol is not relevant here). The user requires some assurance that the printer

[1] In fact much of the work that we examine describes itself as security for ad-hoc networks, but, in essence, this is a part of the ubiquitous computing security field.

they are sending the document to is indeed the one they are looking at, as opposed to some other device elsewhere within communications range.[2]

Balfanz *et al.* eschew dependency on any trusted infrastructure; instead their proposal to secure the PDA to printer wireless link is based on the concept of a "location-limited channel", as found in 'The Resurrecting Duckling' of Stajano, [13]. Essentially, public key information is exchanged on physical contact between the PDA and the printer and then standard authenticated key exchange protocols secure the wireless printer to PDA link. No pre-existent authentication mechanisms, such as certificates, are needed, thus they are establishing trust between strangers. However, the location-limited channel requires users to touch their chosen printer with their PDA, on an appropriate physical interface (this being the location-limited channel).

They describe their solution as "secure against passive attacks on the privileged side-channel and *all attacks* on the wireless link". The emphasis is ours and we assume "the privileged side-channel" is the same as the location-limited channel. In summary:

- Attackers should not be able to transmit on location-limited channels, though this constraint is a little equivocal since they add "or at least to transmit without being detected".
- Attackers should not be able to eavesdrop on a location-limited channel, but it appears as if this is the security property achieved, against an apparently unspecified attacker.
- They further clarify that *active attacks* are where the attacker transmits, and they imply that *passive attacks* mean eavesdropping. Other types of attack, such as blocking communications, appear not to be considered.

The location-limited channels are the cornerstone of Balfanz *et al.*'s solution, but they are only the means to an end, namely a secure wireless link between the PDA and the printer. The paper's abstract makes the rather sweeping statement that this second secure channel will be secure against "all attacks", but it is not clear the domain over which *all* ranges.

2.2 The Threat Model when Bootstrapping Group Keys

For our second example imagine a set of people meeting in some place and wanting to work together securely. Of course, they will have their PDAs (laptops, or whatever) and so they will want these PDAs to form a 'secure' network[3]. Here

[2] One may not be entirely convinced of the plausibility of this scenario, or that there may not be more commercially attractive solutions, such as establishing a key by swiping the credit card to be charged. However, it is illustrative of a class of authentication and key-negotiation problems which is likely to become more important as ubiquity strikes.

[3] This is an example of promotion of human trust, to extend it to cover their electronic representatives.

secure may be taken to mean that a group key exists and is known only to the PDAs owned by the people holding the meeting.

This problem is discussed by Asokan and Ginzboorg [1]; their solution is inspired by the EKE protocol of Bellovin and Merritt [3], which uses weak password-based encryption to agree a strong encryption key. Asokan and Ginzboorg's work also extends this concept from point-to-point key agreement to group key agreement. The basic idea is that the users agree a password and then they type that password into their PDAs. This password makes the weak encryption key, which is, nevertheless, sufficient for the legitimate PDAs to agree a strong key - using the protocols that they present. Thus they provide a solution for bootstrapping security that requires no pre-existent electronic trust. In effect they have *manual initialisation* of trust, since the users are (implicitly) responsible for controlling the exposure of the password.

The well-known "Alice-Bob" notation of the crypto-protocol community is utilised, which usually implies the standard Dolev-Yao threat model, however, this appears not to be the case. The threat model is implicitly included in their discussion security requirements. The security properties the protocols are required to uphold include:

- Contributory key agreement.
- Tolerance to disruption attempts.

Contributory key agreement means that malicious principals cannot limit the size of the key space and so enable cryptanalysis attacks. As such it implies a cryptanalysis enabled attacker, which differs from a pure Dolev-Yao attacker.

The meaning of "tolerance" in "Tolerance to disruption attempts" is not clear, we assume it means that "disruption attempts" cannot violate the other security properties. In explaining what "disruption attempts" mean the authors state that:

> *The strongest attacker can disrupt any protocol by jamming the radio channel or modifying communication among legitimate participants. However, a slightly weaker attacker who can insert messages but cannot modify or delete messages sent by others is also of interest in this scenario. We will consider disruption tolerance against this type of disruption attacks only.*

The strongest attacker sounds similar to a Dolev-Yao attacker, but the weaker attacker is more limited. Thus the proposed attacker appears to be a hybrid of something weaker than Dolev-Yao, but with cryptanalysis capabilities.

2.3 The Threat Model in Proxy-Based Security Protocols

In the ubiquitous computing future many devices will have relatively limited computing resources. To overcome this limitation it is often touted that computationally limited devices should be endowed with virtual proxies. These virtual

proxies will have access to high-bandwidth communications and plentiful computing resources. In [5] Burnside *et al.* propose a security architecture for virtual proxies[4]. Security concerns in systems of this sort will of course be very many and very complex, their paper concentrates on presenting two security protocols:

- A protocol for device-to-proxy secure communication.
- A protocol for proxy-to-proxy secure communication.

We assume (though it is not stated explicitly) that "security" means that the communication is protected by *confidential* and *authenticated* key agreement[5]. Unfortunately, the paper does not discuss the device-to-proxy *protocol*. Rather the assumption is stated that "the proxy and device share 128-bit keys". The protocol for initialising such shared keys would perhaps have been of most interest to us. For this protocol the paper only discusses the messaging hashing and encryption techniques used, which imply that only a cryptanalysis attacker is being considered, though this is not stated.

The proxy-to-proxy protocol, is described as a "typical challenge-response protocol", which, to a formalist, might imply a Dolev-Yao attacker. However, the only explicit discussion of "the adversary" is as part of a brief discussion of how time-stamps are used to protect against replays. The state- of-affairs is made a little more confusing when the authors make clear that the protocol does *not* provide a range of security services or defences against attacks "from the network". The authors then discuss how security services can be layered, and indicate that added security may be achieved by layering their services on protocols such as SSL/TLS. Thus, if formal verification were to be applied to their architecture, then an approach such as that developed in [4] might apply.

2.4 The dual-interface assumption

The threat models presented in Sections 2.1 and 2.2 above utilise properties of a dual interface to provide security. For example, both require a channel involving physical contact or human interaction for *part* of their protocols (while "bootstrapping" a relationship, for instance). Such a dual interface seems to be a key to this genre of problem[6].

In the ubiquitous arena, where most communications are through the essentially broadcast medium of wireless, it may sometimes be sensible to restrict the powers of the attacker: in particular, while a hostile agent can undoubtedly overhear any message or perhaps, by some form of jamming, prevent its delivery, it is arguably unreasonable to suppose that it can do both at the same time. (This is only reasonable when we know the intended recipient is active and is

[4] In fact their work concerns using proxies for resources discovery.
[5] Interestingly, by authentication they implicitly mean *attribute authentication*, much as espoused by [6].
[6] Unfortunately a lack of information in [5] means that we are unable to tell whether the same dual interface would be required or not.

within range.) Thus we gain the advantages of some kind of atomicity in transmission: if anyone receives a given message, then the intended recipient (also) does. Note that this argument holds good only for direct, unmediated, radio communications: if the message is relayed by a router node, for instance, then it would be open to a nearby attacker to overhear the first leg of its journey and to interfere with its second. In this case it might be possible to design acknowledgement schemes which effectively restore atomicity, but these in turn would require careful verification.

Another variant might be appropriate to the airport-printer problem [2, 6], discussed in Section 2.1 : with suitable tamper-proof hardware (and a flashing light to show "secure" printing) a printer might be able to achieve a one-way channel that was secure against messages being faked on it (and, perhaps less crucially, against overhearing, in the absence of cameras focused on its output tray). It is probable that in many cases additional security may be obtained at higher cost: be that in monetary terms, lower bandwidth, or reduced ease of use.

The reliance on a duality of channels, where one can be relied upon more than the other to provide certain security charateristics, does not appear to be unreasonable. However, it goes without saying that where a weaker threat model is assumed, careful examination of an implementation is required to ensure the model accurately captures the danger. What follows is a formalisation of three different variants of such systems.

3 Variant Threat Models

3.1 Twin-Channel Threat-Model Variants

In our examples below we presume the existence of two communications channels: N and E. The first channel, N, is the high bandwidth bidirectional medium with low or unreliable security. This represents the *network* wireless communications medium, such as wireless LAN, Bluetooth, or IrDA. The second channel, E, represents the more costly channel with higher security, which may according to the details of the scenario be either uni-directional or bi-directional. E typically represents an *empirical* channel, such as reading a message on the printer display panel, physically inputing a code via the printer control panel, or checking that the flashing light is present as in the example above. We will generally want to minimise the use of channel E and seek to delay it until relatively late in a protocol since it is likely to involve the human user in some effort.

By varying how the attacker can manipulate these two channels (the attacker's capabilities on E being the most limited), a variety of hybrid threat models can be created. We will express threat model variants using the following notation, each specifying a restriction placed on the powers of the Dolev-Yao attacker:

- AOT_C : the attacker cannot both block and hear messages at the same time on channel C, where AOT stands for *Atomicity of Transmission*.

– NS_C : the attacker cannot spoof messages on channel C, where NS stands for *No Spoofing*.
– NOH_C : the attacker cannot overhear messages over channel C, where NOH stands for *No OverHearing*.

Combined restrictions can be represented with a hyphen, such as: AOT_{C1}-NS_{C2}, which means the attacker cannot both block and hear messages at the same time on channel $C1$ nor can it spoof messages on channel $C2$. The channels will of course be either N or E. If no restriction is specified for a channel, then that means the standard Dolev-Yao model applies.

The following two scenarios will be used to demonstrate the usefulness of these variant threat models. We have considered the complete matrix indexed by the assumed properties of each channel and the direction of E, and we have found credible protocols for the majority of the different configurations. Due to space limitations, however, we present only three distinct models here; a further paper will set out our exploration of the broader space.

3.2 Scenario 1 - The wireless printer

We begin by returning to the airport-printer problem of 2.1. We will assume that all suitable printers are manufactured with a generic public/secret key pair. In addition we will consider the case where the printer possesses its own unique public/secret key pair. What is required is an authentication protocol which identifies the desired device as being the one on the receiving end of the communication channel, and the only one which can read the sent data. In addition one might want to verify some behavioural attributes of the device, but we will not consider that question here beyond checking the possession of the key belonging to the appropriate genus of printer.

NS_E-AOT_N Here we make the assumption that the user A has a unique key certificate, $kc(A)$; and that the printer is manufactured with knowledge only of the generic key pair associated with the class of printers to which it belongs, P. Moreover we assume that it is fitted (in a reasonably tamper-proof way) with a light which flashes while (and only while) it is printing data that it itself is communicating as part of a protocol run. This effectively gives a no-spoofing assurance that the printer expelling the paper is the one generating the contents of the paper. An alternative mechanism might be to use the LCD panel on the printer as a reliable medium between the printer and its user. Without some such facility, it would be hard to avoid the possibility of a suborned device-in-the-middle engaging in the protocol and simply using the intended printer as a slave to reprint what the protocol requires.

A wants to check that the printer she is communicating with over the N channel is indeed the printer she can see, as in Figure 1.
More specifically, the security goal is to establish a shared secret known only to the user and that specific printer (which may then be used as a symmetric-encryption key for the document, for instance).

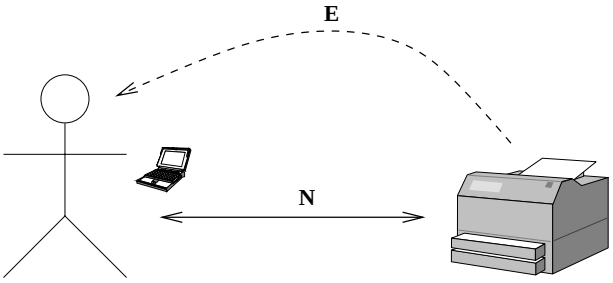

Fig. 1. A unspoofable channel E exists between the printer and the user; the network N gives a bidirectional, atomic link.

Our first protocol uses the fact that the attacker cannot both block and receive network messages at the same time, thus AOT_N. It also depends on the channel E, running from the printer to the user, being impervious to spoofing, NS_E. The protocol proceeds as described below (the N or E subscript on the arrow denotes the channel over which the communication event occurs). If, however, at any stage before the protocol has completed either the user or the printer receives additional messages 1 to 4 directed to them then the protocol will abort.

$$
\begin{array}{ll}
1.\ A \rightarrow_N P(B) & : \{A, kc(A), N_A\}_{pk(P)} \\
2.\ B \rightarrow_E A & : \text{print } A, N_A \\
3.\ B \rightarrow_N A & : \{k, N_A, N_B\}_{pk(A)} \\
4.\ A \rightarrow_N B & : \{N_B, N'_A\}_k \\
5.\ B \rightarrow_E A & : \text{print } N'_A
\end{array}
$$

In Message 1 A sends the printer a message containing her name, her key certificate and a random nonce, all encrypted with the generic printer public key. The printer then prints both A's name and the nonce, which A verifies empirically over channel E. At this stage, given that an attacker cannot spoof messages on channel E, A knows that the printer she is looking at has received Message 1 and is a printer, but she cannot exclude a corrupt printer-in-the-middle having overheard and understood Message 1. We can, however, guarantee that the printer heard Message 1 as sent by A, thanks to AOT_N; and that an attacker has not sent another Message 1 because of our assumption about B aborting. Therefore, the $kc(A)$ which B holds was indeed just sent by A.

In Message 3 the printer sends A a message containing a session key, k, the previous nonce N_A from Message 1, and a new nonce N_B, all encrypted using A's public key, extracted from the $kc(A)$ received in Message 1. A then sends a message to the printer which contains the second nonce N_B and a new nonce N'_A, encrypted using the key sent in Message 3. Since only A can have read the contents of Message 3 (it was encoded using her unique public key), it follows that only A and the device which sent Message 3 know the key k. However since

an intruder knows N_A and may have blocked Message 3 from B and sent his own, A does not know that the Message 3 that she heard was from B.

The printer then prints the new nonce N'_A, which A verifies over channel E. Since this new nonce was sent in Message 4 and was encrypted using the key k in a message containing N_B, the printer would not print N'_A unless the Message 3 that A received really was from B. It follows that at this point A is certainly connected to B (the desired printer).

Note that the first printed output also serves to guard against spoofing on channel N, since if an attacker were to force B to abort after this point in the run, then the attacker could not get beyond this point without a further Message 2 being spotted, which A could see. This observation is somewhat application specific, since it may not always be the case that the E channel that B would use with an intruder would be observed by A. For this reason we additionally require that a "printer" aborting a run after Message 2 should send an E-message to A saying so.

If we were to assume a conventional Dolev-Yao model for the high bandwidth medium then the above protocol would not be so secure since a printer-in-the-middle could act as an intermediary, sending A's values for N_A and N'_A on to the real B for printing. The point where he might have problems doing this is in understanding Message 3 if it really was A's real public key certificate that was sent to B in message 1. However in the absence of a usable PKI this may not be true, and it may be possible to send B a public key "for A" that he has made up.[7]

We do have protocols which we believe work with the same model for E but with full Dolev-Yao on the high bandwidth channel. They differ in having B print out hashes rather than literal values sent by A. The latter might be a disadvantage where a human is expected to check equality. These will be discussed in a later paper.

NOH $_{\boldsymbol{E}}$ As a variant of this scenario, illustrated by Figure 2, let us now assume that the printer has its own unique public/secret key pair, and that the E channel communicates from the user to the printer (in the opposite direction to the first example), perhaps by discreetly pressing buttons on a front panel. If we consider a different threat model, namely that of no overhearing on channel E, we can achieve the same goal by means of a different protocol.

$$
\begin{array}{ll}
1.\ A \rightarrow_N P(B) & : \{A, kc(A), N_A\}_{pk(P)} \\
2.\ B \rightarrow_N A & : \{B, kc(B), N_A\}_{pk(A)} \\
3.\ A \rightarrow_E B & : N'_A \\
4.\ B \rightarrow_N A & : hash(N'_A, pk(A), kc(B), N_A)
\end{array}
$$

In Message 1 A sends a copy of her key certificate and identity, and a nonce, encoded with the generic printer key, to the printer. The printer then replies in

[7] In this case the concept of sending a public key *certificate* as opposed to just sending a public key may have limited value.

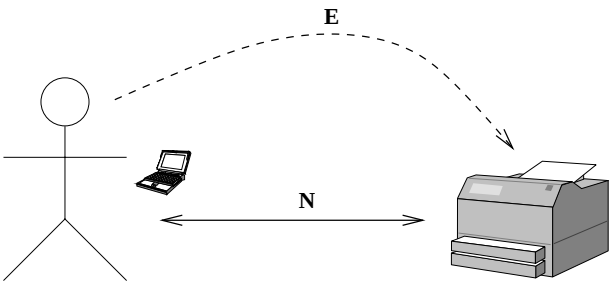

Fig. 2. Channel E now runs from user to printer, unobservable by the attacker. Channel N is bi-directional with the standard Dolev-Yao vulnerabilities.

Message 2 with a copy of its own unique key certificate and identity, and a copy of the nonce it received from A in Message 1, all encoded with A's public key. At this stage A knows that someone has received the original message, but she cannot be sure that it is printer B.

In Message 3 A inputs a nonce N'_A into the printer directly, over the E channel. In this threat model we are assuming that this input cannot be overheard, and therefore only the printer physically interacted with can know N'_A. So in Message 4 the printer sends a hash of the nonce N'_A, A's public key, the printer's key certificate sent earlier in Message 2 and the first nonce, N_A, sent by A. As a result, A knows that the printer she is communicating with over N is also the printer being communicated with on E. This message certainly originated at the physically present hardware, since only one printer can know N'_A; the inclusion of evidence of the identities of both A and B in the hash is necessary to prevent an intruder acting as a man-in-the-middle for Messages 1 and 2, and persuading each that they are engaged in the protocol with a different principal. If, for example, we were to have encrypted message 4 under $pk(A)$, this would have opened up a man-in-the-middle attack. Of course, it is essential in this protocol that E cannot be overheard, as otherwise an imposter who had been taking part in the rest of the protocol could forge Message 4.

Since N_A is a shared secret here, it can be used as a session key; alternatively the protocol leaves each of A and B with knowledge of each others' public keys so they can use these.

3.3 Scenario 2 - The PDA Mesh

In this example we use our hybrid threat model approach revisits the problem from Section 2.2, where a group of people want to agree a key across their electronic representatives (PDAs, or whatever). In our solution the E channel is used to agree a hash value, which will appear on the users' screens and which they can compare.

$\boldsymbol{NS_E}$ Here we again restrict the attacker on the E channel to NS, no spoofing. We will consider the example where each user possesses a unique key certificate pair, and a generic manufacturer PDA certificate. Each message is described as being sent from each user A to all other users: the first can be broadcast to *Every* B, but the rest are sent in turn to *Each* B that a Message 1 was received from:

1.Each A $A \rightarrow_N$ Every B : $\{A, kc(A), N_A\}_{pk(PDA)}$
2.Each A $A \rightarrow_N$ Each B : $\{$all Messages 1, $N'_A\}_{pk(B)}$
3a.Each A A displays : $hash(\{$all Messages 2$\})$, number of processes present
3b.Each A $A \rightarrow_E$ Each B : Users compare hashes and numbers on screens
4.Each A $A \rightarrow_N$ Each B : $hash'(\{$all Messages 2$\})$

We assume that the boundary between Message 1 and Message 2 is determined by some time-out. Tit is not necessary to include this in the security analysis since any failure to synchronise properly will only have the effect of causing the eventual hash values to differ, meaning that no node believes the protocol has completed successfully.

In Message 2 a given A knows that it has just sent nonce N'_A to a given set of identities, using the keys it received in Message 1. One purpose of Message 2 is to allow disagreements about the Messages 1's, in the most part, to be caught by the PDA's without troubling their users. When all the users agree on the hash of the values transmitted in Message 2, which they check on channel E in Message 3, and the number of participants displayed corresponds to their expectations, they know that there cannot be any additional devices that they are unaware of involved in the protocol. The number shows each of them that they are in a network of the right size, and the hash shows them they are in the same network. It follows that there can be no unwelcome participants. The nonces are present to ensure freshness and prevent replays of messages in previous sessions. Message 4 acts as a final message to close the protocol and confirm possession of the shared secret; this value may also be used as a session key. We have to use a different hash function because the previous value has been displayed, and in any case it reduces the probability of hash collisions.

4 Verification in the Presence of Restricted Attackers

One of the main challenges to using process-algebraic and model-checking approaches to verify (or, more strictly, to look for counterexamples to) the security of crypto-protocols was to find an effective way of modelling the attacker and his ability to recombine elements of the network traffic to date in decrypting incoming messages and synthesising messages to trick his targets. The solution devised for FDR and incorporated in Casper [9, 12] proved to be an enabling technology, and similar schemes have been used in many subsequent model-checking and related approaches, such as strand spaces [8].

Casper exploits the fact that the intruder intercepting a message and then "faking" it (unchanged) to its intended recipient as if from the sender amounts to much the same as eavesdropping on its uninterrupted transmission. This means

that all messages are in fact mediated by the intruder [12]. This simplifies some specification idioms, but makes it harder to implement the weaker attackers we are considering.

In the original concept [9], however, there were separate channels: *comm* to represent direct communication, and *take* and $fake$ to represent interception and introduction by the intruder, respectively. Renaming is used in the CSP model so that sending on *comm* and *take*, and receiving on *comm* and $fake$, are indistinguishable to the principals. In this context it is straightforward to modify the definition of the intruder to reflect restricted powers:

- leaving the intruder's knowledge unchanged after a *take* gives the AOT semantics;
- barring communications on a subset of the message space on $fake$ can model a reliable one-way *NS* channel;
- disconnecting both *take* and the tap on *comm* on a subset of the message space captures the confidentially of *NOH* transmission.

Thus there is no difficulty to the mechanical checking of these weakened attackers, at least in the CSP/FDR/Casper paradigm. (Proponents of other technologies must answer for themselves! It would be interesting to learn whether the encoding of the attacker in alternative approaches is sufficiently flexible to cope with this kind of variation in its powers.)

5 Conclusions and Further Work

The problem of formalising the threat model is fundamental to the formal analysis of security. But a superficial look at the literature on security for pervasive computing shows that it appears to be informally (and perhaps a little loosely) specified. In our own examination of the problem, for authenticated key-agreement protocols, we found that the standard Dolev-Yao model was arguably unrealistic, and yielded a dearth of protocols which are simultaneously both secure and useful, in the absence of the assumptions and support infrastructure postulated for more traditional environments. This is consistent with the need that other researchers have felt to invent mechanisms outside the normal communications structure, to bootstrap trust in some way.

Among the redeeming features of the ubiquitous world are that there will often be physical proximity, which can sometimes be exploited to allow contact-based communication, and that the unique data-processing capabilities of human agents can also be brought into the equation as a last resort. The properties that such mechanisms win for us appear to be reasonably straightforward to capture and model. We believe that developing an understanding of the power of the full range of variants (and intuition as to how they may be implemented) and of their interactions will help in designing techniques for bootstrapping security in a world where no backbone infrastructure is required (or available).

Further work is needed in crystallising the notions introduced in this paper, and a subsequent paper is planned that will populate the matrix of attacker capabilities with appropriate countermeasures, and explore the ideas in this paper

more deeply. A number of interesting technical issues regarding the desirability of including the various potential component data in messages or hashes have arisen in the course of our initial investigations, and these aspects, too, merit further exploration.

6 Acknowledgements

The authors would like to thank give special thanks to Gavin Lowe for stimulating discussions.

References

1. N. Asokan and Philip Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.
2. D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to strangers: Authentication in ad-hoc wireless networks, Feburary 2002. In Symposium on Network and Distributed Systems Security (NDSS '02), San Diego, California.
3. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Press, May 1992.
4. Philippa Broadfoot and Gavin Lowe. On Distributed Security Transactions that use Secure Transport Protocols. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, Monterey, CA, June–July 2003.
5. M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, and R. Rivest. Proxy-based security protocols in networked mobile devices, 2002.
6. S. Creese, M. H. Goldsmith, Bill Roscoe, and Irfan Zakiuddin. Authentication in pervasive computing. In D. Hutter and M. Ullman, editors, *First International Conference on Security in Pervasive Computing*, Boppard, March 2003. Springer LNCS.
7. D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
8. F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
9. Michael Goldsmith and Bill Roscoe. The perfect 'spy' for model-checking cryptoprotocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Cryptographic Protocols*, Rutgers, 1997.
10. http://www.pampas.eu.org/.
11. Jan Peleska. Formal Methods and the Development of Dependable Systems. Technical Report 9601, University of Bremen, 1996. Project UniForM, www.informatik.uni-bremen.de/agbs/jp/papers/depend.html.
12. P.Y.A. Ryan, S.A.Schneider with M.H. Goldsmith, G. Lowe, and A.W. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.
13. Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In B. Christianson, B. Crispo, and M. Roe, editors, *Security Protocols, 7th International Workshop Proceedings*, pages 172–194. Springer LNCS, 1999.
14. http://www.gpc.ecs.soton.ac.uk/research/TSAS.html.

Belief and Trust in Fault Tolerance

Geraint Price

Information Security Group,
Mathematics Department,
Royal Holloway University of London,
Egham, Surrey, TW20 0EX,
United Kingdom.
geraint.price@rhul.ac.uk

Abstract. In this paper we modify the BAN logic to allow it to reason about an N-model redundant authentication protocol. Our goal in undertaking this modification is to understand how, under certain circumstances, the notion of belief changes when participants are not trustworthy. We conclude with a discussion on how our adaption of the logic effects the way in which protocol designers might consider reliance in the actions of principals.

1 Introduction

In this paper we modify the BAN logic [BAN89] to allow it to reason about a fault tolerant authentication protocol. Fault tolerance is a branch of computer science which provides a means of systems to continue in the face of an error without resulting in the failure of the system.

Our goal in undertaking this modification of the BAN logic is to build a foundation for understanding for how, under certain circumstances, the notion of belief changes when participants within a protocol are not trustworthy. We believe that this is a first step toward understanding how best we can make use of the mechanisms fault tolerance provides us with.

Our adaption allows us to relax the assumption within BAN that all participants are trustworthy. This notion is not new to the community, and Pancho [Pan99] discusses this shift in protocol analysis.

We use BAN as a tool for a number of reasons. It is easy to understand and has a very intuitive feel to it, which provides us with an ideal starting point for our analysis.

The remainder of this paper is organised as follows. In Section 2 we expand on our discussion on the effect of fault tolerance on *belief* within the BAN logic and in the processes modify the logic to our requirements. In Section 3 we discuss how the adaption we make to the logic can help us understand how the change that using fault tolerance brings might be borne out in protocol design. We examine the scope for future work and draw some conclusions in Section 4

2 Belief

In their paper introducing the BAN logic, Burrows, Abadi and Needham [BAN89] use a notion of *belief*[1] in order to analyse protocols.

Their analysis is based around the transformation of each protocol message into a logical formula. This process results in an idealised version of the original protocol. They also annotate each idealised protocol with assertions which are expressed in the same notation used to write the messages. An assertion usually describes the beliefs held by the principals at the point in the protocol where the assertion is inserted.

Their aim is to provide a logical means by which the goals of a protocol may be formulated and tested against a set of pre-conditions.

A strong assumption in their paper is that the principals involved in the protocol must trust the other principals involved in the protocol to behave correctly (e.g. that the Server in a Key Distribution Protocol does not mis-use the keys available to it).

While this may be realistic in some environments, it is easy to see that this may not always be the case.

Our aim in this paper is to look at the way in which these beliefs and goals might change in an environment where assurance in the actions of others does not necessarily hold.

We are not the first to consider relaxing this stringent assumption of the honesty of the players in a protocol. Pancho [Pan99] discusses the shift in protocol analysis from the early work presented in the original BAN paper. She charts the way in which modifications in protocol analysis can lead to new attacks. She also notes that several of these new attacks are based on modifying the context in which the original design assumptions were made. Pancho goes on to say that this is not necessarily a bad thing, but points out that it is important to understand, and be explicit about, the assumptions made when designing a protocol in the first place. She notes that one way in which such attacks are discovered is to relax the assumption that all participants are honest. This results in a need to consider "insider attacks" as well as the traditional external attacks discussed in the original BAN work.

2.1 Belief and N-Model Redundancy

As a focal point for our study, we use the canonical example within fault tolerance of N-Model redundancy [AL81, pages 68–69]. This mechanism is a method of circumventing the failure of a component by providing N replicates of that component and then using a *voter* to provide the final result. The use of a voter masks the failure of a subset of the components.

When using this mechanism in security, its primary motivation is to allow the malicious failure of some subset of the components to be masked by the benign

[1] We note that this is not meant to imply belief in the real world meaning, but is meant to dictate how a principal may act if a function were true.

components. In order to accommodate this into a logic such as the BAN logic, we need to make some changes to the structure of the logic.

In making these changes, our aim is to build an understanding of how a principal's set of beliefs, along with their actions upon that set, adapt when fault tolerant mechanisms are used. In doing so we can better understand the way in which secure system design can embrace fault tolerant principles in a sympathetic manner.

Again, our notion of borrowing from fault tolerant community is nothing new. Meadows was one of the earliest to call for the use of dependability paradigms within security [Mea94] and has since charted the progress of such work [Mea95,MM99].

2.2 Key Distribution

To illustrate our example, we provide a simplified Public-Key Distribution Protocol and naively modify it to follow the N-Model redundancy principle mentioned above.

A Simplified Key-Distribution Protocol

$(1) A \rightarrow S : \{A, B\}_{K_S}$
$(2) S \rightarrow A : \{A, B, CERT_B, K_B\}_{K_S^{-1}}$

Where, A and B are the principals, S is the public key distribution server, $CERT_B$ is S's certificate on B's public key, and K_S and K_S^{-1} are S's public and private keys respectively.

To modify this to a simple N-Model redundant service, let G be a group of servers, where S_i is an instance of the public key distribution server, where $i = 1 \ldots N$. Let $\Rightarrow$ denote a simple broadcast send and receive mechanism that A uses to contact the group of servers G.

An *N*-Model redundant KDP

$(1) A \Rightarrow S_i : \{A, B\}_{K_{S_i}}$
$(2) S_i \Rightarrow A : \{A, B, CERT_B, K_B\}_{K_{S_i}^{-1}}$

A then compares the results from each of the servers S_i and if there is a clear majority that agree on B's key, then A uses that key to contact B.

Although the protocol we present is effectively a straw man protocol, we note that the voting mechanism used in it, is the principal that underpins previously fielded implementations of group agreement mechanisms within security – such as the Rampart system of Reiter [Rei94b,Rei94a].

2.3 Re-defining principals

We now expand on the structure of the logic, modifying the notion of a principal involved in a protocol and modifying the derivation rules to accommodate these changes.

In the original BAN logic, all principals are considered trustworthy. The beliefs that the protocol participants derive are evolved from what are deemed to be the actions of other trustworthy principals. This assumption does not hold if we are to consider faulty – in our case malicious – principals. We now re-define the types of principals to the logic to accommodate our N-Model redundant protocol in Section 2.1. In doing so, we introduce two new principals, *group* and *group member*:

P : Principal : P is a principal as traditionally defined in the BAN logic.
G : Group : A group is a collection of principals, which have a recognisable form within the model. The abstraction of a group allows us to derive beliefs about the action of a replicated service that are otherwise unattainable if we independently assess the statements of individual group members.
S_i : Group member : An identifiable member of a given group G. As we will discuss in Section 2.4 and Section 2.6, their ability to speak with authority in the terms of the logic needs to be restricted, but their inclusion is necessary in order to form beliefs about the actions of the group as a whole.

We assume that instances of each type of principal are identifiable as such within the protocol environment.

When analysing the pairwise communication channels between each principal type (e.g. when a group member sends a message to an ordinary principal), it is clear that the logical postulates that allow us to reason about communication need to be modified.

With traditional BAN logic, a principal is allowed to reason about what another principal has said because the channel is cryptographically protected. In our new environment, the assumptions made in the BAN logic about the means by which that channel is protected do not hold. For example, a principal is not going to trust the individual group members to share a single key. In this case, it would make little sense for a principal to attribute anything to a message received by a key shared between itself and a group of the type introduced above.

Table 1 is the set of channels we allow in our modification. We consider the type of sender by row and the type of receiver by column. We sub-divide each pair-wise link according the the type of protection reasoned about within the BAN logic. Within each cell, we stack the postulates in the following order: symmetric key ($P \overset{k}{\longleftrightarrow} P$), public key ($\overset{k}{\mapsto} P$), shared secret ($P \overset{x}{\rightleftharpoons} P$).

We place the type of communication allowed or a period where the column and row meet as appropriate. For example, a *principal* can reason about a message from a *group member* using a symmetric key ($S_i \overset{k}{\longleftrightarrow} P$), but a *group* cannot reason about a message received from a *principal* with the use of a shared secret ($.$).

		Receiver		
		P	G	S_j
Sender	P	$P \overset{k}{\longleftrightarrow} P$ $\overset{k}{\mapsto} P$ $P \overset{x}{\rightleftharpoons} P$	. $\overset{k}{\mapsto} P$.	$P \overset{k}{\longleftrightarrow} S_j$ $\overset{k}{\mapsto} P$ $P \overset{x}{\rightleftharpoons} S_j$
	G	. . .	. . .	. . .
	S_i	$S_i \overset{k}{\longleftrightarrow} P$ $\overset{k}{\mapsto} S_i$ $S_i \overset{x}{\rightleftharpoons} P$	. $\overset{k}{\mapsto} S_i$.	$S_i \overset{k}{\longleftrightarrow} S_j$ $\overset{k}{\mapsto} S_i$ $S_i \overset{x}{\rightleftharpoons} S_j$

Table 1. Pairwise sender / receiver – key / secret belief postulates

The fact that the central row is filled with periods indicates that we are unable to directly associate anything with a group G purely by receiving a message from that group. We need to build beliefs about what a group says without a group being able to directly say anything. To achieve this, we now form an extension to the original BAN rules that allow us to derive belief.

2.4 Changes to the logical statements

The core of the logic is built around a set of constructs which allow us to reason about the beliefs held by principals. We quote the following definition of two of the constructs from the original paper to provide a flavour of what the constructs aim to capture:

> "$P \models X$, *P believes X*, or P would be entitled to believe X. In particular, the principal P may act as though X is true. This construct is central to the logic.
> $P \triangleleft X$, *P sees X*. Someone has sent a message containing X to P, who can read and repeat X (possibly after doing some decryption)."

As noted earlier, the postulates on which the BAN logic derives its goals are built around a belief in the actions of principals. We now look at how those beliefs change in our environment.

The following postulates need no change in our revised model: $P \models X$ (*P believes X*), $P \triangleleft X$ (*P sees X*), $\#X$ (*X is fresh*), $\{X\}_K$ (*X encrypted under key K*), $< X >_Y$ (*X combined with formula Y*).

The other logical postulates need some changes to their definitions and we discuss each in turn below:

$P \mid\sim X$ *P once said X* - This postulate needs to be modified slightly. This is necessitated by the notion of different types of principals we define above and how

to attribute 'saying' in our new environment. Principals and group members should be allowed to say things directly, but there needs to be other means of building up the ability of a group to say something.

$P \Rrightarrow X$ *P has jurisdiction over X* - This postulate rests on the assumption that a principal, is trusted with regards the truth of X and will do so correctly. Again, this does not always hold for our revised model, and we will introduce new methods of achieving this Section 2.6. In contrast to the last point, our aim is to allow a group to have jurisdiction, but not allow individual group members to have jurisdiction.

$P \overset{k}{\longleftrightarrow} Q$ *P and Q may use the shared key K* - In N-Model redundancy, we need to differentiate between a server that is replicated and a member of the group that forms the server. This means that we need to change the assumptions by which a shared key can be used. Because of the nature of symmetric cryptography, it would be infeasible to share a key between multiple principals where a subset of those principals were assumed dishonest.

$\overset{k}{\mapsto} P$ *P has a public key K* - We change this postulate to accommodate our redefinition of principals explored above [2].

$P \overset{x}{\rightleftharpoons} Q$ *X is a secret known only to P and Q* - This postulate clearly does not stand between an individual principal and a group.

To allow the original aim of the logic to be achieved, we need to accommodate the points made above into the structure of the logic.

2.5 Threshold operators

We include two new operators into the logic to capture the nature of N-model redundancy. These operators can then be used within the logic with the same assumptions that current operators (e.g. encryption) can be used, without needing to include in the logic assumptions regarding their concrete implementation.

The two operators relate to the agreement of the output of the group and to a threshold of response:

- $\amalg$ - Signifies that a threshold of agreement between group members has been reached
- $\trianglelefteq$ - Signifies that the threshold for response has been reached

The second operator signifies that there is a sufficient number of responses such that agreement might have been reached (i.e. there may be dissenting members in the group). The first operator indicates that there is an agreement on statement made by the group, above the required threshold, between the group members who respond [3].

[2] Although there are mechanisms for sharing a secret key between a group of processes in order to generate a shared signature, we do not explore the effect of this on our logic here.

[3] The values used in these cases are going to particular to the voting protocols used.

2.6 Changes to the derivation rules

One of the more important changes we make to the logic is to the *jurisdiction* rule. A traditional principal is not effected in its ability to have jurisdiction over a statement. Although, the very notion of N-Model redundancy means we require a group have jurisdiction, without any of the individual group members having jurisdiction.

Here is an example of how the new principals can be used in existing derivation rules. In this case, how a principal determines that a group member once said X:

$$\frac{P \models \overset{k}{\mapsto} S_i, P \lhd \{X\}_K^{-1}}{P \models S_i \mid\sim X}$$

But as we do not allow S_i to have jurisdiction, we cannot move forward to achieve $P \models X$. In order to achieve this, we have to define a new rule. The following rule is a modification of the existing jurisdiction rule and makes use of the threshold agreement operator ($\amalg$). It makes an assumption that all S_i which once said something are all believed by P to part of the same group G.

$$\frac{P \models G \Rightarrow X, P \models \amalg S_i \mid\sim X, P \models S_i \in G}{P \models X}$$

That is, if P believes that group G has jurisdiction over X, and that a threshold limit of members of the group S_i agree on X, then P believes X.

We now demonstrate the use of the operator for reaching a threshold of response ($\trianglelefteq$) and how it affects P's beliefs with respect to what the group says.

$$\frac{P \models G \Rightarrow X, P \models \trianglelefteq S_i, P \models \not\amalg S_i \mid\sim X, P \models S_i \in G}{P \not\models X}$$

That is, if P believes that G has jurisdiction over X, and that a threshold of S_i have responded, but that a threshold of S_i do not agree on X, then P cannot believe X.

In order that P may reach some conclusion about the response of members of G, then P needs to be convinced that $S_i \mid\sim X$ is current, in order to achieve this we add a freshness rule.

$$\frac{P \models \forall S_i \mid\sim (X_i, Y), P \models \#Y}{P \models \trianglelefteq S_i}$$

That is, if P believes that for all S_i which said X_i, that they also say Y, which is fresh (e.g. a nonce based request identifier), then P believes that a threshold

of S_i have responded. Each S_i is allowed to say a different X for the response threshold to hold.

We now give a rule (which we term the *combination* rule), that is used to move from response threshold to agreement threshold.

$$\frac{P \models \trianglelefteq S_i, P \models \forall S_i | S_i \mid\sim X \ \forall S_i | S_i \in G}{P \models \amalg S_i \mid\sim X}$$

That is, if P believes that if a threshold of S_i responded with the same X, then the group G said X. It is this rule that allows us to derive belief about what a group said.

message meaning As stated earlier, we do not have the explicit means for a group to be able to say anything, this is because we do not allow a key to be associated with a group. To complete the changes to the statements we discussed in Section 2.4 we cover the message meaning rules here. While we do not allow groups to be represented, we flesh out the relationships from Table 1.

private keys

$$\frac{P \models S_i \overset{k}{\longleftrightarrow} P, P \triangleleft \{X\}_K}{P \models S_i \mid\sim X}$$

That is, if P believes that k is a good key between S_i and P, and P sees X encrypted under k, then P believes that S_i said X.

The following two rules carry on from this, allowing S_i to believe that P and S_j said X:

$$\frac{S_i \models P \overset{k}{\longleftrightarrow} S_i, S_i \triangleleft \{X\}_K}{S_i \models P \mid\sim X}$$

and:

$$\frac{S_i \models S_j \overset{k}{\longleftrightarrow} S_i, S_i \triangleleft \{X\}_K}{S_i \models S_j \mid\sim X}$$

public keys

$$\frac{P \models \overset{k}{\mapsto} S_i, P \triangleleft \{X\}_{K^{-1}}}{P \models S_i \mid\sim X}$$

That is, if P believes that k is a good public key for S_i and P sees X encrypted under k^{-1}, then P believes that S_i said X.

This rule carries to S_i, allowing S_i to believe that P and S_j said X:

$$\frac{S_i \models \overset{k}{\mapsto} P, S_i \lhd \{X\}_{K^{-1}}}{S_i \models P \mid\sim X}$$

and:

$$\frac{S_i \models \overset{k}{\mapsto} S_j, S_i \lhd \{X\}_{K^{-1}}}{S_i \models S_j \mid\sim X}$$

shared secrets

$$\frac{P \models S_i \overset{Y}{\rightleftharpoons} P, P \lhd < X >_Y}{P \models S_i \mid\sim X}$$

That is, if P believes that Y is a good shared secret for S_i and P, and P sees X combined with Y, then S_i said X.

This rule also carries to S_i, allowing S_i to believe that P and S_j said X:

$$\frac{S_i \models P \overset{Y}{\rightleftharpoons} S_i, S_i \lhd < X >_Y}{S_i \models P \mid\sim X}$$

and:

$$\frac{S_i \models S_j \overset{Y}{\rightleftharpoons} S_i, S_i \lhd < X >_Y}{S_i \models S_j \mid\sim X}$$

3 Trust: belief and control

In this section we discuss the nature of the changes we made to the logic in Section 2.4. Through this discussion we shed light on how we might, under certain circumstances, change protocol design to accommodate the notions we have put in place within the logic.

We adapted the BAN logic to accommodate N-Model redundancy in order to explore what happened to the belief of ordinary principals [4] if we adopted such a model.

[4] For example, clients of a server S in the cryptographic key distribution scenario.

As has been mentioned previously, there was a strong assumption in the early days of protocol design and analysis as to the honesty of principals that took part in a protocol run. As was noted by Pancho in her discussion [Pan99], this assumption has been eroded over time.

One of the main themes of Pancho's discussion is on how the change from honest to potentially malicious participants has come about. She notes that whilst the new attacks which are uncovered by these different techniques are significant, it is also important to be explicit in how the context and assumptions on which protocols are built are used and defined. The resulting protocol analysis could be operating on a different protocol context to that which the designers of the protocol envisaged.

BAN explicitly states that it assumes the honesty of participants. Under their original analysis, A's ability to derive belief from what another principal (B) says (X) is a function of the very fact that B said X.

When accepting the N-Model redundancy view presented in Section 2.1, we note that this no longer holds. Instead, the principal's belief now relies on his understanding of the trust afforded to the mechanism that combines the output of multiple parties within some recognised structure – i.e. in a traditional N-Model scenario, the *voter*. In this sense, A is moving her trust from B into the mechanism for scrutinising B's actions.

How this change is represented within our modification comes down to two things predominantly:

- The change in the jurisdiction rule that is applied to our newly added *group* principal. In order to allow a group to say anything, we are required to bind the jurisdiction rule to the threshold operator and hence the voting algorithm.
- Our addition of the *combination* rule that allows a principal to move from merely observing that members of the group individually said X to believing that the group as a whole said X.

It is by using these two in conjunction, we are able to achieve goals such as $A \models A \overset{k}{\longleftrightarrow} B$.

Returning to our use of N-Model redundancy as an example, figure 1 shows the difference, in what we term A's trust boundary, between trust in a standard cryptographic protocol and one in which an N-Model protocol is employed. In the diagram, V represents the voting mechanism.

Given that A now trusts a mechanism which we can potentially detach from S, the next logical step is to place the mechanism directly under A's control, for surely if A is to trust anyone, then it must be herself.

Dobson and Randell [DR86] criticise the Trusted Computing Base (TCB) view of secure computing in a distributed environment because it is generally at odds with the user-perceived model of the security-critical behaviour of the system. If we consider a client's *security policy* as an embodiment of a user's model, then by implementing the mechanism that enforces jurisdiction rule to

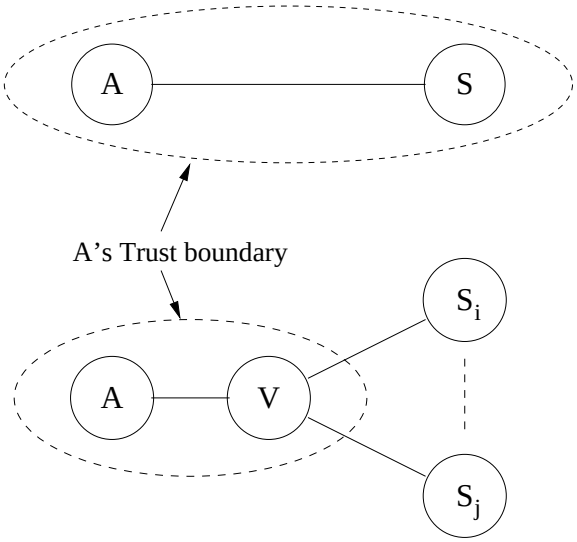

Fig. 1. Reliance in cryptographic protocols

be directly under the control of the client, can help the client enforce its policy locally.

This can potentially lead to less need for A to trust, and subsequently rely upon, other parts of the system.

The very notion of reducing reliance in other parts of the system – mainly considered single points of failure – is nothing new to security. What we believe we have illustrated, through our discussion in this paper, is that using fault tolerant mechanisms can, in some cases, help the client ensure that its security policy is being adhered to.

By necessity, such mechanisms are not achievable in every scenario. It will depend on the application, the security policies of others and the type of fault tolerant mechanism we aim to make use of.

In concluding this discussion section of the paper, we use an analogy to strengthen our case. One of the benefits of asymmetric cryptography, over symmetric cryptography, is the ability to reduce the reliance a client needs to place in a key server. This is typically achieved by allowing the client to generate its own keys, of which only the public key needs to be handed to the server. We believe that this is analogous to our example of having the principal be in charge of the completion of its *belief* through the management of the voting mechanism.

4 Conclusions

In carrying out this work, we set out to gain a greater understanding of what it meant for a fault tolerant mechanism to be used in a secure protocol.

We have started to build an understanding for the way in which fault tolerance can be viewed as a different tool for the designer from traditional security techniques.

By using BAN as a vehicle for our modifications, it has given us a platform on which we can reason about what including a fault tolerant mechanism does to the beliefs of those taking part within the protocol run. A crucial part of our work was modifying the jurisdiction rule within the BAN logic to encompass the notion of a *voting* protocol.

The notion of reducing the honesty of principals and using mechanisms from the dependability community is not new. What we have demonstrated is that looking at these ideas in a slightly different light, can potentially demonstrate new ways in which such mechanisms can be of use.

Although we consider this a useful starting point, the next step is to explore other facets of dependability. In doing so our intention is to reason about their use in a secure environment, which will hopefully shed light on the way in which we could use them to help us design secure systems with new means of supporting diverse security policies.

Acknowledgements

We would like to thank the various anonymous referees for their comments in helping clarify some of the discussion in this paper.

References

[AL81] T. Anderson and P.A. Lee. *Fault Tolerance: Principles and Practice*. Prentice-Hall International, 1981.

[BAN89] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, 246:233–271, 1989.

[DR86] J.E. Dobson and B. Randell. Building reliable secure computing systems out of unreliable insecure components. Technical Report 214, University of Newcastle upon Tyne, Computing Laboratory, University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, England., March 1986.

[Mea94] Catherine Meadows. The Need for a Failure Models for Security. In *Proceedings of the* 4^{th} *International Workshop Conference on Dependable Computing for Critical Applications*. Springer-Verlag, 1994.

[Mea95] Catherine Meadows. Applying the Dependability Paradigm to Computer Security. In *Proceedings of the 1995 New Security Paradigms Workshop*, 1995.

[MM99] Catherine Meadows and John McLean. Security and Dependability: Then and Now. In Paul Ammann, Bruce Barnes, Sushil Jojodia, and Edgar Sibley, editors, *Computer Security,Dependability and Assurance: from needs to solutions*, pages 166–170. IEEE Computer Society Press, 1999.

[Pan99] Susan Pancho. Parasigm Shifts in Protocol Analysis. In *Proceedings of the 1999 New Security Paradigms Workshop*, pages 70–79, September 1999.

[Rei94a] Michael K. Reiter. The rampart toolkit for building high-integrity services. In K. P. Birman, F. Mattern, and A. Schiper, editors, *International Workshop on Theory and Practice in Distributed Systems*, volume 938 of *Lecture Notes in Computer Science*, pages 99–110, September 1994.

[Rei94b] Michael K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the* 2^{nd} *A.C.M. Conference on Computer and Communications Security*, pages 68–80, November 1994.

Security and Trust in a Voter-Verifiable Voting Scheme

Jeremy Bryans and Peter Ryan
School of Computing Science, Newcastle University, UK
{Jeremy.Bryans|Peter.Ryan}@newcastle.ac.uk

Abstract

Digital voting systems provide an excellent context in which to investigate issues of security and trust in socio-technical systems. We describe the integrity and secrecy requirements of digital voting schemes, noting the tension between these requirements. We give an overview of a variant of the Chaum digital scheme [2] and use this as a vehicle to explore the nature and feasibility of these requirements. Finally we discuss how public trust in such a system might be engendered and maintained.[1]

1 Introduction

More technologically sophisticated alternatives to the traditional pen and paper methods of casting and counting votes are currently being investigated. The UK government has stated its enthusiasm for such schemes [6] and a number of trials have been performed, e.g. [3].

Such systems provide an excellent context to investigate issues of security and assurance in complex, socio-technical systems. Of particular interest to us is a scheme proposed by Chaum. In this scheme, assurance arises from voter-verification and independent auditing of the behaviour of the components: the booths and trustees. We are using this scheme as a vehicle to investigate the mechanisms for achieving assurance and for engendering public trust and confidence.

2 Requirements of a Voting System

In this section we discuss the requirements of a voting scheme. The primary requirements are accuracy and ballot secrecy. The system must also be robust and resilient in the face of accidental and malicious threats. In particular, failures with respect to the primary requirements should be detectable and recoverable.

Ideally a voting scheme should also be usable, available, efficient, unbiased and scalable but we will not discuss these aspects here. These are being addressed in the context of a broader, socio-technical study of digital voting schemes being conducted by the DIRC project, [4].

[1] The authors would like to thank David Chaum for many useful discussions. We would also like to thank the members of the DIRC project for useful discussion and input.

It is also essential the such a system gain public trust: voters must be confident that their vote will be accurately included in the final count and that the secrecy of their vote is guaranteed. One way to help engender confidence in the accuracy is to provide voter verifiability: some way for the voter to assure themselves that their vote has been accurately included in the tally. This immediately creates a tension between the requirement for ballot secrecy and that of verifiability. A naive implementation of verifiability would immediately violate secrecy. In [2], Chaum conjectures that it is impossible to achieve absolute assurances of unconditional accuracy and privacy simultaneously. His scheme provides both requirements up to certain probabilistic and computation bounds.

2.1 Accuracy

What precisely we mean by accuracy will depend on what level of abstraction we are working at and where we are drawing the system boundaries. At the most abstract level, we would like the outcome of a election, referendum etc. to accurately reflect the "intentions" of the eligible electorate. At this level we will need to consider social and psychological issues that might favour certain sectors of society, might bias choices or encourage voter error and so on. These aspects are being investigated in the DIRC project.

For the purposes of this paper we will restrict ourselves to the purely technical question of ensuring that votes counted in the final tally accurately reflect votes cast. We will assume that issues of authentication and prevention of double voting have been addressed.

In practice, absolute assurance of complete accuracy is not feasible and, arguably, too strong a requirement. The real requirement is that the outcome be "correct", e.g. that the candidate with the largest number of votes wins. The Chaum scheme provides a good probabilistic assurance of accuracy: the chance of p votes being corrupted undetected diminishes exponentially with p. As a result, tight statistical bounds can be placed on the probability of the outcome of an election being swung by undetected corruption of votes.

2.2 Ballot Secrecy

It will typically be a requirement that the way any individual voter voted remain secret. For some forms of vote this might not be required, for example voting in the UK House of Commons. For this paper we will assume that ballot secrecy is a requirement. Besides the natural desire for privacy, ballot secrecy serves to prevent coercion or vote buying. The key point is that there should be no way for a voter to prove to a third party which way they voted.

Note that absolute assurances of total secrecy may not be realistic here. In certain exceptional circumstances secrecy will be violated in any case, for example, if all the votes went one way. In any case, an observer will be able to make probabilistic estimates of which way a vote was cast on the basis of the final counts.

Instead of ballot secrecy we might require voter anonymity. At first glance one might suppose that these are equivalent. We adapt the approach of Schneider et al. [7] in formalising the notion of anonymity using CSP. A system S satisfies anonymity with respect to some set V of voters and viewpoint given by the process abstraction $\mathcal{A}$ if:

$$\forall \pi : Perm(V) \bullet \mathcal{A}(S) \equiv \mathcal{A}(S[\pi])$$

where $[\pi]$ denotes the CSP renaming operator.

Thus, if we transform the system by arbitrary permutation on the set of voters, the resulting system is indistinguishable from the original, at least from the viewpoint represented by the abstraction $\mathcal{A}$. The abstraction serves to hide internal details not visible to an outside observer. For the simplified scheme set out in this paper, an observer would only be able to see the values on the outside of each envelope.

Note that, using such a definition, the scenario of everyone voting for the same candidate would still be deemed to satisfy anonymity but would fail the ballot secrecy requirement. Given that such a scenario is perfectly admissible, this would seem to suggest that voter anonymity is the more appropriate requirement.

2.3 Assurance and Verifiability

System malfunctions and compromises will inevitably occur. It is essential therefore that mechanisms be provided to detect, contain and recover from failures. These mechanisms need to be robust in the face of malicious as well as accidental threats.

The ways of achieving assurance can be thought of as lying along a spectrum with verification at one end and run-time monitoring at the other, with testing lying somewhere in between.

In order to verify a system we assume some model of its behaviour and subject this to various forms of mathematical analysis to prove that it will satisfy certain (formally stated) requirements. This is fine up to a point but suffers from a number of deficiencies:

- Our analysis will only be as good as the models on which it is based. In order to render them tractable, our models will typically be pretty drastic abstractions of the real systems.
- It is difficult to ensure that the verified system will correspond exactly to the fielded system.
- Even supposing that our models start off as being faithful representations of the real system and its environment, systems and environments evolve. This evolution, which could include degradation, patches etc, can invalidate the original analysis.

On the other hand, run time monitoring also suffers various drawbacks:

– Monitoring can be difficult: you need to know exactly what to monitor, monitoring has to be accurate and, in a hostile environment, robust against subversion.
– Detecting a violation at run time may be too late. For example, once a secret it out, it is out.
– Some properties, by their very nature, are not amendable to run-time monitoring. Information flow properties provides a classic example: it is not a trace property and so failures of information properties typically will not be manifest from monitoring the target system.

Given these deficiencies, one should use these techniques in combination to achieve increased levels of assurance.

These issues are thrown into sharp focus in the context of digital and electronic voting systems. Many of the proposed and even deployed electronic voting systems depend heavily on (claims for) carefully verified code and provide little or nothing in the way of run-time monitoring. The Chaum scheme is at the other extreme: virtually no reliance need be placed in the components and all the assurance comes from close monitoring of the behaviour.

The Chaum scheme strives to provide the voter with good levels of assurance that their vote will be accurately recorded and that the privacy of their vote will be guaranteed. In particular, with respect to the accuracy requirement, the scheme provides the user with a physical receipt and the means to check, in principle, that their vote is accurately represented in the final count.

On the surface of it, it would appear to be impossible to devise a form of receipt that would, on the one hand allow the voter to check that their vote is accurately represented in the final tally whilst, on the other hand, not providing any evidence to a third party as to which way the vote was cast. Many commentators on the subject seem to assume that this is indeed impossible. For example, the assumption is implicit in Rebecca Mercuri's question 14 of her set of questions for evaluators of voting schemes, [5]:

"How is vote confirmation provided without ballot-face receipt?"

Most of the alternative, digital voting schemes require the voter to trust the hardware and/or software that processes their vote and provide little or no means for the voter (or indeed anyone) to detect a failure in the processing of votes.

3 A simplified variant of the Chaum voting scheme

Due to lack of space we will not attempt to describe the Chaum scheme in detail but rather we outline a simplified, analogous scheme that conveys the essence. In this sense, the paper can be thought of as an exploration of the requirements and their realisabilty. For full details of the original Chaum scheme we refer the reader to [2, 1].

The scheme presented here preserves the spirit of the Chaum original but abstracts from many of the cryptographic details, in particular the details of the use

of visual cryptography of the original (by far the most difficult aspect to explain and understand). Thus, for example, we use sealed envelopes as a metaphor for cryptographic wrappers, and a bulletin board in place of web postings. We are not proposing this as a serious, alternative scheme, the abstractions introduce several vulnerabilies, but rather as a vehicle to explore requirements and discuss trust and assurance in the context of voting schemes.

The key idea is to provide the voter with a form of receipt. Obviously, such a receipt cannot show voter's choice in the clear. The trick then is for the receipt to record the voter's choice in encrypted form. This provides a way for the voter to check that their (encrypted) vote has been correctly included in the tallying process. The problem that now remains is how to provide the voter with good assurance that their encrypted receipt will be faithfully decrypted to their actual choice.

Suppose that there are n voters and k trustees. Suppose further that the voter is asked to choose from amongst s options. The voter's choice can therefore be encoded as a number in the set $\{0, \cdots, s-1\}$.

The task of the voting machine, when the voter enters the booth, is to assign a unique serial number q to the voter and prepare $2k$ random, independent numbers drawn from the set $\{0, \cdots, s-1\}$. Denote these by $B_{q,i}$ and $G_{q.i}$, where i is the trustee index. These numbers will serve to (super)-encipher the vote. The booth now prepares two nested envelopes, one Blue and one Green. Consider the Blue envelope (the Green one is similar): it comprises k nested envelopes. The inner envelope contains a slip bearing the value $B_{q,k}$. The next envelope contains the inner envelope along with another slip bearing the value $B_{q,k-1}$, and so on until the outer envelope that contains all the other, nested envelopes and a final slip bearing the value $B_{q,1}$. The outer envelope also shows the unique serial number q.

The voter now provides their choice of candidate, via a touch screen for example. Suppose that this is encoded as the numerical value $Vote_q$. The booth now computes two receipt values as follows:

$$Receipt_{q,Blue} = \left(Vote_q + \sum_{i=0}^{s-1} B_{q,i} \right) \text{mod s}$$

This value is printed a blue receipt slip. Similarly for the Green receipt.

The voter is now invited to choose between the blue and green receipts. Suppose that she chooses to retain the blue. In this case the green envelopes are opened, and she calculates the sum (mod s) of the values on the enclosed slips. She then checks that when she subtracts this (mod s) from the value shown on the green receipt she gets the numerical code for her choice of candidate. The green receipt, envelopes and slips are now destroyed in front of her, she retains the blue receipt and sees the blue envelope (with $Receipt_{q,Blue}$ printed on the outside) dropped into a locked sealed ballot box.

Note that all these envelopes could be prepared in advance, possibly by some other authority and printed on counterfeit-proof stationery.

3.1 The Role of the Trustees

Once all the votes have been cast, the ballot box is passed on to the first trustee, who gathers all the envelopes and shuffles them throughly. Then for each envelope, he does the following:

- He breaks the seal and opens the envelope.
- He subtracts the value on the enclosed slip from the receipt value on the printed on the outside of the envelope.
- He prints the resulting number on the enclosed envelope along with a new counter number q_1.
- He retains the enclosed code sheet and records on it the counter number of the enclosing envelope (q, in the case of the outermost envelope) as well as the new, scrambled number q_1. This is for checking purposes later.

Having done all this he posts all the outer (now empty) envelopes back to the first column of the bulletin board and the enclosed envelopes are passed on to the second trustee.

The second trustee now repeats this process on the envelopes it receives from the first trustee. This continues until we reach the last of the trustees. She will open the inner envelopes to reveal the final encryption value which she duly subtracts from the value of the inner envelope which, to reveal the numerical code for the voter's original choice. These are posted on the last column of the bulletin board in scrambled order according to another new counter number.

The final column will now have the full set of correctly decrypted votes, but in (multiply) scrambled order with respect to the order in which the original, sealed envelopes were posted. Thus we should have accuracy: the set of votes in the last column accurately reflect the votes cast, and anonymity: decrypted votes cannot be linked back to the voters.

3.2 Checks on the Booths and Trustees

The fact that the voters can check that the outer envelopes with the correct serial number and encrypted vote appear in the first column of the bulletin board serves as a check on the booths. The seals on the envelopes make it difficult for the booths to mess about with their contents. The voter's free choice between the two envelopes and ability to check the correct decryption of the receipt/envelope pair that is then destroyed means that any attempt by the booth to cheat by incorrectly encrypting the receipt stands a 50/50 chance of being caught.

To check on the behaviour of the trustees the following procedure is followed: for the first trustee, an auditor randomly designates half of the ballots to be checked. For these, the first trustee is required to reveal the corresponding decryption slip. This indicates the number assigned to the enclosed envelope posted to the second column of the bulletin board and the encryption value. The auditor is thus able to check that the decrypted value the trustee printed on the enclosed envelope bears the correct relationship to the value on the first envelope.

The auditor now requires the second trustee to reveal the "complementary" links as created by the second trustee. Thus none of the links of the first trustee should line up with links revealed by the second trustee.

The auditor now performs the corresponding checks on the links revealed by the second trustee, and so on through all the trustees. Thus any attempt by a trustee to perform the decryption incorrectly on any ballot will stand a 50/50 chance of being detected by the auditor.

The decrypted votes will all be available in the final column output of the last trustee and so the overall count will be checkable by anyone.

4 The trust model

From the point of view of the accuracy requirement, the trust that we need to place in the booths and trustees is minimal since the scheme has been designed to detect any significant attempt at fraud.

Some degree of trust is required in the auditors who perform the partial random checks on the trustee transformations. This can be minimised as necessary by using a number of independent auditors. One place where a higher level of trust is required is in the process for the selection of the links to be revealed. This is one step in the process that cannot be replicated.

Assurances of anonymity are provided by the secret, random permutations on the envelopes applied by the trustees. If there are k trustees then, in the original scheme, we can tolerate the subversion of $k-1$ of these and still provide anonymity.

We assume that the checks are performed sufficiently assiduously, i.e. a sufficient proportion of the voters do choice to run checks on their receipts and check that their receipt is correctly posted on the web site. We also assume, for this paper at least, that the auditors perform their checks correctly and that there is no collusion between trustees and auditors. Note that if a trustee knew in advance which links would be selected for audit then they could corrupt votes on the other links.

5 Public trust

The Chaum scheme has been carefully devised, and the checks carefully constructed, so that essentially no trust need be placed in the components of the system. Thus, with respect to the accuracy requirement, it would be extremely difficult for either a booth or a trustee to corrupt of falsify votes undetected. The scheme thus offers a high degree of transparency: significant malfunctions or compromises of booths or trustees will be detected, as long as the checking procedures are applied reasonably assiduously.

On the other hand, careful verification of the design and validation of the assumptions is required.

Being trustworthy, however, is not the same as being trusted. The voter's ability to check that their (encrypted) ballot is included in the tally should help

foster used confidence. On the other hand, the subtlety and complexity of the arguments demonstrating that ballot receipts will be correctly decrypted by the trustees will be difficult for most voters to understand. This could prove problematic from the point of view of public uptake. It is extremely difficult to provide a simple explanation of the grounds for regarding the scheme as trustworthy that the average voter would find both understandable and convincing.

The majority of the public therefore would have to take on trust the claims for such a scheme. Presumably having the system reviewed by a number of authorities and experts would help engender public confidence, but would this suffice? Could confidence be undermined by spurious claims to be able to read the receipts?

6 Conclusions

We have discussed the dependability requirements of digital voting systems and have presented the elements of the Chaum scheme. We have argued that the scheme appears to meet these requirements to a high level of assurance and requires essentially no trust to be placed in the components. It offers a high level of transparency and allows voter-verifiability and for all the steps to be independently audited.

We have briefly discussed the obstacles to engendering and maintaining public trust in such a system. In future work we will investigate further the socio-technical aspects of this and similar digital voting schemes.

References

1. Jeremy Bryans and Peter Ryan. A Dependability Analysis of the Chaum Voting Scheme. Technical Report CS-TR-809, Newcastle University School of Computing Science, 2003.
2. David Chaum. Secret-Ballot Receipts and Transparent Integrity: Better and less-costly electronic voting at polling places. http://www.vreceipt.com/article.pdf.
3. The Electoral Commission. Modernising Elections: A Strategic Evaluation of the 2002 Pilot Schemes. http://www.electoralcommission.gov.uk/about-us/modernisingelections.cfm, Oct 2002.
4. Interdisciplinary Research Collaboration in Dependability. http://www.dirc.org.uk.
5. Rebecca Mercuri. Questions for voting systesm vendors. http://www.notablesoftware.com/checklists.html.
6. Office of the E-envoy. http://www.edemocracy.gov.uk, July 2002.
7. Steve Schneider and Abraham Sidiropoulos. CSP and Anonymity. In *ESORICS*, volume 1146 of *LNCS*, 1996.

Anonymity with Identity Escrow

Lindsay Marshall and Carlos Molina-Jiminez

School of Computing Science, University of Newcastle, Newcastle upon Tyne, UK
Lindsay.Marshall@ncl.ac.uk, Carlos.Molina@ncl.ac.uk

Abstract. The ability to have a conversation with someone whilst remaining anonymous is, in some circumstances, an important, arguably essential, right that people expect and ought to have. With electronic communication it becomes harder to safeguard this right whilst paying heed to the legitimate objections that raised to anonymous communication. This paper describes a system using identity escrow where people can create anonymous identities which can be used for communication but where their true identity can be recovered if there is a reason to do so.

Introduction

In the present political climate, there is much talk of the dangers to society from the use of encrypted and/or anonymous communication by criminal. Many of the arguments put forward are without merit, nevertheless, there are arguments, particularly with respect to anonymity, which need to be considered carefully: people should be able to expect to be protected from anonymous harassment. However, this does not mean there are no circumstances where preserving a someone's anonymity is acceptable. There are many situations where we expect and get anonymity (e.g. personal help lines).

As electronic media are increasingly the way that people communicate it is necessary that mechanisms for providing anonymous communication using them should be investigated . In this paper we describe a method of establishing an anonymous electronic identity where in well-defined circumstances the true identity of the user can be revealed.

Total Anonymity

[4] describes a protocol (MMP) that allows users of wireless computers to access communication services without revealing their identity. Excluding attacks that would require a "funded organisation" [1, 2], MMP allows a user to send messages where the recipient cannot identify the sender unless this information is part of the message.[1]

MMP provides for one-way anonymous communication, and if the two parties establish, say, a TCP connection between them, they can exchange messages with the

[1] The details of this protocol are not relevant to the subsequent discussion in this paper.

initiator remaining anonymous. However, this kind of communication leaves itself open to "call tracing" attacks. The best way of overcoming this is to allow the exchange of messages between named but anonymous users. This was the service provided by penet.fi, but its weakness was that there was a centrally held mapping between real and anonymised identities, and it was the existence of this that lead to its closure. This mapping let the penet.fi server be a conduit for anonymised messages, which is efficient but weakened the reassurances of anonymity that it provided.

With MMP, it is possible to have totally anonymous bi-directional communication. The simplest method would be for someone to offer a service which allows users to register an identity which they can use on their MMP connection and which provides a range of services. However as total anonymity appears to be unacceptable to many people, we propose a service that supports anonymous users but where the mapping between real and anonymised identities can be reversed easily under controlled circumstances. This can be achieved by the use of an escrowed identity scheme. Previous uses of escrowed identity (e.g. [3]) have concentrated mainly on its use for group signatures rather than for the provision of a general anonymising service.

Escrowed Identity

Users wishing to use the service anonymously must first sign up and create an identity within the system. This relies on the availability of an MMP-style communication channel as without this the act of creating an anonymous identifier is open to attacks that can allow the identity to be linked to the creator.

Rôles, Assumptions and Notation

We assume the following players in what follows:

A	Alice, someone who wishes to send anonymous messages
$\overline{A}$	Alice when using anonymous communication so that her identity is not revealed.
S	A provider of an anonymous interactive service
n,m	Constants defined by S for use with Rabin's algorithm [5] (or a similar method) which allows the segmentation of a data block into n parts but allows its recovery from m *blocks,* where $n > m$ and $n,m > 0$
T	$\{T_i : i \in 1 \ldots q\}$ is a set of identity token providers, $q > 0$
E	$\{E_i : i \in 1 \ldots p\}$ is a set of escrow holders, $p > n$
H	$H \subseteq E, \lvert H \rvert = n$ is a set of escrow holders chosen by A from E
V	$\{V_i : i \in 1 \ldots n\}$ is a set of votes. The exact nature of a vote is discussed below

Set T is defined by S and consists of well-known, independent organizations (for example, banks) that have agreed to perform identity token generation for S. Set E is also defined by S and consists of people (or organizations) who have agreed to "hold"

components of escrowed identities and to take part in votes concerning the uncovering of specific identities.

Issues of trust are clearly important and we shall discuss these further below. For the purposes of the protocol description we assume that everyone involved is trusted not to attempt to comprise the anonymity of A. Some parts of the protocol as described are not necessary in a system where everyone is trusted; we include them as they serve as a guarantee that certain kinds of attacks are not possible

We use the following notations :

$X \prod Y : M$	X sends Y the message M over normal untrusted channels
$X \prod \text{“mmm”} \prod Y : M$	X sends Y the message M over an anonymous channel using MMP
$\{D_1, D_2, ...\}_K$	Encrypt/Decrypt the data items $D_1, D_2, ...$ using the key K
$sign_X(D)$	Sign D as being from X
$valid_X(D)$	Validate that X signed D, returns True or False
$R(n, m, D)$	Segment D using Rabin's algorithm into *n* segments, recoverable using *m* segments
$store_X(D_1, D_2, ...)$	X stores the data $D_1, D_2, ...$
$\square m_i$	The concatenation of series of messages m_i
$\square_A$	Alice's token signed by Identity Token Providers (ITKP)
L	A unique, anonymous identifier

In what follows we show encryptions, but usually omit the symmetric decryptions.

Signup protocol

Alice first sends a signed request for an identity token (*ITKRequest*) to her chosen provider. Assuming that the request is correctly signed and that the token provider is prepared to validate it, then she receives a signed token readable only by the provider. For simplicity, step 1 indicates that the identity string encrypted is Alice's public key, see below for discussion of the possible contents of this string.

In a real system it would be necessary to prevent the reuse of identity tokens, and this can be achieved by the use of timestamp, etc, however we have omitted this step from the protocol description

1. $A \square T_i : \{sign_A(ITK\,\text{Re}\,quest)\}_{K^{T_i}_{pub}}$

2. $T_i \rightarrow A : \left\{ sign_{T_i} \left\{ K^A_{pub} \right\}_{K^{T_i}_{pub}} \right\}_{K^A_{pub}}$

At this point, Alice can stop and proceed with $\Phi_1 \equiv \Phi_A$, however she may suspect that her chosen provider is in collusion with the ITKP and might be able to find her identity. Steps 3 and 4 (which can be repeated as often as wished) ask another token provider to sign the token after encrypting it (*ITKSignatureRequest*). This process is carried out over an anonymous connection as Alice's real identity is not required as this is implied by the validity of the token.

3. $\overline{A} \xrightarrow{anon} T_j : \left\{ ITKSignatureRequest, \Phi_1 \right\}_{K^{T_j}_{pub}}$

4. $valid_{T_i}(\Phi_1), then\ T_j \xrightarrow{anon} \overline{A} : \left\{ sign_{T_j} \left(\left\{ \Phi_1 \right\}_{K^{T_j}_{pub}} \right) \right\}_{K^{\overline{A}}_{pub}}$

$\Phi_1 = sign_{T_i} \left\{ K^A_{pub} \right\}_{K^{T_i}_{pub}}$

$\Phi_A = \Phi_m = sign_{T_x} \left(\left\{ \Phi_{m-1} \right\}_{K^{T_x}_{pub}} \right)$

Alice now generates a new key pair, establishes an MMP connection to S, and asks for provision of anonymous service (*ServiceRequest*), sending her temporary public key, her identity token and a password to be used with the new identity.

5. $\overline{A} \xrightarrow{anon} S : \left\{ ServiceRequest, K^{\overline{A}}_{pub}, \Phi_A, password \right\}_{K^S_{pub}}$

S validates the token and returns a list of escrow holders and the number *n* of holders she must pick.

6. $valid_{T_i}(\Phi), then\ \ S \xrightarrow{anon} \overline{A} : \left\{ n, E \right\}_{K^{\overline{A}}_{pub}}$

Alice decides which *n* escrow holders to trust and returns this set to S.

7. $\overline{A} \xrightarrow{anon} S : \left\{ H \right\}_{K^S_{pub}}$

S then encrypts Alice's token and partitions the result into *n* segments using Rabin's algorithm (step 8). The encryption is necessary to show that there is no possibility of collusion between escrow holders during complaint resolution.

8. $C = \left\{ C_i : i \in 1 : n, R\left(n, m, \left\{ \Phi_A \right\}_{K^S_{pub}} \right) \right\}$

Each of these segments is encrypted with the public key of the corresponding escrow holder selected by Alice (step 10). S generates L, stores L, H and the encrypted

segments (step 11) and then returns L to Alice who can now use this as her identity when using the services S provides.

9. $D = \left\{ D_i : i \in 1:n, \{C_i\}_{K^{H_i}_{pub}} \right\}$
10. $store_s(L, H, D)$
11. $S \xrightarrow{anon} \overline{A} : \{L\}_{K^{\overline{A}}_{pub}}$

Note that the escrow holders are not involved in the process of setting up an anonymous identity, nor are they actually required to hold any data

Complaint Resolution

If S receives a complaint Ω about anonymous user L, then it retrieves the triple (L, H, D) stored in step 10. S sends a copy of the complaint (*Adjudicate Request*) to each of the escrow holders nominated by L (step 1), that is, the members of H.

1. $\forall H_i \in H, S \rightarrow H_i : \{Adjudicate\,\mathrm{Re}quest, \Omega\}_{K^{H_i}_{pub}}$

Each replies with a signed vote. These consist of packets signed by the escrow holders containing a ballot and the complaint Ω. A ballot can be a simple yes or no vote, or it could be a numerical indicator of support, say a number in the range -1…1. The inclusion of the complaint in the vote prevents attacks based on reusing votes.

2. $H_i \rightarrow S : \{sign_{H_i}(V_i)\}_{K^S_{pub}}$

S sends the set of all the votes to each of the escrow holders along with *m*, the list of voters, and the segment of Ω that was encrypted using their key.

3. $\forall H_i \in H, S \rightarrow H_i : \{V, m, H, D_i\}_{K^{H_i}_{pub}}$

The escrow holders can read all the votes and verify that *m* or more holders have voted "yes" (or that the sum of their ballots > *m*) and if so, decrypts the identity segment and returns this value to S

4. $H_i \rightarrow S : \left\{ \{D_i\}_{K^{H_i}_{priv}} \right\}_{K^S_{pub}}$

S concatenates these segments and uses Rabin's algorithm to retrieve the encrypted identity token.

5. $I = \left(R^{-1}(n, m, \prod_{i=1}^{m} D_i) \right)_{K^S_{priv}}$

S then decrypts the token, derives the identity provider from it and asks (*Reveal*) for the real identity associated with the token. S sends the details of the complaint, the results of the voting and the identities of the escrow holders. Since the token may

have been blinded, this step may pass through several token providers until it reaches the original creator.

6. $S \rightarrow T_i : \{\text{Reveal}, I, \square, H, V\}_{K_{pub}^{T_i}}$

The original token provider replies “Alice”, that is, the real identity of the user with login L.

7. $T_i \rightarrow S : \{\text{"Alice"}\}_{K_{pub}^{S}}$

Simpler versions of the voting protocol are possible, but these do not exclude certain attacks. For instance, an escrow holder could vote “yes” by returning their decrypted segment and “no” by doing nothing, however this would let S save decrypted segments from a series of failed complaints until it has *m* different blocks and then regenerate the token. With a simple yes/no ballot system, escrow holders who vote “no” do not need to decrypt their segment, however with weighted ballot systems a more complex system of agreement may need to be used.

Identity Tokens

A trusted identity supplier generates an identity token for someone based on unique data, for example, passport number. The contents of the token are not important so long as they can be uniquely tied to the entity and are not easily forgeable or stealable. We assume that the ITKP has knowledge of the person and that it takes all possible steps to prevent fraud. Typically these suppliers would be banks, credit card companies or government departments. The identity data is encrypted and signed by the supplier. This lets anyone check a token claimed to have been generated by an ITKP.

Escrow Holders

An escrow holder is trusted by both the ASP and customer to provide a fair and balanced adjudication for complaints. Escrow holders whose public keys are known to the ASP need only be actively involved when a complaint arises. They vote on whether a complaint warrants the identity of the user being revealed, and then decrypt a string if the vote yields *m* “yes” votes out of *n*. Escrow holders are sent full details of the voting so that they can verify that the target of *m* “yes” votes was reached. The use of Rabin’s algorithm ensures that escrow holders who vote “no” do not need to reveal their segment of the token.

Difficulties and Attacks

Because trust is involved, there is no guarantee that the identity of a user cannot be retrieved without going through the escrow process. The aim has to be to reduce the trust requirements to a minimum and to identify the parts of the process where trust

could be compromised. Let us look at the trust requirements for each of the participants in the system.

Escrow Holders Escrow holders never see plaintext data so there is no data manipulation that they can do individually to find the identity of an anonymous user. A group could collude to complain about a user and agree to vote to reveal the identity. However, as the they are chosen for being independent and unbiased this ought to be an unlikely.

Users We assume that users of the system have not been the victims of identity theft. That is, when they present their identity token, they really are who the token represents. This is always going to be a difficult area in anonymous systems and cannot be avoided. The best we can do is to ensure that tokens have a limited life span and that users must re-authenticate themselves at regular intervals.

The Anonymous Server Provider (ASP) If the ASP colludes with other parties then an identity is not safe. This unavoidable, but suspicion of collusion would damage the reputation of the ASP and so is unlikely. However, employees could collude to steal data in order to retrieve an identity. There is only one point in the protocol where this is a danger as once the token has been split it is encrypted using escrow holder's keys. This happens when users applying for an anonymous id send their token to the ASP. To check that the token is valid it must be in a verifiable form, and since it is also the identifier that is presented to an ITKP when asking for an identity to be revealed, the ASP's system must ensure that the open token is not stored. A stolen token allows the theft of an anonymous identity, and there needs to be a way to warn of this, similar to key revocation systems.

Identity Providers We trust that ITKPs generate tokens backed up by proper identity checks and that they secure their data. ITKPs must be seen to be trustworthy or else their business will suffer. If an ITKP, for whatever reason, ceases to operate then all tokens that they signed must be revoked as they can no longer be used to reveal a valid identity. Users whose tokens become invalid must reauthorise by presenting a new token and their old token.

Attacks

In addition to the attacks based on collusion there are other possibilities, for instance:

Bogus complaints We have not addressed the issue of how compliants are made and deemed to warrant being voted on. A complaint that seemed to be valid would encourage escrow holders to vote to uncover an identity, but generating the cause for such a complaint may be hard, unless this is based on material alleged to be defamatory that is in fact true. Using the law to force revelation is difficult as the

service provider does not have a reversible identity mapping, so lawyers would have to apply pressure to everyone involved in the process which may be impractical.

Overloading the third party By generating enough complaints you might be able to overload the escrow holders so that they vote "yes" all the time. Equally they might vote "no", so this may not be a useful attack! Overloaded escrow holders may decide not to take part in the process in which case all identities that they escrow must be held until they have been re-established. A user could take advantage of delays in the resignation process to cause complaints that cannot then be resolved.

Identity theft The system is always vulnerable to people who have managed to obtain valid credentials for another person. If the ITKPs are properly managed then this risk can be reduced, though coercion can always be used to make someone give up their identity. Identity theft is used as a way of avoiding being discovered or of causing trouble to another person so is different from the attacks that could be used to try to reveal an identity. The theft of an existing anonymous identity is also a possibility and has exactly the same problems as the theft of an ordinary login identifier.

Conclusions

The aim of our protocol is to make anonymity practical by making it reliable for the anonymous user and safe for the public: the protocol guarantees that the identities of anonymous users remains concealed unless they abuse a service. The protocol is applicable in circumstances that often arise and provides a workable solution for them. Our next step is to build a variety of systems that embody the protocol and which provide such services as surveys or anonymous business collaboration. A more detailed version of this paper is available from the authors.

Acknowledgements

This work is part-funded by European Union Project IST-2001-34069 "TAPAS". Thanks to Peter Ryan, Jeremy Bryans, Santosh Shrivastava, Dick Snow and Cliff Jones for their helpful comments and suggestions.

References

1. D.G. Abraham, G.M. Dolan, G.P. Double and J.V. Stevens "Transaction Security System", in IBM Systems Journal, v30 no. 2 (1991), pp. 206-229
2. Ross Anderson "*Security Engineering: A Guide to Building Dependable Distributed Systems*" John Wiley & Sons, 2001
3. Jan Camenisch and Anna Lysyanskaya "*An identity escrow scheme with appointed verifiers*." Advances in Cryptology -- Crypto 2001, Lecture Notes in Computer Science

4. Carlos Molina-Jiminez and Lindsay Marshall "*Anonymous and Confidential Communications from an IP Addressless Computer*" In Proceedings of the International Symposium on Handheld and Ubiquitous Computing (HUC '99), Karlsruhe, Germany, 27-29 September 1999, Gellersen, H.-W. (ed) Lecture Notes in Computer Science, Volume: 1707, pp. 383-385 Springer-Verlag, 1999
5. Michael O. Rabin "*Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance*" Journal of the Association for Computing Machinery. Vol. 36, No. 2, April 1989, pp. 335-348.

Defining Authentication in a Trace Model

C.J.F. Cremers[1], S. Mauw[1], and E.P. de Vink[1,2]

[1] Eindhoven University of Technology
Department of Mathematics and Computer Science
P.O. Box 513, NL-5600 MB Eindhoven, the Netherlands
{ccremers,sjouke,evink}@win.tue.nl
[2] LIACS, Leiden University, the Netherlands

Abstract. In this paper we define a general trace model for security protocols which allows to reason about various formal definitions of authentication. In the model, we define a strong form of authentication which we call *synchronization*. We present both an injective and a non-injective version. We relate synchronization to a formulation of agreement in our trace model and contribute to the discussion on intensional vs. extensional specifications.

1 Introduction

Security protocols have become an established application area of formal methods today. Over the past years various modelling languages, logics and process algebras have been proposed for the systematic and tool supported analysis of security protocols. Exploitation of formal methods in this setting leads to a better understanding of implicit assumptions and gives feedback on what the protocol does or, in many cases, what it unexpectedly does not achieve.

In this paper we focus on the notion of authentication as a so-called intensional security property. We formulate a variant of authentication, called synchronization. In [10] Roscoe introduces the notion of canonical intensional specification, stating that all parties involved in a protocol, after completion of their role, are convinced that for their part the protocol has been executed according to its rules. Intensional properties are those induced by the form or structure of the protocol; extensional properties are related to the effect the protocol achieves. The notion of synchronization proposed here captures Roscoe's notion of canonical intensional specification as a general trace property. By casting intensionality in the same framework as extensional security properties, intensionality can be compared with and, as we shall see, related to so-called full injective agreement.

Important to this point of view is the observed behavior from the perspective of an individual agent. It will make no difference to an agent engaged in the protocol if a sequence of communications was the result of its interaction with an honest principal, with a malicious intruder controlling the network, or a mix of these. If the agent is entitled to believe at a point in time that the interaction thus

far is consistent with the agents role in the protocol, the agent simply proceeds under the assumption that all parties involved have obeyed the protocol rules.

In [8] Lowe studies, building on earlier work of [1] and [4], four different forms of authentication, viz. aliveness, weak agreement, full non-injective agreement and full injective agreement. On top of this, agreement on subsets of data items and recentness is considered (two topics we do not address here). By casting the notions in the setting of CSP Lowe shows that the notions constitute an ascending chain of authentication principles. Additionally, the relationship with Roscoe's definition of intensional specification is discussed at the conceptual level. The notion of injective synchronization that is proposed in this paper as intensional authentication can be shown to be strictly stronger than full injective agreement and thus, can be added at the top of Lowe's authentication hierarchy. This extension constitutes the body of the present paper.

Authentication and agreement is also studied in [3] by Focardi and Martinelli in the context of the so-called GNDC scheme. In a process algebra extended with inference systems reflecting cryptographic actions, one can reduce reasoning about security properties with respect to any arbitrary environment to the analysis of the behavior of the security protocol in the most general environment. It is argued that the GNDC scheme is valid for establishing various security properties, in particular agreement (as well as its weaker variants).

In Section 2 below we gather the machinery required for our description of security protocols. The notions of non-injective and injective synchronization are the main topic of Section 3. The thread of reasoning is continued for the case of agreement. In Section 4 agreement is reformulated in our setting and compared with its definition in [8]. We further present a taxonomy result for the introduced notions of injective and non-injective synchronization and agreement, and provide examples to show the strictness of the implied relationships. Section 5 then closes off with concluding remarks.

Acknowledgment We are grateful to Jerry den Hartog for his useful suggestions at various points.

2 Trace model

In this section we will define the notions which are essential for the definition of synchronisation as elaborated in Section 3. Synchronisation is based on the property that every successful execution of a protocol by an agent implies that its communication partners exactly follow their roles in the protocol and exchange the intended message in the intended order. Therefore, we first provide a formal definition of a security protocol and the partial order implied on its events.

The second step is the introduction of the trace model for asynchronously communicating agents. Synchronisation is verified as a property on the set of execution traces. It can be defined independently of the way in which the trace model is constructed for a given security protocol, a given set of agents and a given intruder model. Therefore, we will not find a need to introduce a formal semantics precisely defining the trace model. Rather, we expect that many of

the existing formal approaches towards security protocols can be molded in such a way as to produce the required trace model.

2.1 Security protocols

A security protocol defines the interacting behaviour of a number of agents. Agents can take part in one or more runs of a protocol by performing a role defined in the protocol.

In most approaches, a protocol description consists of a list of messages exchanged by some principals. In contrast to this approach we take as a starting point the projection of this behaviour onto the different roles. Thus, we split a communication into two separate events, viz. a send event and the corresponding read event. These two corresponding events are described in two (different) role definitions. We elaborate on the implications of this choice later.

We assume a finite set of role names, called *Role*. The behaviour of a role is defined by a role definition (*RoleDef*). A role definition is simply a list of role events (taken from the set *RoleEvent*). In the following, we will define send, read, and claim events.

Roles exchange message from the set *RoleMess* of role messages, by executing the events $send(r, r', m)$ and $read(r, r', m)$. The send event is executed by role r, which intends to send message m to role r'. The read event is executed by role r' and indicates the reception of message m, apparently sent by role r.

Because it is allowed that the same message occurs more than once in a role definition (having the same sender and the same recipient), we will need to disambiguate these events by extending them with labels from the finite label set *Label*. Therefore, we have role events $send_\ell(r, r', m)$ and $read_\ell(r, r', m)$ for a label $\ell \in Label$. We require that all events in a security protocol have distinct labels, except for corresponding send and read events (from different roles), which share a common label.

These labels also serve a second purpose. Due to our decision to split communications into separate send and read events, we lost the intended correspondence between these events. We use the event labels to keep this correspondence information available. If a send event from one role definition shares a label with a read event from another role definition this expresses that these are corresponding events. We define the (partial) functions *sendrole* and *readrole* to determine for a given label the sending role and the receiving role, respectively.

Our treatment of security claims is somewhat different from other approaches. Rather than considering correctness of a security protocol as a property of the protocol as a whole, we consider a more refined approach where claims are local to each role. For this purpose, we introduce claim events.

A claim event has the form $claim(r, c)$, where r is the claiming role and c is the claim, taken from a set of claims *Claim*. An example of such an event is $claim(r, alive(r'))$. This means that if an agent executing role r has executed his part of the protocol up to the claim event, he can be sure that the agent executing role r' was alive. In Sections 3 and 4 we will give the definitions of four claims: *ni-synch*, *i-synch*, *ni-agree* and *i-agree*.

The fact that security claims are local to a role is a consequence of our approach to split communication events into separate send and read events and to consider role definitions as a basic concept.

Finally, we define a security protocol $p \in \mathit{SecProt}$ as a collection of role definitions, or rather as a function from Role to $\mathit{RoleDef}$. In summary, we have the following definitions.

$$\begin{aligned}
\mathit{SecProt} &= \mathit{Role} \to \mathit{RoleDef} \\
\mathit{RoleDef} &= \mathit{RoleEvent}^* \\
\mathit{RoleEvent} &\supseteq \{\mathit{send}_\ell(r, r', m), \mathit{read}_\ell(r, r', m), \mathit{claim}_\ell(r, c) \mid \\
&\qquad \ell \in \mathit{Label}, r, r' \in \mathit{Role}, m \in \mathit{RoleMess}, c \in \mathit{Claim}\} \\
\mathit{sendrole}(\ell) &= r \quad \text{if } \mathit{send}_\ell(r, r', m) \in p(r) \text{ or } \mathit{read}_\ell(r, r', m) \in p(r') \\
\mathit{readrole}(\ell) &= r' \quad \text{if } \mathit{send}_\ell(r, r', m) \in p(r) \text{ or } \mathit{read}_\ell(r, r', m) \in p(r')
\end{aligned}$$

Please notice that we did not provide an exhaustive definition of $\mathit{RoleEvent}$. We only require inclusion of the given events. The reason is that these events suffice for our definition of authentication. Nevertheless, a formal semantics could consider more role events, such as assignments to local variables.

The authentication property which will be defined in Section 3 expresses that the message exchanges between the agents take place exactly in the order as implied by the security protocol. In order to express this, we define the causality relation on the events in a security protocol as a partial order. Since security protocols are often visualised as Message Sequence Charts (MSCs, see [5]), it comes as no surprise that the partial order semantics for MSCs can be used to express the order of events in a security protocol. This partial order simply expresses that the events in a role definition are sequentially ordered, and that every read event is preceded by its corresponding send event. Adopting notation from [2], we start with defining the role order $\prec_r$ for $r \in \mathit{Role}$ as the total order on its events. Event e causally precedes event e' (notation $e \prec_r e'$) if e occurs before e' in the role definition of r. Next, we define the send-before-read order $\prec^{sr}$ which expresses that a send causally precedes the corresponding read (i.e. $\mathit{send}_\ell \prec^{sr} \mathit{read}_\ell$). Finally, we define the causality preorder $\prec_p$ of protocol p as the transitive closure (denoted by $^+$) of the union of all role orders and the send-before-read order, i.e.

$$\prec_p = (\bigcup_{r \in \mathit{Role}} \prec_r \cup \prec^{sr})^+$$

Example The well-known Needham-Schroeder-Lowe protocol (NSL) (cf. [9, 6]) can be depicted by the MSC in Figure 1. The added *i-synch* claims at the end of the roles will be defined in Section 3.

The NSL protocol has two roles, viz. one of the initiator I and one for the responder R. The role definitions for the initiator and responder are

$$\begin{aligned}
&\mathit{send}_1(I, R, \{I, n_I\}_{pk_R}) \cdot \mathit{read}_2(R, I, \{R, n_I, n_R\}_{pk_I}) \cdot \mathit{send}_3(I, R, \{n_R\}_{pk_R}) \cdot \mathit{claim}_4(I, \textit{i-synch}) \\
&\mathit{read}_1(I, R, \{I, n_I\}_{pk_R}) \cdot \mathit{send}_2(R, I, \{R, n_I, n_R\}_{pk_I}) \cdot \mathit{read}_3(I, R, \{n_R\}_{pk_R}) \cdot \mathit{claim}_5(R, \textit{i-synch})
\end{aligned}$$

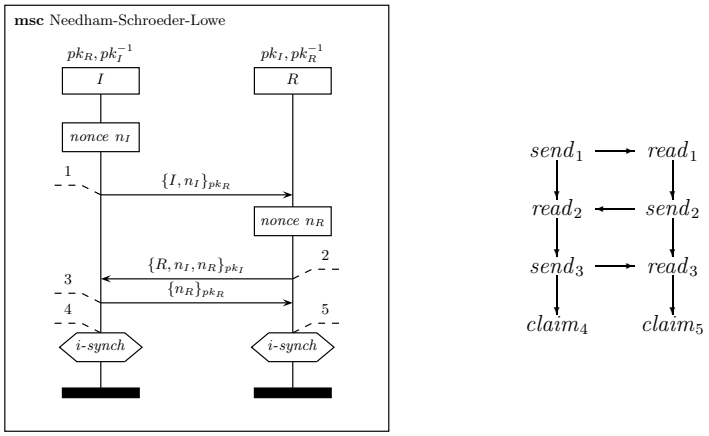

Fig. 1. The NSL protocol and the partial ordering on its events.

The indices of sends, reads and claims refer to the labels listed aside of the roles. On top of the roles, the initial knowledge of the agents is listed. It is assumed that both agents know the public key of the other and the private key of their own. The generation of nonces n_I and n_R is visualized in the *action boxes*. The hexagons contain the claims made by the agents. In fact, these are the security goals the protocol is supposed to achieve. The causality preorder $\prec_p$ for NSL is given by the lattice on the right in Figure 1.

2.2 Traces

Now that we have introduced security protocols, we can discuss the execution of a security protocol (or rather a role in a security protocol) by an agent. We start by defining the set of agents *Agent*. A role executed by an agent is called a run. Whereas we can consider a run as an instantiated role, we can consider run events (*RunEvent*) as instantiated role events. This instantiation simply amounts to replacing abstract role names by concrete agent names. Since the concrete messages sent in a run may be different from the abstract messages specified in a role (e.g. because of the instantiation of role names by agent names), we introduce the set of run messages *RunMess*. An agent can execute several roles (possibly the same) in an interleaved way. Although it is not necessary for the definitions below, a proper semantics will require that every (honest) agent executes the events exactly as prescribed by its role definition.

Because we need to be able to distinguish the events from different runs (possibly stemming from the same role), we need to add information to disambiguate run events. This is done by assigning a unique run identifier from the set *RunId* to every run and extending the events of every run with their run identifier. An event e extended with run identifier rid is denoted by $e\sharp rid$. Thus, we have the

following run events.

$$RunEvent \supseteq \{send_\ell(a, a', m)\sharp rid, read_\ell(a, a', m)\sharp rid, claim_\ell(a, c)\sharp rid \mid rid \in RunId, \ell \in Label, a, a' \in Agent, m \in RunMess, c \in Claim\}$$

Each of these events is executed by some agent, performing some role in the protocol. The role of the actor of event e is denoted by $role(e)$. It can be derived easily from the label of the event (which is unique in the protocol specification) taking into account if it is a send or read event, or a non-communication event.

The final step is to introduce the traces of a security protocol, induced by some semantic definition (e.g. assuming operational behaviour of agents and an intruder model). The semantics of security protocol p is denoted by $T(p)$. We assume that it is defined as a collection of sequences of run events.

$$T(p) \in \mathcal{P}(RunEvent^{\leq\omega})$$

We denote the i-th event of trace $\alpha \in T(p)$ by α_i (for $i \geq 0$). If we require a property of a security protocol p, we will, in our set-up, have to check this property for all of the traces in $T(p)$.

Thus, the semantics of a security protocol is expressed as a set of traces and each such trace is an interleaving of a number of runs. An agent can execute many such runs in an interleaved way and every run may be an instantiation of each of the roles in the protocol definition. We assume $T(p)$ to contain, for example, traces corresponding to the double instantiation of p where agent a is running both role r and r', and agents b and c are running roles r' and r, respectively. But $T(p)$ will also contain traces for the situation where role r is taken up by agents a and c and role r' by agents b and d with a talking to b and c to d. More concretely, in the case of NSL described above, $T(NSL)$ contains all the shuffles of the following four traces induced by the instantiation just given (and many more).

$$send_1(a, b, \{a, n_a^1\}_{pk_b})\sharp 1 \cdot read_2(b, a, \{b, n_a^1, n_b^2\}_{pk_a})\sharp 1 \cdot send_3(a, b, \{n_b^2\}_{pk_b})\sharp 1 \cdot claim_4(a, i\text{-}synch)\sharp 1$$
$$read_1(a, b, \{a, n_a^1\}_{pk_b})\sharp 2 \cdot send_2(b, a, \{b, n_a^1, n_b^2\}_{pk_a})\sharp 2 \cdot read_3(a, b, \{n_b^2\}_{pk_b})\sharp 2 \cdot claim_5(b, i\text{-}synch)\sharp 2$$
$$send_1(c, a, \{c, n_c^3\}_{pk_a})\sharp 3 \cdot read_2(a, c, \{a, n_c^3, n_a^4\}_{pk_c})\sharp 3 \cdot send_3(c, a, \{n_a^4\}_{pk_a})\sharp 3 \cdot claim_4(c, i\text{-}synch)\sharp 3$$
$$read_1(c, a, \{c, n_c^3\}_{pk_a})\sharp 4 \cdot send_2(a, c, \{a, n_c^3, n_a^4\}_{pk_c})\sharp 4 \cdot read_3(c, a, \{n_a^4\}_{pk_a})\sharp 4 \cdot claim_5(a, i\text{-}synch)\sharp 4$$

Please note that the send and read events that make up these traces are really executed by the running agents. So, the occurrence of an event $read_\ell(r, r', m)$ in trace α denotes that in this scenario α agent r' really receives message m, which might or might not have been sent by agent r. This means that the read and send events are not intruder events. Nevertheless, the order and contents of these events may be under control of an intruder, as specified in a formal semantic definition. Due to the fact that the read and send events are executed by the agents (performing some role in the protocol) it is clear that the extension of these events with labels and run identifiers is by no means under control of the intruder. The label (like a program counter) simply indicates the state of the agent when executing his part of the protocol. The run identifier expresses in which run of the agent activity takes place.

Although it is not common to include labels and run identifiers in the semantics of a security protocol, they are often left implicit or occur when developing tool support for a particular semantics. For an example of such use of labels and run identifiers see [7]. Error traces of a protocol are displayed as follows: "$Msg\ \alpha.1.\ I_A \rightarrow B : A, B;\ Msg\ \alpha.2.\ B \rightarrow I_A : B, N_b;\ Msg\ \beta.1.\ I_A \rightarrow B : A, N_b$". Clearly, α and β represent run identifiers and the numbers $1, 2$, etc. (the program counters in the role) represent labels. Making this information explicit allows us to formally reason about these notions.

3 Synchronisation

In this section we formally define the notion of synchronization based on the trace model introduced in Section 2. As is the case with other forms of authentication, we will make a distinction between non-injective synchronisation, and the somewhat stronger notion of injective synchronisation, which rules out a class of replay attacks.

3.1 Non-injective Synchronisation

For the purpose of a modular presentation, we present the definition of non-injective synchronisation in three steps. The first step is the definition of authentication for a single message, or rather, for a single label shared by two corresponding send and read events. For this we define the predicate *1L-SYNCH* (for 1 label synchronisation). It has five arguments: the trace α being validated up to index k, the label ℓ for which we check the authentication property, and the two runs rid_1 and rid_2 containing the read and send events that must be validated. This basic authentication property simply amounts to checking whether the send labeled with ℓ from the first run is followed by the corresponding read from the second run. These corresponding send and read events should exactly agree upon the the sender, the recipient, and the contents of the message.

Definition 3.1 For all traces α, $k \in N$, labels ℓ and run identifiers rid_1, rid_2, the single-label synchronization predicate *1L-SYNCH* is given by

$$\begin{array}{l} \mathit{1L\text{-}SYNCH}(\alpha, k, \ell, rid_1, rid_2) \iff \\ \quad \exists_{i,j \in N, a,b \in Agent, m \in RunMess} \\ \qquad i < j < k \wedge \alpha_i = send_\ell(a, b, m) \sharp rid_1 \wedge \alpha_j = read_\ell(a, b, m) \sharp rid_2 \end{array}$$

If $\mathit{1L\text{-}SYNCH}(\alpha, k, \ell, rid_1, rid_2)$ holds, we say that the communication labeled with ℓ, sent by run rid_1 and read by run rid_2 has occurred correctly in a trace α before position k.

The second step towards the definition of synchronisation is to extend the *1L-SYNCH* predicate to range over multiple communications, given by a set of labels. This predicate is denoted by *ML-SYNCH* (for multi-label synchronisation).

When dealing with multiple communications, we require consistency over roles. For example, if an initiator receives two messages from a responder during a run, we require that they are sent from the same responder in the same run. The requirement that roles from a protocol are consistently mapped to runs leads to the introduction of an instantiation function $inst : Role \rightarrow RunId$. This explains the four parameters of the *ML-SYNCH* predicate: α and k give the part of the trace that is to be checked, L is the set of labels for each of which we check the *1L-SYNCH* property, and *inst* is the instantiation function which determines a run for each role definition from the protocol. The definition of *ML-SYNCH* consists of the requirement that for every label from L the predicate *1L-SYNCH* holds. In order to determine for a given label the sending role and the receiving role, we use the auxiliary functions *sendrole* and *readrole*, respectively, which have been introduced in Section 2.

Definition 3.2 For all traces α, $k \in N$, label set L and instantiation *inst*, the multi-label synchronization predicate *ML-SYNCH* is given by

$$\begin{aligned} &\mathit{ML\text{-}SYNCH}(\alpha, k, L, \mathit{inst}) \iff \\ &\quad \forall_{\ell \in L}\ \mathit{1L\text{-}SYNCH}(\alpha, k, \ell, \mathit{inst}(\mathit{sendrole}(\ell)), \mathit{inst}(\mathit{readrole}(\ell)))\ \end{aligned}$$

If $\mathit{ML\text{-}SYNCH}(\alpha, k, L, \mathit{inst})$ holds, we say that the set of labels L has correctly occurred in a trace α before position k with respect to the instantiation *inst*.

For the third and final step towards the definition of synchronisation we need to fix the set of labels that should be checked and we must quantify over the α, k and *inst* parameters of *ML-SYNCH*.

It is tempting to take for L the set of all labels occurring in the protocol. However, as a consequence of our choice to have security claims attached to individual roles, rather than to a protocol as a whole, this would result in too strong a predicate.

For any (local) synchronisation claim, we can at most require that all communications that causally precede the claim, have occurred correctly. This is because the claiming role may only expect these events to have been executed. Since a communication consists of two events, we must make it more precise when a communication precedes a claim. Clearly, it must be the case that if a read event precedes the claim, the corresponding send event precedes as well. However, in general we cannot expect the opposite. It may be the case that a send (causally) occurs before a claim, while the corresponding read does not. This observation leads to the definition of the set of labels that causally precede a synchronisation claim.

Definition 3.3 The set $prec(p, cl)$ of causally preceding communications of a claim role event labeled with cl, for a security protocol p is given by

$$prec(p, cl) = \{\ell \mid read_\ell(\_, \_, \_) \prec_p claim_{cl}(\_, \_)\}$$

Section 2, and Figure 1. Note that for the NSL protocol, in agreement with the discussion in Section 2, we have that $prec(NSL, 4) \neq prec(NSL, 5)$. In particular, $read_3(I, R, \{n_R\}_{pk_R}) \prec_p claim_5(R, ni\text{-}synch)$, but $read_3(I, R, \{n_R\}_{pk_R}) \not\prec_p claim_4(R, ni\text{-}synch)$, as can be seen in Figure 1.

Finally, we define the synchronisation predicate *NI-SYNCH*. It states that if we encounter a synchronisation claim $claim_\ell(a, ni\text{-}synch)\sharp rid$ as an event in any of the traces α of a security protocol, the communications causally preceding this claim must be well behaved. With respect to the instantiation function *inst*, we can simply require the existence of any such function, mapping roles to run identifiers, with the restriction that it maps the role of the claiming agent to the run containing this claim (i.e. $inst(role(\alpha_k)) = rid$, where α_k is the claim event).

Definition 3.4 For all security protocols p, the synchronisation predicate *NI-SYNCH* for claims labeled with ℓ, is given by

$$
\begin{array}{l}
NI\text{-}SYNCH(p, \ell) \iff \\
\quad \forall_{\alpha \in T(p), k \in N, a \in Agent, rid \in RunId}\ \alpha_k = claim_\ell(a, ni\text{-}synch)\sharp rid \Rightarrow \\
\qquad \exists_{inst:Role \to RunId}\ inst(role(\alpha_k)) = rid \wedge ML\text{-}SYNCH(\alpha, k, prec(p, \ell), inst)
\end{array}
$$

It expresses that for all instantiated claims in any trace of a given security protocol, there exist runs for the other roles in the protocol, such that all communications preceding the claim must have occurred correctly within these runs.

3.2 Injective synchronisation

Synchronisation guarantees that a single protocol role run has executed as expected. For each other role in the protocol, there exists a run that has sent and read messages according to the protocol. However, this property does not rule out a particular type of replay attacks. If we consider, for instance, a simple two-party protocol, it can be the case that every run of the initiator neatly matches the same run of the responder. Such a protocol could be abused by an intruder replaying an old responder run for every new session. (See Figure 3 in Section 4.2 for a more detailed discussion.) This weakness of the protocol could easily be detected by requiring that there is an injective relation between the claiming run and the other runs. This notion of injectivity has been defined for several authentication properties (see e.g. [8]). Here we will define it for synchronisation.

The definition of injective synchronisation proceeds in two steps. The first step is to bring the definition of non-injective synchronisation into a form that allows us to easily add the injectivity criterion. Note that in Definition 3.4 the relation between the claiming run (with run identifier *rid*) and the runs playing the other roles (as expressed by the instantiation function *inst*) is not made explicit. If we want to require that this relation is injective, we must first formulate an explicit version of this definition. The functional dependence of the instantiations *inst* on the claiming run *rid* will be expressed by a function

$Inst : RunId \times Role \rightarrow RunId$. We can easily extract the instantiation functions from $Inst$ by taking $inst = \lambda r.Inst(rid, r)$, i.e. the bindings that are relevant for the run identifier rid. These observations yield the following rewrite of Definition 3.4.

$$\begin{array}{l} NI\text{-}SYNCH(p,\ell) \iff \\ \quad \forall_{\alpha \in T(p)} \exists_{Inst:RunId \times Role \rightarrow RunId} \forall_{k \in N, a \in Agent, rid \in RunId} \\ \qquad \alpha_k = claim_\ell(a, ni\text{-}synch) \sharp rid \Rightarrow Inst(rid, role(\alpha_k)) = rid \\ \qquad\quad \wedge\ ML\text{-}SYNCH(\alpha, k, prec(p,\ell), \lambda r.Inst(rid, r)) \end{array}$$

We can now define injective synchronisation in almost the same way as the second formulation of *NI-SYNCH*, by requiring that the function *Inst* in the definition is injective.

Definition 3.5 For all security protocols p, the injective synchronisation predicate
I-SYNCH for claims labeled with ℓ, is given by

$$\begin{array}{l} I\text{-}SYNCH(p,\ell) \iff \\ \quad \forall_{\alpha \in T(p)} \exists_{Inst:RunId \times Role \rightarrow RunId \text{ injective}} \forall_{k \in N, a \in Agent, rid \in RunId} \\ \qquad \alpha_k = claim_\ell(a, i\text{-}synch) \sharp rid \Rightarrow Inst(rid, role(\alpha_k)) = rid \\ \qquad\quad \wedge\ ML\text{-}SYNCH(\alpha, k, prec(p,\ell), \lambda r.Inst(rid, r)) \end{array}$$

It expresses that for all instantiated claims, there exist runs for the other roles in the protocol. All communications preceding the claim must have occurred correctly within these runs. Furthermore, for each such claim there are unique runs executing the roles in the protocol.

4 Agreement

In this section we formalize a form of authentication called agreement. This concept is weaker than synchronisation, as it places less restrictions on the ordering of the occurring events. We first give formal definitions. We will argue that these definitions correspond to a form of *agreement* as described by Lowe in [8]. Next, we present an extension of the authentication hierarchy and provide examples showing the strictness of the various relationships.

4.1 Non-injective and injective agreement

We construct the definition of agreement analogously to synchronisation. Synchronisation required that all sends occur before their corresponding reads. The definition of *1L-AGREE* does not require that the send occurs before the corresponding read.

Definition 4.1 For all traces α, $k \in N$, labels ℓ and run identifiers rid_1, rid_2, the single-label agreement predicate *1L-AGREE* is given by

$$\begin{array}{l} 1L\text{-}AGREE(\alpha, k, \ell, rid_1, rid_2) \iff \\ \quad \exists_{i,j \in N, a,b \in Agent, m \in RunMess} \\ \qquad i < k \wedge j < k \wedge \alpha_i{=}send_\ell(a, b, m) \sharp rid_1 \wedge \alpha_j{=}read_\ell(a, b, m) \sharp rid_2 \end{array}$$

If $\mathit{1L\text{-}AGREE}(\alpha, k, \ell, rid_1, rid_2)$ holds we say that the communication labeled with ℓ, sent by run rid_1 and read by run rid_2, is agreed upon in trace α before position k.

There are no requirements on the order of the read and send. We observe that if *1L-AGREE* holds for some communication, all variables sent as part of the message m are received exactly as expected. Therefore, both parties will agree over the values of the variables that are sent. Further definitions are constructed as with synchronisation. First we define a notion of multi-label agreement similar to *ML-SYNCH*.

Definition 4.2 For all traces α, $k \in N$, set of labels L and run identifiers rid_1, rid_2, the multi-label agreement predicate *ML-AGREE* is given by

$$\begin{array}{l}\mathit{ML\text{-}AGREE}(\alpha, k, L, \mathit{inst}) \iff \\ \quad \forall_{\ell \in L}\ \mathit{1L\text{-}AGREE}(\alpha, k, \ell, \mathit{inst}(\mathit{sendrole}(\ell)), \mathit{inst}(\mathit{readrole}(\ell)))\end{array}$$

If $\mathit{ML\text{-}AGREE}(\alpha, k, L, \mathit{inst})$ holds we say that the set of labels L is agreed upon in trace α before position k, for instantiation function inst.

Next we define non-injective agreement.

Definition 4.3 For all security protocols p, the agreement predicate *NI-AGREE* for claims labeled with ℓ, is given by

$$\begin{array}{l}\mathit{NI\text{-}AGREE}(p, \ell) \iff \\ \quad \forall_{\alpha \in T(p), k \in N, a \in \mathit{Agent}, \mathit{rid} \in \mathit{RunId}} \\ \qquad \alpha_k = \mathit{claim}_\ell(a, \mathit{ni\text{-}agree}) \sharp \mathit{rid} \Rightarrow \exists_{\mathit{inst}:\mathit{Role} \to \mathit{RunId}}\ \mathit{inst}(\mathit{role}(\alpha_k)) = \mathit{rid} \\ \qquad\quad \wedge\ \mathit{ML\text{-}AGREE}(\alpha, k, \mathit{prec}(p, \ell), \mathit{inst})\end{array}$$

The agreement predicate expresses that for all instantiated claims in any trace of a given security protocol, there exist runs for the other roles in the protocol, such that all communication events causally preceding the claim must have occurred before the claim.

Injective agreement is defined in the same way as injective synchronisation.

Definition 4.4 [*I-AGREE*] For all security protocols p, the injective agreement predicate *I-AGREE* for claims labeled with ℓ, is given by

$$\begin{array}{l}\mathit{I\text{-}AGREE}(p, \ell) \iff \\ \quad \forall_{\alpha \in T(p)}\ \exists_{\mathit{Inst}:\mathit{RunId} \times \mathit{Role} \to \mathit{RunId}\ \text{injective}}\ \forall_{k \in N, a \in \mathit{Agent}, \mathit{rid} \in \mathit{RunId}} \\ \qquad \alpha_k = \mathit{claim}_\ell(a, \mathit{i\text{-}agree}) \sharp \mathit{rid} \Rightarrow \mathit{Inst}(\mathit{rid}, \mathit{role}(\alpha_k)) = \mathit{rid} \\ \qquad\quad \wedge\ \mathit{ML\text{-}AGREE}(\alpha, k, \mathit{prec}(p, \ell), \lambda r.\mathit{Inst}(\mathit{rid}, r))\end{array}$$

It expresses that for any trace and for any run of any role in the protocol there exist unique runs for the other roles of the protocol such that for all claims occurring in the trace all communications preceding the claim must have occurred correctly within these runs.

In [8], Lowe defines several forms of authentication. The strongest form of authentication, not involving time, is called *full (injective) agreement*:

> Initiator I is in agreement with responder R, whenever I as initiator completes a run of the protocol with R, then R as responder has been running the protocol with I. Moreover, I and R agree on all data variables, and each run of I corresponds to a unique run of R.

We will argue that for any protocol for which an *I-AGREE* claim at the end of a role holds, full (injective) agreement holds with respect to all other roles and data items, and vice versa. Note that we will only consider the case where agreement is claimed over all roles involved in the protocol.

The main difference between our approach and that of Lowe, is that we consider abstract messages only. The definition of full injective agreement does not depend on the actual messages that are passed through. Instead, Lowe refers to the data items contained in messages. The relation between the messages and the data items does imply however, that when a message is read exactly as it is sent, both agents will agree over the variables sent in the message, and the other way around. Thus, if all messages in a protocol are read as they were sent, there must be agreement over all variables in the protocol.

The definition of *I-AGREE* does not involve *all* communications, but only the set $prec(p, \ell)$ of communications that precede a claim. However, it turns out that the way in which full agreement is made precise in terms of CSP, as can be checked by compiling Casper-code into CSP, it also takes only preceding communications into account. For this, running-commit signals (see [11]) are introduced in the protocol. For each role, a running signal is added to the last communication preceding the agreement claim. In the role of the claim, a commit signal is added to the last communication. Full injective agreement over all roles requires that the running signals of each role precede the commit signal. This corresponds to the order requirements of *I-AGREE*.

It follows that the notions of *I-AGREE* and full injective agreement over all roles coincide.

4.2 Hierarchy

The formulations of the four security properties in the previous sections clearly reveal their relative strengths in preventing attacks. Every injective protocol is also non-injective and if a protocol satisfies agreement then it satisfies synchronisation too. Figure 2 shows this hierarchy. An arrow from property X to property Y means that every protocol satisfying X also satisfies Y. Phrased differently, the class of protocols satisfying X is included in the class satisfying Y.

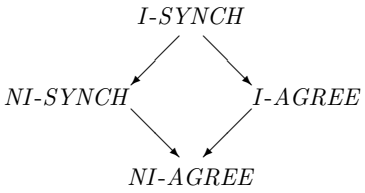

Fig. 2. Hierarchy of security properties.

The proof of the following theorem simply follows from the definitions above.

Theorem 1. *The security properties I-SYNCH, NI-SYNCH, I-AGREE, and NI-AGREE satisfy the inclusion relation as depicted in Figure 2.*

The question whether the inclusions in Figure 2 are strict is harder to answer. This is due to the abstractness of our trace model. Since our approach is parameterised over the actual semantics, and thus over the intruder model, we cannot determine for a given protocol to which class it belongs. Therefore, strictness of the inclusions can only be answered relative to a given semantics. Consequently, the following reasoning will be at a conceptual level only.

If we take e.g. a model where all agents simply follow their roles and the intruder has no capabilities at all, then the diamond in Figure 2 collapses into a single class. The same holds if the intruder can only eavesdrop on the communications. However, in the Dolev-Yao model, all inclusions are strict. The case of injectivity vs. non-injectivity has been studied extensively before. The MSC on the left in Figure 3 shows a protocol that satisfies *NI-SYNCH* and *NI-AGREE*, but neither *I-SYNCH*, nor *I-AGREE*.

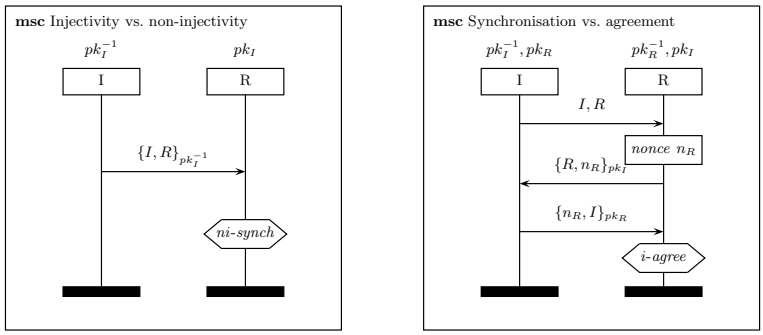

Fig. 3. Distinguishing Protocols

The intruder will only be able to construct message $\{I, R\}_{pk_I^{-1}}$ after having eavesdropped this message from a previous run. Therefore every read event of this message is preceded by a corresponding send event, so the protocol is both *NI-SYNCH* and *NI-AGREE*. However, once the intruder has learnt this message, he can replay it as often as desired, so the protocol is not injective.

A distinguishing example between synchronisation and agreement is depicted on the right in Figure 3. As confirmed by the Casper/FDR tool set, this protocol satisfies unilateral authentication in the sense of agreement (both injective and non-injective). However, the protocol does not satisfy synchronisation (both variants): the intruder can send message I, R long before I actually initiates the protocol, making R to believe that I has requested the start of a session before he actually did.

The two examples show that the diamond in Figure 2 is strict if the intruder has the capabilities to eavesdrop, deflect and inject messages. Both examples also imply that there are no arrows between *NI-SYNCH* and *I-AGREE*.

In Figure 4 we show how the difference between *NI-SYNCH* and *NI-AGREE* might be exploited. Here, R is an Internet Service Provider, used by I. I pays R for the time he is connected. When I wants to connect, R retrieves the certificate of I from the trusted server S and uses this to authenticate I. After a successful session, I is billed from the moment the first message was received by R.

This protocol is a slightly modified version of the Needham-Schroeder-Lowe protocol. It can be exploited as follows. An intruder can send the first message preemptively, causing R to initiate a session with what it believes is I. If at some later time I decides to initiate a session with R and finishes it successfully, I will receive a bill that is too high. In fact, although, this protocol satisfies agreement for R, the first message is not authenticated at all.

This protocol does not satisfy synchronisation. The protocol can be easily modified to satisfy *NI-SYNCH* and thus to be resilient against the sketched type of timing attacks.

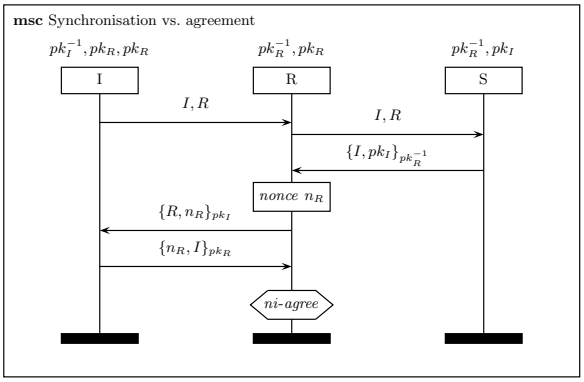

Fig. 4. A protocol that satisfies *NI-AGREE* but not *NI-SYNCH*.

5 Conclusions

In this section we summarize the main contributions of our research and discuss some directions for future work.

First of all, we have defined a general trace model for security protocols which allows for the definition of security properties in an intensional style. This trace model is not tied to a particular semantics, making it independent of e.g. the execution model of an agent and the intruder model. The starting point of the trace model is a role-based protocol description, where security claims are local to

the protocol roles. Such a subjective security claim expresses what an agent may safely assume after having executed his part of the protocol. The events in our model are extended with event labels and run identifiers to unambiguously define the origin of an event. These attributes are not under control of an intruder, but serve to identify the events when reasoning about protocols. The main motivation for developing the trace model was to study intensionality of specifications (which we call synchronisation) and agreement in an abstract framework, allowing us to pinpoint what the exact differences are. Due to the uniform phrasing, the two notions of authentication can be distinguished easily: agreement allows that an intruder injects a (correct and expected) message before it is sent by the originator of the message. As for agreement, we provide both an injective and a non-injective variant of synchronisation.

Since we only assumed an abstract trace model, the theory presented in this paper does not suffice to prove protocols (in)correct. For this purpose we are currently defining a canonical operational semantics of security protocols which satisfies the requirements for the trace model put forward here. Experiments with this operational definition of synchronisation indicate that formal proofs that security protocols satisfy synchronisation are feasible.

Finally, we think that it would be interesting to study extensions of our model, such as timing and recentness as has been done in [8].

References

1. W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key-exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.
2. A.G. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for Message Sequence Charts. *SoCP*, 44:253–292, 2002.
3. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *World Congress on Formal Methods (1)*, pages 794–813, 1999.
4. D. Gollmann. What do we mean by entity authentication. In *Proc. Symposium on Research in Security and Privacy*, pages 46–54. IEEE, 1996.
5. ITU-TS. *Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1999.
6. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 147–166. LNCS 1055, 1996.
7. G. Lowe. Some new attacks upon security protocols. In *Proc. 9th Computer Security Foundations Workshop*, pages 162–169. IEEE, 1996.
8. G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th Computer Security Foundations Workshop*, pages 31–44. IEEE, 1997.
9. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:120–126, 1978.
10. A. W. Roscoe. Intensional Specifications of Security Protocols. In *Proc. 9th Computer Security Foundations Workshop*, pages 28–38. IEEE, 1996.
11. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols.* Addison Wesley, 2000.

Nested Timing Attacks *

Damas P. GRUSKA[1] and Andrea MAGGIOLO-SCHETTINI[2]

[1] Institute of Informatics, Comenius University,
Mlynska dolina, 842 48 Bratislava, Slovakia,
gruska@fmph.uniba.sk.
[2] Dipartimento di Informatica, Università di Pisa,
Via Buonarroti 2, 56127 Pisa, Italy,
maggiolo@di.unipi.it

Abstract. A formal model for a description of timing attacks nested in (computer) networks is presented. It is based on timed process algebra which can express also network properties. Robustness of systems against nested timing attacks is defined in the style of Bisimulation-based Non-Deducibility on Composition. Other, finer privacy properties for basic process algebras are defined with the help of timed and networked process algebras.

Keywords: Process Algebras, Timing Attacks, Computer Networks

1 Introduction

Timing attacks represent a powerful tool for "breaking" "unbreakable" systems, algorithms, protocols, etc. For example, by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems (see [Ko96]). This idea was developed in [DKL98] were a timing attack against smart card implementation of RSA was conducted. In [HH99], a timing attack on the RC5 block encryption algorithm, in [SWT01] the one against the popular SSH protocol and in [FS00] the one against web privacy are described.

Formal methods play a growing role not only in the design of (mainly critical) software applications and hardware components but also in checking their security properties. In practice, various formalisms are used. They range from graphical languages such as Petri Nets or Statecharts to algebraic, logic or special programming languages/models. *Process algebras* are an important class of such formalisms. They successfully describe the behavior of systems as "communicating systems" and they usually abstract away many real properties of existing systems, such as duration

* Work supported by the grant VEGA 1/0172/03 and MIUR Progetto MEFISTO.

and structure of actions, properties of communication networks, distribution of system components, and so on. If one wants to analyze systems with respect to timing attacks in interconnection networks, a special process algebra which is enriched by time and network reasoning has to be used. The aim of this paper is to formalize a notion of "nested timing attacks" by means of a particular (dialect of) process algebra NTiCCS (see [GM01]), which is based on Milner's CCS. Moreover, we show a way how to define system robustness with respect to timing attacks. We assume that attackers are *passive*, namely that they base their attacks only on observing a given timed sequence of public (low) actions, and deriving from this observation the certainty that a certain secret (high) action has been performed. Moreover, we assume that attackers can be nested in a network of heterogeneous structure where some connections might be of a limited capacity. Depending on its nesting the attacker might be or might not be able to perform a timing attack. A possible nesting of the attacker is modeled by a special place holder. The place holder might be seen as an unspecified component, and thus the whole system might be seen as an open system (see [M99,M03]) of processes with the place holder as the context (see [CSFW03]).

This paper is going in the direction given by papers [FG01], [FGM00] and [FGM03]. While in the first of them attacks are defined in general, in the second and third paper discrete time setting is introduced. Here we add also a structure of interconnection network. By this finer, more descriptive calculus one can express and analyze situations which are otherwise very difficult to handle. Note that most of timing attacks strongly depend on network properties. Sometimes only speed of communication is important (for example, well known timing attacks on smart cards are not possible without very fast communication between the card and an attacker), but in other cases an attacker must exploit finer network properties than just its speed (for example, an attacker can discover and exploit internal communications between subsystems).

The paper is organized as follows. In Section 2 we describe the language NTiCCS. In Section 3 we present notion of nested timing attacks expressed in terms of this formalism.

2 Process Algebras for Network Communication

In this section we introduce a dialect of NTiCCS suitable to express timing attacks. To illustrate the design of the language we are going to present, let us start with a typical example of a communication network (see Fig. 1,

but note that the presented formalism allows modeling networks of an arbitrary topology). It contains two kinds of interconnection (sub)networks: fast networks between C_i and C_i' (for example local buses) and shared network with limited throughput among C_i' (for example optical networks with wavelength-division multiplexing).

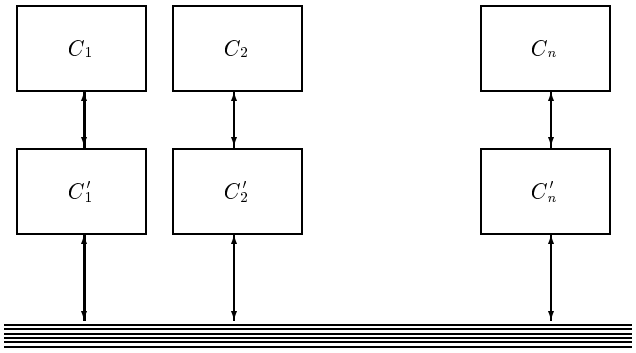

Fig. 1. Network Scheme

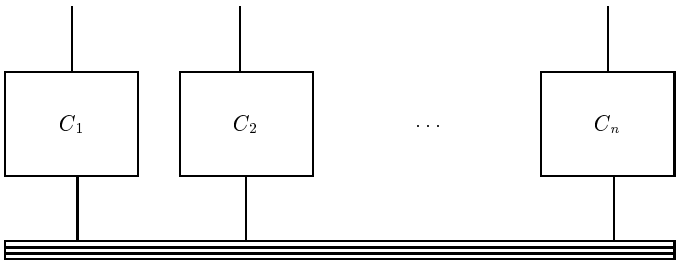

Fig. 2. Network Scheme 1

In general, the presented calculus allows modeling two types of connections and any combination of them. In this way it covers any possible network architecture. Fig. 2 shows a network of subsystems $C_1, \ldots, C_n$

which are connected via an "ethernet like" link of a limited capacity. This means that any communication between C_i and C_j occupies one link (not just a path between C_i and C_j). Communications inside each component C_i are considered to be made by means of a fully connected network.

A network where a communication between two subsystems has no influence on other communications is depicted in Fig. 3.

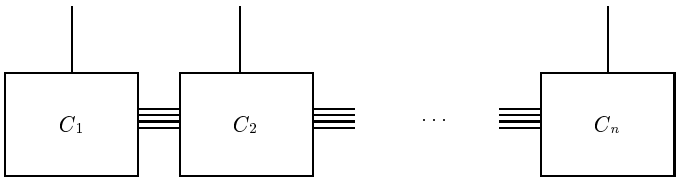

Fig. 3. Network Scheme 2

We assume that performing communications via a limited capacity network decreases the number of free communication links. This means that a communication of this kind consumes a communication link, and therefore the communication is possible only if there is free link in the network. After completion of the communication the link is free again. We only require that n links can transmit n pieces of information at the same time. We assume that the current number of free links depends on the current number of communicating processes. This means that, during an execution, a process may have to wait for a free communication link. Now, if there are no free links, a networked process cannot perform any communication via the corresponding (sub) network. At this point we depart from a common assumption of timed calculi that communications are instantaneous (minimal idling property), which simplifies problems of communications to an unrealistic level, and instead we assume a *restricted minimal idling property*. This means that, in case of "limited" communications, an internal communication can idle if there is no free communication link in a network. In general, we assume that the duration of an action might be different from the duration of its communication part, i.e. the time for which the action occupies a link. So, this does not lead to a restriction on the maximal number of concurrently running actions or processes.

Now an atomic action will not represent a communication but just its beginning. We will distinguish two kinds of communications. For communications (of concurrently running processes) via a complete network we

will use the standard CCS parallel operator $|$. For communications via a network with a limited number of links we will use the new parallel operators $[. \parallel \ldots\ldots \parallel .]^n_L$. To count the number of busy links, we will indicate the total number of links and the current number of busy links. As regards busy links, we need two pieces of information. The number of busy links and the time needed to finish each communication. We will indicate this by a multi-set of positive integers $L = \{n_1, n_2, \ldots, n_k\}$. A process $[P_1 \parallel P_2 \ldots \parallel P_m]^n_L$ has n links at its disposal but at the moment k of them are busy. Number n_i indicates how many time units are needed to complete a communication and to release a link. To express run of time and duration of actions we exploit a special action t as for ticks.

To define the language NTiCCS, we first assume a set of atomic action symbols A not containing symbols τ and t, and such that for every $a \in A$ there exists $\overline{a} \in A$ and $\overline{\overline{a}} = a$. We define $Act = A \cup \{\tau\}$, $Actt = Act \cup \{t\}$. We assume that $a, b, \ldots$ range over A, $u, v, \ldots$ range over Act, and $x, y \ldots$ range over $Actt$. We suppose that L is a finite multi-set of positive integers. By $|L|$ we will indicate the cardinality of L. Let $L = \{n_1, \ldots, n_k\}$. By $L - 1$ we will mean a multi-set $\{n_i - 1 | 1 \leq i \leq k, n_i \in L, n_i - 1 > 0\}$. If L is empty then $L - 1$ is the empty set. Let $c, d : A \to N$ such that $d(a) \geq c(a)$. By $d(a)$ we will indicate the duration of action a and by $c(a)$ we will indicate the duration of a communication part of a. The set of NTiCCS terms over the signature Σ is defined by the following BNF notation:

$$P \ ::= \ X \ \mid \ op(P_1, P_2, \ldots P_n) \ \mid \ \mu X P$$

where $X \in Var$, Var is a set of process variables, $P, P_1, \ldots P_n$ are NTiCCS terms, $\mu X -$ is the binding construct, $op \in \Sigma$. Assume the signature $\Sigma = \bigcup_{n \geq 0} \Sigma_n$, where

$\Sigma_0 = \{Nil\}$
$\Sigma_1 = \{x \mid x \in A \cup \{t\}\} \cup \{[S] \mid S \text{ is a relabeling function}\} \cup \{\backslash M \mid M \subseteq A\}$
$\Sigma_2 = \{|, +, [. \parallel .]^n_L\}$
$\Sigma_j = [. \parallel . \parallel \ldots\ldots \parallel .]^n_L, j > 2$

with the agreement to write unary action operators in prefix form, the unary operators $[S], \backslash L$ in postfix form, and the rest of operators in infix form. To have shorter terms, sometimes the term Nil will be omitted. Relabeling functions, $S : Actt \to Actt$ are such that $\overline{S(a)} = S(\bar{a})$ for $a \in A, S(\tau) = \tau$ and $S(t) = t$. Moreover, $M \subseteq A$, $n \in N^+$ and L is a

multi-set of positive integers. The set of CCS terms consists of NTiCCS terms without t action and $[. \parallel . \parallel \ldots\ldots \parallel .]^n_L$ operators. Closed terms are called processes.

We give a structural operational semantics of terms by means of labeled transition systems. The set of terms represents a set of states, labels are actions from *Actt*. The transition relation $\rightarrow$ is a subset of NTiCCS $\times$ *Actt* $\times$ NTiCCS. We write $P \stackrel{x}{\rightarrow} P'$ instead of $(P, x, P') \in \rightarrow$ and $P \stackrel{x}{\not\rightarrow}$ if there is no P' such that $P \stackrel{x}{\rightarrow} P'$. The meaning of the expression $P \stackrel{x}{\rightarrow} P'$ is that the term P can evolve to P' by performing action x, by $P \stackrel{x}{\rightarrow}$ we will denote that there exists a term P' such that $P \stackrel{x}{\rightarrow} P'$. We define the network transition relation as the least relation satisfying the following inference rules:

$$\frac{}{x.P \stackrel{x}{\rightarrow} P} \; A1 \qquad \frac{}{u.P \stackrel{t}{\rightarrow} u.P} \; A2$$

$$\frac{}{Nil \stackrel{t}{\rightarrow} Nil} \; A3 \qquad \frac{P \stackrel{u}{\rightarrow} P'}{P \mid Q \stackrel{u}{\rightarrow} P' \mid Q} \; Pa1$$

$$\frac{P \stackrel{u}{\rightarrow} P'}{Q \mid P \stackrel{u}{\rightarrow} Q \mid P'} \; Pa2 \qquad \frac{P \stackrel{a}{\rightarrow} P', Q \stackrel{\overline{a}}{\rightarrow} Q'}{P \mid Q \stackrel{\tau}{\rightarrow} P' \mid Q'} \; Pa3$$

$$\frac{P \stackrel{t}{\rightarrow} P', Q \stackrel{t}{\rightarrow} Q', P \mid Q \stackrel{\bar{\tau}}{\not\rightarrow}}{P \mid Q \stackrel{t}{\rightarrow} P' \mid Q'} \; Pa4 \qquad \frac{P \stackrel{u}{\rightarrow} P'}{P + Q \stackrel{u}{\rightarrow} P'} \; S1$$

$$\frac{P \stackrel{u}{\rightarrow} P'}{Q + P \stackrel{u}{\rightarrow} P'} \; S2 \qquad \frac{P \stackrel{t}{\rightarrow} P', Q \stackrel{t}{\rightarrow} Q'}{P + Q \stackrel{t}{\rightarrow} P' + Q'} \; S3$$

$$\frac{P_i \stackrel{u}{\rightarrow} P_i', \text{for some } i, 1 \leq i \leq m}{[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m]^n_L \stackrel{u}{\rightarrow} [P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_m]^n_L} \; N1$$

$$\frac{P_i \stackrel{a}{\rightarrow} P_i', P_j \stackrel{\overline{a}}{\rightarrow} P_j', \text{for some } i, j, 1 \leq i, j \leq m, |L| < n}{[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_j \parallel \ldots \parallel P_m]^n_L \stackrel{\tau}{\rightarrow} [P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_j' \parallel \ldots \parallel P_m]^n_{L \cup \{c(a)\}}} \; N2$$

$$\frac{P_i \stackrel{t}{\rightarrow} P_i', \text{for every } i, 1 \leq i \leq m \text{ and } [P_1 \parallel \ldots \parallel P_m]^n_L \stackrel{\bar{\tau}}{\not\rightarrow}}{[P_1 \parallel \ldots \parallel P_m]^n_L \stackrel{t}{\rightarrow} [P_1' \parallel \ldots \parallel P_m']^n_{L-1}} \; N3$$

$$\frac{P \stackrel{x}{\rightarrow} P'}{P \setminus M \stackrel{x}{\rightarrow} P' \setminus M}, (x, \overline{x} \notin M) \; Res \qquad \frac{P[\mu X P / X] \stackrel{x}{\rightarrow} P'}{\mu X P \stackrel{x}{\rightarrow} P'} \; Rec \qquad \frac{P \stackrel{x}{\rightarrow} P'}{P[S] \stackrel{S(x)}{\rightarrow} P'[S]} \; Rl$$

Here we mention rules that are new with respect to CCS. Axioms $A2, A3$ allow arbitrary idling. Concurrent processes connected via full connection network can idle only if there is no possibility of an internal

communication ($Pa4$). A run of time is deterministic ($S3$). If concurrent processes are connected via a network of a limited capacity, they behave similarly except internal communications and t action. Any internal communication increases the number of busy links. It adds to L the duration of the communication part of the corresponding actions ($N2$). By any t step every member of L is decreased by 1. If any number was equal to 1 then after that step it is removed from L. This means that a link gets free. The action t can be performed ($N3$) if there is no possibility of an internal communication either between processes P_i, P_j (there are no two processes ready to communicate or there is no free link for such communication) or there is not a possibility of an internal communication "inside" P_i for every i (it can be easily proved by structural induction that $P \xrightarrow{t}$ implies $P \overset{\overline{t}}{\not\rightarrow}$).

3 Timing Attacks

In case of timing attacks, an attacker can see public actions of a system to be attacked and can measure time of their occurrence. From this information (s)he tries to deduce performing of private actions. For example, in [FS00] a timing attack on web privacy is described. The attack compromises the privacy of user's web-browsing histories by allowing a malicious web site to determine whether or not the user has recently visited some other, unrelated, web page w. An applet is embedded in the malicious web site and is run by the user's browser. The applet first performs a request to a file of w, and then performs a new request to the malicious site. Hence, the malicious site can measure the time elapsed between the two requests and it can infer that w was in the cache of the browser (implying that w has been recently visited by the user). In case of this attack the attacker exploits only very rough information about an interconnection network - (s)he knows only that communications with the cache are much faster than those with the web page. But situation becomes more complicated when the browser's cache is off and instead a proxy server is in use. Now the local network properties play a more important role. It depends on exact locations of the proxy server, switches, hubs, on possible sharing of telephone line etc. whether the attacker can detect if some local network user or who exactly, visited the web page.

In general, systems respect the property of privacy if there is no leaking of private information, namely there is no *information flow* from the high level to the low level. This means that the secret behavior cannot influence the observable one, or, equivalently, no information on the ob-

servable behavior permits to infer information on the secret one. In the case of timing attacks, timing of actions plays a crucial role. Moreover, the real-time behaviour of a system depends on its embedding in a particular interconnection network. Some communications are fast, some are slower and some might even be delayed if there is no free communication link. The above mentioned web attack was based on the property that cache communications (via local bus) are faster than communications with remote sites. By measuring time (duration) of communications an attacker could deduce whether the site has been visited or not.

To formalize nested timing attacks we define the set $\mathcal{A}-$NTiCCS of *attack opened* processes. The set of $\mathcal{A}-$NTiCCS terms over the signature Σ is defined by the following BNF notation:

$$P ::= \mathcal{A} \mid X \mid op(P_1, P_2, \ldots P_n) \mid \mu X P$$

where $\mathcal{A}$ is a place holder for an attacker and the rest is the same as in case of NTiCCS terms. Note that NTiCCS terms are $\mathcal{A}-$NTiCCS terms and for $\mathcal{A}-$NTiCCS terms we can use the same transition system which has been defined for NTiCCS terms where $\mathcal{A}$ behaves like Nil. An attack opened NTiCCS term is called process if it is closed and it contains exactly one occurrence of the place holder $\mathcal{A}$ which does not occur in a subterm of the form $x.F$. When it is clear from the context we will use the same notation for the set of processes and for the set of terms. For every NTiCCS process P there exists its "opening" for possible timing attacks. To formalize this we need some definitions.

Definition 1. *A binary relation $\Re \subseteq \mathcal{A}-NTiCCS \times \mathcal{A}-NTiCCS$ is a* bisimulation *if it is symmetric and if $P \Re Q$ and $P \xrightarrow{x} P', x \in Actt$, then there exists Q' such that $Q \xrightarrow{x} Q'$ and $P' \Re Q'$. Two terms P, Q are* bisimilar*, abbreviated $P \sim Q$, if there exists a bisimulation relating P and Q.*

Definition 2. *An attack opened process P' is an opening of process P iff $P \sim P'$.*

Clearly there might be several "openings" for a given process P, but all of them are bisimilar (and bisimilar with P itself). Each opening represents a possible "entrance" (the placeholder $\mathcal{A}$) for an attacker. Fig. 4 shows a network with two possible placings for an attacker, but we will consider only processes with only one placeholder $\mathcal{A}$ for an attacker.

Now, to model timing attacks by means of NTiCCS calculus, first we divide all actions in two groups, namely public (low level) actions $LActt$

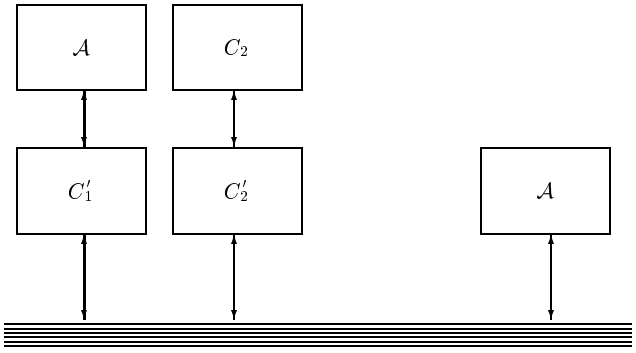

Fig. 4. Two different placings of an attacker

and private (high level) actions $HActt$ i.e. $A = LActt \cup HActt, LActt \cap HActt = \emptyset$ (see [FGM00]). Then we use Bisimulation-based Non-Deducibility on Composition (BNDC for short, see [FGM00]): a system is BNDC if for every high level user A, the low level view of the behaviour of P is not modified (in terms of bisimulation) by the presence of A. First some notation is needed: let $x \in Actt$ then $\hat{x} = x$ if $x \neq \tau$ and $\hat{\tau} = \epsilon$ (empty sequence). We will write $P \stackrel{x}{\Rightarrow} P'$ if $P(\stackrel{\tau}{\rightarrow})^* \stackrel{x}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* P'$. Now we can define weak bisimulation.

Definition 3. *A binary relation $\Re \subseteq NTiCCS \times NTiCCS$ is called a* weak bisimulation *if it is symmetric and if $P \Re Q$ and $P \stackrel{x}{\rightarrow} P', x \in Actt$, then there exists Q' such that $Q \stackrel{\hat{x}}{\Rightarrow} Q'$ and $P' \Re Q'$. Two terms P, Q are* weakly bisimilar*, abbreviated $P \approx Q$, if there exists a weak bisimulation relating P and Q.*

Now we need to distinguish processes which can perform only high level actions (high level users). Let $Sort(P) = \{x | P \stackrel{s.x}{\longrightarrow} \text{ for } s \in Actt^\star\}$. The idea of BNDC in the framework of "networked" processes can be formulated as follows.

Definition 4. *$P \in BNDC$ iff for every P', where P' is opening of P,*

$$(P'[A/\mathcal{A}]) \setminus HActt \approx P \setminus HActt$$

for all $A, Sort(A) \subseteq HActt \cup \{\tau, t\}$ *where* $P'[A/\mathcal{A}]$ *represents the substitution of the place holder* $\mathcal{A}$ *by the process* A.

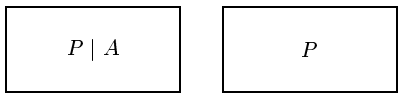

Fig. 5. Non-nested attacker

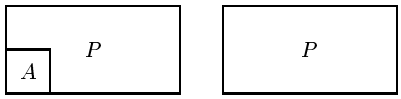

Fig. 6. Nested attacker

The presented notion of the BNDC property differs from those defined in [FGM00,FG01]. The original definition assumes that an attacker (A) communicates with the process to be attacked (P) at the top most level (see Fig. 5) and an observer cannot distinguish between those two systems if they perform only low level actions. Here we assume that an attacker A can be nested anywhere in the system P, which might of an arbitrary network architecture (see Fig. 6).

Example 1. Let

$$P_1 = h'.t.l_1.Nil + \bar{h}.t.l_1.Nil$$
$$P_2 = \bar{h}'.t.l_2.Nil + \bar{h}.t.l_2.Nil$$
$$S = [P_1 \parallel P_2]^1_{\{\emptyset\}}$$

and suppose that all communications take exactly one time unit i.e. $c(h) = c(h') = 1$. The system S consists of two processes P_1 and P_2 which communicate via a network of capacity 1, i.e. an ethernet like network which can transmit only one piece of information at the moment. Assume that we have attacker A, $A = h.t.Nil \mid h.t.Nil$ which is nested

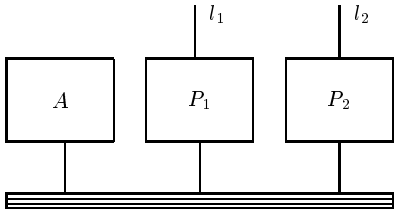

Fig. 7. Nested attack

in the system S (see Fig. 7), i.e. $S'[A/\mathcal{A}] = [A \parallel P_1 \parallel P_2]^1_{\{\emptyset\}}$ where S' is an opening of S.

Clearly, system S does not have BNDC property since $S'[A/\mathcal{A}] \setminus HActt$ can perform the sequence of actions $\tau.t.\tau.t.l_1$ while $S \setminus HActt$ cannot and hence $S'[A/\mathcal{A}] \setminus HActt \not\approx S \setminus HActt$. The reason is such that when the attacker A starts communication with P_2 by means of action h, the similar communication with P_1 can start only when the only link which they have at disposal is again free. So, after τ action, t action has to be performed to finish that communication before the next τ can be performed. In this case the attacker exploits "link busyness" to detect private behaviour.

On the other side, without the networked calculus we would model the system just by the ordinary parallel operator $|$, and the system above would have the property BNDC, what does not correspond to our intuition. □

NTiCCS processes can be seen as refinements of timed CCS ones and these again can be seen as refinements of pure CCS ones. If we take the basic specification of a system expressed as CCS process P we might add "time information" to it and get a timed (say, TiCCS) process P_t. Then, again, if we know the implementation (embedding) of the process P_t in a particular network architecture, we can add this information to P_t and get a networked (NTiCCS) process P_{nt} (see Fig. 8).

For each of these calculi we have the appropriate BNDC property. But if a process seems to be secure at some level (in the sense of the BNDC property) it can lose this property at a lower level where more details of its behaviour can be expressed. In this sense, it is meaningful to define a variant of BNDC property which is satisfied by a process if its more concrete variant exhibits BNDC property at a lower level. Hence, we can say that a process P has BNDC$_t$ property if any of its timed versions (i.e. for arbitrary timing of its actions) has BNDC property on the level of TiCCS. And, similarly, a process P has BNDC$_{nt}$ property if any NTiCCS process

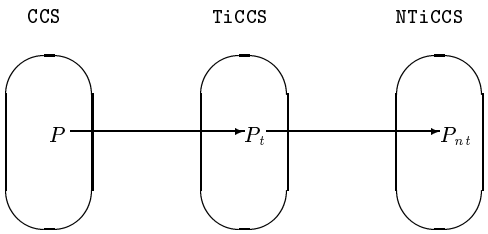

Fig. 8. Process refinements

P_{nt}, which is its timed and networked refinement, has BNDC property. Therefore, we have three properties for CCS which reflect different requirements for system security. If we denote by BNDC_{nt}, BNDC_t, BNDC appropriate subsets of CCS processes, we get the following inclusions (it can be checked that all of them are proper - see Example 1).

Theorem 1. *$BNDC_{nt} \subset BNDC_t \subset BNDC$.*

According to Definition 4 it seems to be difficult to check the BNDC property. One has to check it for every P', where P' is opening of P and for every A such that $Sort(A) \subseteq HActt \cup \{\tau, t\}$. Fortunately there are only finitely many "reasonable" openings for every process P.

Theorem 2. *For every process P there exists a finite set of its openings such that to check the BNDC property of P it is enough to check openings from this set.*

Proof. Sketch. For a given process P there are only finitely many positions where $\mathcal{A}$ can be meaningfully nested. □

As regards possible attacks A, there are infinitely many of them. One can check an approximation of BNDC called BSNNI (Bisimulation-based Strong Nondeterministic Non-Interference, see [FGM00]) by considering as an attacker the process of the form:

$$A = \sum_{h \in HActt} h.A.$$

Note that BSNNI property is weaker than BNDC also in the case of networked calculus. On the other side, an approximation which is stronger than BNDC can be obtained by requiring that also every derivation of the

process has BSNNI property. In case of the presented networked calculi it seems to be more appropriate to consider "the most powerful attacker" in the form $A = \sum_{h \in HActt, n \geq 0} h.t^n.A | A$. This attacker can perform immediately any of its action before waiting for completion of other its actions. Moreover, the attacker tries different durations of the actions. Actually, there are infinitely many possible durations and it is not clear whether it is enough to check only finitely many of them.

In general, BNDC property is rather strong as it requires that process P is safe against attacks wherever an attacker A is placed. If a designer can decide which parts of the system are open for an attacker placing, it is enough to check only the appropriate openings. (For example, it might be the case that some subsystems are hardware circuits, parts with only one-way communications with the rest of the system and so on and hence placing attacker there is not possible.) In other words, a system might be considered to be safe even it does not enjoy BNDC property but there are other obstacles to put attacker A in such a position from where it could violate system safeness. To check this kind of "limited" BNDC property the designer has to specify the set of "reasonable" openings for P.

Assume that there exists A such that $(P'[A/\mathcal{A}]) \setminus HActt \not\approx P \setminus HActt$. It is still not clear whether this is the case of "ordinary" or of networked or of non-networked timing attack. To distinguish these cases we define t-weak bisimulation ($\approx_t$) in the style of Definition 3 but which neglects not only τ action but also t action. We will write $P \stackrel{x}{\Rightarrow}_t P'$ if $P(\stackrel{\tau,t}{\rightarrow})^* \stackrel{x}{\rightarrow} (\stackrel{\tau,t}{\rightarrow})^* P'$. Now we can define t-weak bisimulation.

Definition 5. *A binary relation $\Re \subseteq NTiCCS \times NTiCCS$ is called a* t-weak bisimulation *if it is symmetric and if $P \Re Q$ and $P \stackrel{x}{\rightarrow} P', x \in Actt$, then there exists Q' such that $Q \stackrel{\hat{x}}{\Rightarrow}_t Q'$ and $P' \Re Q'$. Two terms P, Q are* t-weakly bisimilar*, abbreviated $P \approx_t Q$, if there exists a t-weak bisimulation relating P and Q.*

Now we can formulate the definition of timing attack.

Definition 6. *A process A represents a* networked timing attack *(*non-networked timing attack*) for P if*

$$(P'[A/\mathcal{A}]) \setminus HActt \not\approx P \setminus HActt$$

for only some (for every) opening P' of P and, moreover, there exists A', $A' \approx_t A$ such that

$$(P'[A'/\mathcal{A}]) \setminus HActt \approx P \setminus HActt.$$

Note that the attack described by Example 1 is a networked timing attack. If we take $A' = h.Nil \mid h.Nil$ clearly $A' \approx_t A$ and we have $S'[A'/\mathcal{A}] \setminus HActt \approx S \setminus HActt$. On the other side if we take an opening S'' such that $S''[A'/\mathcal{A}] = A|[P_1 \parallel P_2]^1_{\{\emptyset\}}$ then $S''[A/\mathcal{A}] \setminus HActt \approx S \setminus HActt$.

To check whether a system (process) P is open to networked or non-networked timing attacks is even more complex than checking BNDC property due to infinitely many processes t-weak bisimilar to every attack A.

4 Conclusions and further work

Timing attacks can "break" systems which are often considered to be "unbreakable". More precisely, the attacks usually do not break system algorithms themselves but rather their bad, from security point of view, implementations. For example, such implementations, due to different optimizations, could result in dependency between time of computation and data to be processed, and as a consequence systems might become open to timing attacks. An attacker can deduce from time information also some information about private data, despite the fact that safe algorithms were used. Hence the importance of their study for privacy. In this paper we have presented a formal model which can describe timing attacks in computer networks, or more precisely, timing attacks which exploits network properties. This kind of attacks could not be modeled without networked calculus and hence now we can detect possibility of timing attacks which could not be detected otherwise. Moreover, with the help of timed and networked calculi we defined security properties for pure CCS which are finer than original BNDC property for CCS processes. On the top of this we can precisely distinguish between timing attacks in which nesting of an attacker plays or does not play role (networked and non-networked timing attacks). This approach enables us to formulate not only the question whether a system is robust with respect to timing attacks but also other questions: how precise must be the measure of time to perform a successful attack, how to modify the system in such a way that attacks will be not possible etc. In case of networked timing attack one possible prophylaxis is to protect non-secure openings against attackers - by firewalls, by redesign of the interconnection network and so on. In case of non-networked processes one possibility is to introduce some dummy idling around time sensitive actions etc. Note that there exists a software tool for NTiCCS (see [M94]) which can be used to check some of these properties.

We see our work as a first step towards an analysis of nested timing attacks on privacy. Further study will concern more efficient decision algorithms, more elaborated "nesting" of attacker in the network or more than one opening position for attackers (for example in the style of Trojan horse), different types of attackers, compositionally properties, and so on. Moreover in the presented paper we have used "Bisimulation-based Non-Deducibility on Composition" approach. High level processes are nested in the network and the attacker observes the system from the top-most level. There are also alternatives to this approach. Observations could be made from a lower level or the nested process might be of "mixed" level etc., but we have decided to use BNDC style as a starting point.

References

[CSFW03] Bossi A., D. Macedonio, C. Piazza, and S. Rossi: Secure contexts for confidential data. Proc. of the 16th IEEE Computer Security Foundations Workshop, CSFW 2003, IEEE Computer Society Press, 2003.

[DKL98] Dhem J.-F., F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater and J.-L. Willems: A practical implementation of the timing attack. Proc. of the Third Working Conference on Smart Card Research and Advanced Applications (CARDIS 1998), LNCS 1820, Springer, Berlin, 1998.

[FS00] Felten, E.W., and M.A. Schneider: Timing attacks on web privacy. Proc. 7^{th} ACM Conference on Computer and Communications Security, 25–32, 2000.

[FG01] Focardi, R. and R. Gorrieri: Classification of security properties. Part I: Information Flow. Foundations of Security Analysis and Design, LNCS 2171, Springer, Berlin, 2001, 331–396.

[FGM00] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[FGM03] Focardi, R., R. Gorrieri, and F. Martinelli: Real-Time information flow analysis. IEEE Journal on Selected Areas in Communications 21 (2003).

[GM01] Gruska D.P. and A. Maggiolo-Schettini: Process algebra for network communication. Fundamenta Informaticae 45(2001), 359–378.

[HH99] Handschuh H. and Howard M. Heys: A timing attack on RC5. Proc. Selected Areas in Cryptography, LNCS 1556, Springer, Berlin, 1999, 306–318.

[Ko96] Kocher P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. Proc. Advances in Cryptology - CRYPTO'96, LNCS 1109, Springer, Berlin, 1996, 104–113.

[M94] Martinelli, F.: "McTACTL: A tool to verify NTiCCS Specifications", *Technical Report TR-21/94*, Dipartimento di Informatica, Università di Pisa, 1994.

[M99] Martinelli, F.: Formal methods for the analysis of open systems with applications to security properties. Ph.D. Thesis, University of Siena, Feb.1999.

[M03] Martinelli, F.: Analysis of security protocols as open systems. Theoretical Computer Science 290(2003), 1057–1106.

[SWT01] Song. D., D. Wagner, and X. Tian: *Timing analysis of Keystrokes and SSH timing attacks*. Pro.10th USENIX Security Symposium, 2001.

SpyDer, a Security Model Checker *

G. Lenzini, S. Gnesi, and D. Latella

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI-CNR)
Area della Ricerca CNR - Via G. Moruzzi, 1 - Pisa (Italy)
fax: +39 050 315 2810
({gnesi, lenzini, latella}@isti.cnr.it)

Abstract. This paper presents SpyDer, a model checking environment for security protocols. In SpyDer a protocol is described as a term of a process algebra (called spy-calculus) consisting in a parallel composition of a finite number of, communicating and finite-behaviored, processes. Each process represents an instance of a protocol's role. The Dolev-Yao intruder is implicitly defined in the semantics of the calculus as an environment controlling all the communication events. Security properties are written as formulas of a linear-time temporal logic. Specifically the spy-calculus has a semantics based on labeled transition systems whose traces are the temporal model over which the satisfiability relation of the temporal logic is defined. The model checker algorithm is a depth first search that tests the satisfiability of a formula over all the traces, generated on-the fly, from a *typed* version of calculus. Here the use of types is finalized to obtain finite-state models, where the number of messages coming from the intruder is finite. Typed terms (*i.e.,* typed versions of a protocol description) are obtained at run-time by providing each variables with a sum of basic types. We prove that an attack over a typed version always implies the presence of an attack over the untyped version and, more interesting, that if there is an attack over a specification, a typing transformation catching the flaw exists.
The main contribution of the paper lives in the flexibility of the framework proposed. In fact the modeling environment is quite general and a protocol is specified once for all as an untyped spy-calculus term admitting infinite-state semantics. Then different typed versions, with finite-state semantics, may be obtained in a semi-automatic way from the same untyped specification just providing types constraints. Moreover the use of a logic allows the specification of a not fixed a priori class of properties.

Keywords: security protocols, security properties, process algebras, temporal logics, model checking.

1 Introduction

Past experiences have shown how formal methods can be successfully applied to the analysis of security protocols (*e.g.,* see [9, 27, 31, 1, 20]). For example se-

* This work was supported by the MIUR project SP4.

crecy, integrity and authenticity properties (see [2]) may be verified over protocols specification written in spi-calculus [3] (a process algebra derived from the π-calculus [25] with operators to encrypt and decrypt messages) via both symbolic trace analysis [15, 4, 5, 16, 6] and type checking over typed versions on the calculus [1, 20].

Here we propose a *logic-based model checking* [11] approach to the verification of security protocols expressed over a typed version of the spi-calculus dialect. Properties are expressed as logic formulas, whose satisfiability is checked over temporal models coming from the operational semantics of spy-calculus protocols. Model checking had been already applied in protocol security analysis (e.g., see [24, 22, 26, 30, 12, 19]), and originality of our approach is synthesized in the schemata in Figure 1. Let us suppose to have a finite term Z, representing a finite number of runs of a protocol involving a finite number of roles, and a security properties f to be checked on it. The semantics of Z is generally infinite-state due to the infinite amount of messages each role can receive from a potential intruder. In our scheme we model check the logic formula f over typed versions of the protocol whose semantics is made finite-state through the use of typed input variables. The use of types allows us to filter out those messages from the intruder that mismatch the required type during an input action. A typed version of a protocol is built at run-time, by providing a transformation function $\mathcal{C}_{\mathcal{S}_Z}$ that supplies a type for each variable appearing in Z. Different transformations, implying an exploration of different partitions of the whole state space, may be used in order to increase the confidence of the analysis.

The proposed approach has both advantages and disadvantages. For example treating with finite-state models we can check both safety (*i.e.,* secrecy and authenticity) and liveness (*i.e.,* anonymity) properties. In some model checker for security, such as the NRL [24] or Murϕ [26] for example, only safety properties are testable. Moreover the use of a logic to express properties adds flexibility in epressing properties. In Casper (*i.e.,* FDR) [23], for example the definition of some (eventually new) property class out of secrecy, authenticity or anonimity may not be easily formalized. On the other hand we cannot treat with unbound network, while for example NRL does. Recent works on data independence analysis and CSP [8] has shown that FDR (so also Casper) can be used to infer security results over of protocol managing infinite nounces, keys etc. from an analysis performed over a protocol that uses only a finite set of them. Our transformation functions may be used only to increase the confidence of the analysis when an unbound number of runs, instead of a small number, is considered. In fact, as theoretical results we prove that an attack (that is a trace that does not satisfy a formula f) over the model of a typed version of a protocol always implies the presence of an attack over the model of the corresponding untyped version of the protocol (soundness). On the contrary the existence of an attack over the untyped version of a protocol, does not implies the same attack over a specific typed version, but we prove that a transformation $\mathcal{C}'_{\mathcal{S}_Z}$ exists such that the typed version $\mathcal{C}'_{\mathcal{S}_Z}(Z)$ shows the same flaw (completeness). This latter result,

whose proof is not constructive, strongly depends on the fact that variables may be labeled with sum of types.

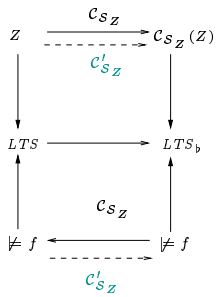

Fig. 1. The general scheme

The rest of the paper is so organized. Section 2 and Section 3 present syntax and semantics of our calculus and the logic used to specify properties. Section 4 introduces the typed spi-calculus dialect with its finite-state labeled transition semantics. Section 5 and Section 6 define protocol transformations which are used to obtain typed protocols from a generic specification and proves the main results of this paper. Section 7 formally describes the model checker algorithm whose correctness is based on the theory previously developed. Section 8 concludes. A running example is described along the paper in order to explain the approach in fact.

2 The Calculus

This section introduces our calculus, the *spy-calculus*, a process algebra whose syntax is inspired by the spi-calculus of Abadi and Gordon in [3]. By the way the semantics differs substantially as it will be discussed in the "semantics" paragraph.

Syntax. In the language we assume an infinite signature $\Sigma = \{N_i\}_{i\in\mathbb{N}} \cup \{\{\_\}\_, \langle \_, \_\rangle\}$, as usual in many process algebras for security (see for example [17, 5, 4]). So we have an infinite set of constants $\mathcal{N}$ (names) and two binary functions, $\{\_\}\_$ standing for encryption and $\langle \_, \_\rangle$ standing for pairing, respectively. The set $\mathcal{M}$ of *messages* is defined over Σ as the collection of messages M, M' containing at least $\mathcal{N}$ and such that $M, M' \in \mathcal{M}$ implies both $\langle M, M'\rangle \in \mathcal{M}$ and $\{M\}_{M'} \in \mathcal{M}$. Similarly the set $\mathcal{T}$ of *terms* is the collection of terms T, build over $\Sigma \cup \mathcal{V}$, where $\mathcal{V}$ is an infinite set $\mathcal{V} = \{x_i\}_{i\in\mathbb{N}}$ of variables. Finally we assume a set of $\mathcal{A}$ of labels a, and a finite set $\mathcal{I} = \{1, \ldots, n\}$ of integer identifiers i. Formally the spy-calculus syntax is defined by the following grammar:

$$\begin{array}{ll} \textit{protocols} & Z ::= Z \| Z' \mid (\nu N)Z \mid (i, P) \\ \textit{roles} & P ::= \mathbf{0} \mid a(x).P \mid \overline{a}(T).P \mid \underline{a}(T).P \mid P + P' \mid (\nu N)P \mid [x = T]P \end{array}$$

A *protocol* $Z = (\nu N_1) \ldots (\nu N_k)(1, P_1) \| \ldots \| (n, P_n)$ is the parallel composition of n role instances (i, P). New names may be introduced via the (νN) prefix. Each *role instance*, or agent, is composed by an identifier i, and by an acyclic *i.e.,* finite, sequential process (role) P which can communicate via a unique public channel (see later). The use of finite processes, if avoids infinite problems due to recursive processes, suffices to describe the behavior of a role running a security protocol (*e.g.,* see [10]). More instances, $(i_1, P), \ldots, (i_k, P)$, of the same role P can be used to represent, as usual in security protocol analysis, a principal running more sessions of the protocol (*e.g.,* see [30, 15, 5]). This means that we do not allow an unbound replication of processes, usually written as $!P$. This implies that we could not treat with infinite runs of a protocol.

Limiting the number of runs to be finite is a widely accepted assumption within the model checking approach to security (*e.g.,* see [13]), where a finite-state analysis is required. This is the case in most of the famous model checkers Murϕ [26], Brutus [12] or FDR [22], even if some of them limit the number of steps not at syntax level but only during the analysis. Instead a more significative observation must be done: by combining a limited number of runs and the fact that agents are finite-behaviored, we implicitly are assuming only a finite number of nounces, keys, process names etc. to be involved in a protocol. Again, this constraint is required in order to have finite-state analysis, although the analysis remains NP-Complete [21] even under these strict conditions. In [28, 8] Roscoe et. al have shown how, by the application of data independence techniques, it is possible to reduce the problem of proving security of a model where an infinite supply of different nounces, keys etc is required, to a finite check where only finite number of them are indeed involved. These techniques, ad-hoc proved within the CSP theory, would merit more attention but a restate of them within our framework is far from the scope of this work. To conclude this off-line discussion, we must observe that even the use of finite runs and finite number of nounces, keys etc. is not sufficient to obtain finite-state models, for an intruder may still generate data of an infinite length. How to cope with this last kind of infiniteness, which is the main topic of this paper, will be argument of Section 6.

A *role* P can be: (1) $\mathbf{0}$, the process that does nothing; (2-4) $a(x).P$, $\overline{a}(T).P$ or $\underline{a}(T).P$ are the processes ready to perform either an input over the variable x, an output or an *assertion*[1] of a term T, respectively. Here a is simply a label distinguishing among different actions (see later); (5) $P + P'$ is the non-deterministic choice between processes P and P'; (6) $(\nu N)P$ the process that generates a new local name N then used within P; finally (7) a *match* process

[1] Assertions, presented the first time by Woo and Lam in [32], are used here to introduce begin-events and end-events useful for specifying protocol authenticity properties.

$[x = T]P$ requires the term T to unify with the content of a variable x (possibly binding free variables in T) in order to proceed as P.

Each role P is usually equipped with a definition equation $p\langle \boldsymbol{x} \rangle \stackrel{def}{=} P$ where $\boldsymbol{x}$ is the tuple of all the free (i.e., not in the scope of any input or match primitive) variables x in P. In that case $p\langle \boldsymbol{N} \rangle$ is the same as $P[\frac{\boldsymbol{N}}{\boldsymbol{x}}]$, *i.e.,* the closed process where each free variable x in $\boldsymbol{x}$ is substituted with the corresponding name N in $\boldsymbol{N}$.

Example 1. As an example of specification in spy-calculus, let us consider the following key-exchange protocol, a simplified version of Needham-Schroeder Shared Key protocol (NSSK), where two principals A and B and a trusted server S are involved. Following a widely accepted informal notation NSSK may be described as in the following:

$$
\begin{array}{l}
1.\ A \rightarrow S : A, B \\
2.\ S \rightarrow A : \{K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}} \\
3.\ A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}
\end{array}
$$

One session of the protocol written in spy-calculus may be described as:

$$
\begin{array}{rcl}
NSSK & \stackrel{def}{=} & (\nu K_{as})(\nu K_{bs})(\nu A)(\nu B)(\nu S) \\
 & & (1, p_A\langle A, B, S, K_{as}\rangle) \| (2, p_B\langle A, B, S, K_{bs}\rangle) \\
 & & \| (3, p_S\langle A, B, K_{as}, K_{bs}\rangle) \\
p_A\langle a, b, s, k_{as}\rangle & \stackrel{def}{=} & \overline{c_{as}}(\langle a,\ b\rangle).c_{as}(x).[x = \langle \{x_1\}_{k_{as}}, x_2\rangle].\overline{c_{ab}}(x_2).\mathbf{0} \\
p_B\langle a, b, s, k_{bs}\rangle & \stackrel{def}{=} & c_{ab}(y).[y = \{\langle a,\ y_1\rangle\}_{k_{bs}}].\mathbf{0} \\
p_S\langle a, b, k_{as}, k_{bs}\rangle & \stackrel{def}{=} & (\nu K_{ab})c_{as}(z).[z = \langle a,\ b\rangle].\overline{c_{as}}(\langle \{K_{ab}\}_{k_{as}}, \{\langle a, k_{ab}\rangle\}_{k_{bs}}\rangle).\mathbf{0}
\end{array}
$$

The protocol specification shows three agents, one for each role running the protocol. Three different labels c_{ab}, c_{as} and c_{bs} are used to distinguish input/output actions over the unique public channel. The new shared keys K_{as} and K_{bs}, and the names of participants A, B and S are hidden to an external intruder, as explained in the semantics paragraph. □

Differently from the usual approach usually taken in process algebras for security (*e.g.,* see [2, 29]) the intruder is not explicitly described here as an additional process. Instead a Dolev-Yao [14] *intruder* Ω, will appear in the semantics as an *environment* having complete control of any communications. A similar approach has been taken in [7] where an environment-sensitive semantics is defined. The assumption of such an embedded intruder implies that, instead of a symmetric and synchronous communication paradigm via shared channels, our calculus uses asynchronous communications via a *unique* and anonymous channel, the intruder/environment. Moreover, having a unique public channel implies that whenever a process performs an output the message will be put

Expanding rules	Shrinking rules
$\frac{m \quad K}{\{M\}_K} \quad \frac{m \quad M}{\langle M, M' \rangle}$	$\frac{\{M\}_K \quad K}{M} \quad \frac{\langle M, M' \rangle}{M} \quad \frac{\langle M, M' \rangle}{M'}$

Fig. 2. Inference rules defining $\vdash$.

into the environment, while during a receiving a message is retrieved from the one's in the environment's *knowledge i.e.,* from the ones that can be composed using the messages standing in the environment. In this way we want to simulate any possible unpredictable intruder's attack. As a formal definition of *knowledge* we can assume the following:

Definition 1 (Knowledge). *Let $W \subseteq \mathcal{M}$ be a finite set of messages. The* knowledge *of W, written $KS(W)$, is the set $W \cup \{M : W \vdash^* M\}$.*

where $\vdash$ is the derivation symbol defined by a proof system modeling specific intruder capabilites in managing messages. Here we require that the proof system is such that the question $M \in KS(W)$ is decidable and derivable as, for example, the one in Figure 2 (see [16]).

Semantics. The operational semantics of spy-calculus is given in term of labeled transition systems (LTS). States of the LTS are couples $\langle \mathcal{G}, Z \rangle$, of *global states* and *protocols*. A global state holds the *local state* of each agent plus the one of the intruder. A local state contains the name p of the role played by the agent, the set W of messages it has received so far, and a variables/values mapping σ of its variables. Formally:

Definition 2 (Global State). *Supposing n process instances involved in a protocol, a* global state *$\mathcal{G}$ is an array of $n+1$ local states. $\mathcal{G}(0)$ is the local state of the intruder/environment Ω, and $\mathcal{G}(i)$ the local state of process instance identified by i.*

We write *Glob* to indicate the set of all global states.

Definition 3 (Local State). *A* local state *l, is triple (p, W, σ) where $p \in \mathcal{N}$ is a name, $W \subseteq \mathcal{M}$ is a set of messages and $\sigma : \mathcal{V} \longrightarrow \mathcal{M}$ is a function from variables to messages*[2].

[2] With the symbol $\perp$ we indicate the function σ undefined everywhere, while we write $\sigma' \sqsupseteq \sigma$ whenever the function σ' coincides with σ in every values of the domain where σ is defined. Moreover we write $\sigma(x) = \sigma'(x)$ if both functions are undefined over x and coincide in x, and finally with $\widehat{\sigma}(T)$ we indicate the message obtained substituting each variable x in T with the corresponding value $\sigma(x)$. The test $\widehat{\sigma}(T) = \widehat{\sigma}(S)$ evaluates true if and only if both functions return the same ground message, false otherwise.

Giving a global state $\mathcal{G}$, we indicate with p_i, W_i, and σ_i respectively, the items of the local state $\mathcal{G}(i)$, for $i = 1, \ldots, n$. In the following W_Ω is used instead of W_0 to point out the set of messages lying in the environment. It is worth to underline that in the operational semantics each W_i and σ_i has a monotone growth rate along the protocol execution. This implies that a variable is bound by the first (outermost) input prefix or matching operator in which it appears.

Transition rules of the LTS describe how a state $\langle \mathcal{G},\ Z \rangle$ evolves as a consequence of an *action* α. An action is composed by the identifier i of the agent performing the action and, if the action is a visible action (*i.e.*, , input, output or assertion), by the relative label a and by the messages M involved. Formally:

Definition 4 (Action). *An* action α *is either: (a)* $i.a\langle M \rangle$ *or* $i.\overline{a}\langle M \rangle$ *or* $i.\underline{a}\langle M \rangle$ *meaning that instance* i *has executed, respectively an input, output or assertion, labeled* a *over the message* M*; (b)* τ*, the internal action.*

We write Act_τ to indicate the set of all actions. The next definition resumes the whole operational semantics:

Definition 5 (Operational Semantics).
Let $Z = (\nu N_1) \ldots (\nu N_k)(1, P_1) \| \ldots \| (n, P_n)$*, be a protocol. The associated LTS is a tuple* $(\mathcal{Q}_Z, \langle \mathcal{G}_0, Z \rangle, Act, \mathcal{R}_Z)$*, where: (a)* $\mathcal{Q}_Z \subseteq Glob \times \mathcal{Z}$ *is the set of states; (b)* $\langle \mathcal{G}_0, Z \rangle$ *is the initial state so defined:*

- $\mathcal{G}_0(0) = (\Omega, W_\Omega, \bot)$*, is the initial local state of the environment, where* Ω *is the name of the environment,* W_Ω *is the initial set of messages known by the environment composed by all the free (i.e. not in the scope of* ν*) names in* Z*.*
- $\mathcal{G}_0(i) = (p_i, W_i, \bot)$ *for all* $i \in \mathcal{I}$*, is the initial local state of the role instance* i*. More precisely,* p_i *is the name of the process* P*, if* $p_i \stackrel{def}{=} P$*,* W_i *is the set of messages* p_i *knows, composed by all the free names in* Z *and by all the free names in* P_i*.*

(c) Act_τ *is the set of actions; (d)* $\mathcal{R}_Z \subseteq \mathcal{Q}_Z \times Act \times \mathcal{Q}_Z$ *is the transition relation defined by the rules in Figure 3 . Whenever* $(q, \alpha, q') \in \mathcal{R}_Z$ *we will write* $q \xrightarrow{\alpha} q'$*.*

Referring to the definition of rule $\mathbf{r}_?$ in Figure 3, note that even if all the premises are decidable, the number of messages in $KS(W_\Omega)$ is generally infinite. This produces infinite branching *in absence of limiting strategies.* We will describe one of such strategies in Section 5. In the definition of the rule $\mathbf{r}_=$ we require that the term $\widehat{\sigma}(T)$ satisfies the condition $\mathcal{P}(\_)$, that informally means that $\widehat{\sigma}(T)$ does not contain variables as decryption keys. This will avoid unfair specifications where encryption could be broken simply by using pattern matching. Anyway $\mathcal{P}(\_)$ is only a syntactic constraint, whose presence or not does not interfere with our analysis.

$$\frac{\langle \mathcal{G},\ Z' \rangle \xrightarrow{\alpha} \langle \mathcal{G}',\ Z'' \rangle}{\langle \mathcal{G},\ Z \| Z' \rangle \xrightarrow{\alpha} \langle \mathcal{G}',\ Z \| Z'' \rangle}\ \mathbf{r}_{\equiv 1} \qquad \frac{\sigma_i(x) \neq \perp,\ \mathcal{P}(\widehat{\sigma}_i(T)),\ \exists \sigma' \sqsupseteq \sigma_i : \widehat{\sigma}'(T) = \sigma_i(x)}{\langle \mathcal{G},\ Z \| (i, [x\ =\ T].P) \rangle \xrightarrow{\tau} \langle \mathcal{G}[\frac{\sigma}{\sigma'}],\ Z \| P \rangle}\ \mathbf{r}_{=}$$

$$\frac{\widehat{\sigma}_i(T) = M}{\langle \mathcal{G},\ Z \| (i, \overline{a}(T).P) \rangle \xrightarrow{\overline{a}\langle M \rangle} \langle \mathcal{G}[\frac{W_\Omega \cup \{M\}}{W_\Omega}],\ Z \| (i, P) \rangle}\ \mathbf{r}_{!}$$

$$\frac{\sigma_i(x) = \perp\ ,\ \exists \sigma' \sqsupseteq \sigma_i : \sigma'(x) = M \in KS(W_\Omega)}{\langle \mathcal{G},\ Z \| (i, a(x).P) \rangle \xrightarrow{a\langle M \rangle} \langle \mathcal{G}[\frac{W_i \cup \{M\}}{W_i}][\frac{\sigma'}{\sigma_i}],\ Z \| (i, P) \rangle}\ \mathbf{r}_{?}$$

$$\frac{\langle \mathcal{G},\ Z \| (i, P) \rangle \xrightarrow{\alpha} \langle \mathcal{G}',\ Z \| (i, P'') \rangle}{\langle \mathcal{G},\ Z \| (i, P + P') \rangle \xrightarrow{\alpha} \langle \mathcal{G}',\ Z \| (i, P'') \rangle}\ \mathbf{r}_{+1} \qquad \frac{\widehat{\sigma}_i(T) = M}{\langle \mathcal{G},\ Z \| (i, \underline{a}(T).P) \rangle \xrightarrow{\underline{a}\langle M \rangle} \langle \mathcal{G},\ Z \| (i, P) \rangle}\ \mathbf{r}_{!!}$$

$$\frac{}{\langle \mathcal{G},\ Z \| (i, (\nu N)P) \rangle \xrightarrow{\tau} \langle \mathcal{G}',\ Z \| (i, P[\frac{N'}{N}]) \rangle}\ \mathbf{r}_{\nu \mathbf{P}} \qquad \frac{}{\langle \mathcal{G},\ Z \| (\nu N)Z' \rangle \xrightarrow{\alpha} \langle \mathcal{G}',\ Z \| Z'[\frac{N'}{N}] \rangle}\ \mathbf{r}_{\nu \mathbf{Z}}$$

Fig. 3. *LTS* transition rules. The obvious rules $\mathbf{r}_{\equiv 2}$ and $\mathbf{r}_{+2}$ symmetric of $\mathbf{r}_{\equiv 1}$ and $\mathbf{r}_{+1}$ are left out.

3 A Logic for Security Properties

This section shows how a linear time temporal logic can be used to express security properties. Different logic for security can be found in literature (*e.g.,* see [9, 18]). Here we borrow the logic definition from [12], because that logic has been proved to be able to express a large set of properties, from secrecy, authenticity, general correspondence properties, and some weak form of anonymity. The syntax of the logic, presented in Figure 4, shares the same term signature $\Sigma \cup \mathcal{V}$ with the spy-calculus. We remind some basic definition:

A term ς it can be (a) **name**(s), the name of an agent's identifier s, (b) $s.T$ the message term T interpreted in the local state of the agent s or (c) a ground message $M \in \mathcal{M}$.

An atomic proposition ρ it can be (a) **Knows**(s, ς), a predicate on the fact that agent s "knows" the term ς; (b) **Acts**$_\lambda(s, \varsigma)$, a predicate on the action λ, performed by agent s and involving the term ς; (c) $\varsigma = \varsigma'$, is an equality test over terms.

A Formula f it can be any propositional logic formula, or the modal formula $\Diamond_P f$, where the symbol $\Diamond_P$ is the modal operator *eventually* in its past interpretation. The formula $\exists s.f$ binds the agent variable s, ranging over the set of agent's identifier $\mathcal{I}$, within the formula f.

Referring to the Example 1, we can express an authenticity property as the following formula:

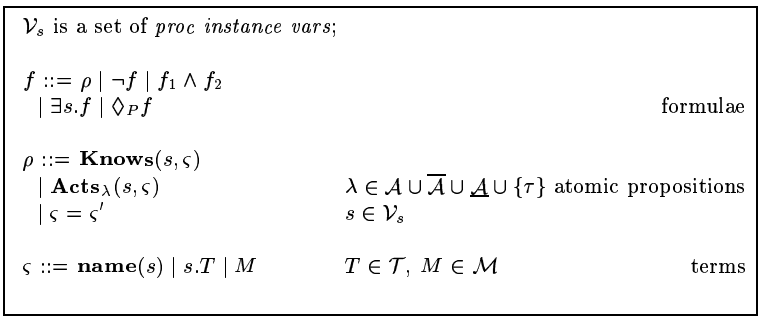

Fig. 4. Logic Syntax (see also Section 2 for definitions of $\mathcal{I}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{V}$).

$$f \stackrel{def}{=} \forall b.\, \exists a.\, \mathbf{name}(b) = B \;\wedge\; \mathbf{name}(a) = A \;\wedge \\ \mathbf{Acts}_{c_{ab}}(b, b.y) \rightarrow \Diamond_P(\mathbf{Acts}_{\overline{c_{ab}}}(a, a.x_2)) \wedge (b.y = a.x_2) \tag{1}$$

Informally the formula (1) says that whenever an agent (running the role of B) receives a message in y (as a consequence of an input action labeled c_{ab}) then there exists an agent (running the role A) that has previously sent a message to B (by an action labeled c_{ab}). Additionally (1) requires the two messages to be equal. Note that the use of $\forall$ and $\exists$ is here syntactic sugar for a finite sequence of respectively $\wedge$ and $\vee$.

The interpretation of atomic propositions and formulae is defined over *traces* $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$ coming from a labeled transition system belonging to LTS, where $q_i = \langle \mathcal{G}_i,\, Z_i \rangle$, and $q_i \stackrel{\alpha_{i+1}}{\longmapsto} q_{i+1}$ is a possible transition from q_i. In other words a trace is the temporal structure over which the satisfiability of a formula is checked. Formally:

Definition 6 (Message Term Interpretation). *Given a global state $\mathcal{G}$, and a term T, the* term interpretation*, is the function $\mathbb{M} : Glob \rightarrow \mathcal{T} \rightarrow \mathcal{M} \cup \{\perp\}$ given below:*

$$\begin{array}{lll} \mathbb{M}(\mathcal{G})(M) & = M, & \textit{where } M \in \mathcal{M} \\ \mathbb{M}(\mathcal{G})(j.T) & = \widehat{\sigma}(T), & \textit{if } \mathcal{G}(j) = (\_, \_, \sigma) \\ \mathbb{M}(\mathcal{G})(\mathbf{name}(j)) & = p, & \textit{if } \mathcal{G}(j) = (p, \_, \_) \end{array}$$

Informally, the interpretation of $j.T$ is the ground message obtained instantiating all the variables appearing in T, using the set of bindings of the process instance whose identifier is j. The interpretation of $\mathbf{name}(j)$ is the message which represents the name of the process instance whose identifier is j.

The interpretation of atomic propositions ρ (resp., formulae f) over a single state $q_i = \langle \mathcal{G}_i,\, Z_i \rangle$ is written $\langle \pi,\, i \rangle \models \rho$ (resp., $\langle \pi,\, i \rangle \models f$). They are defined as:

Definition 7 (Atomic Proposition Interpretation). *Given a trace* $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$*, we have, for* $1 \leq i \leq n$*:*

$$\begin{array}{ll} \langle \pi, i \rangle \models \varsigma = \varsigma' & \textit{iff } \mathbb{M}(\mathcal{G}_i)(\varsigma) = \mathbb{M}(\mathcal{G}_i)(\varsigma') \\ \langle \pi, i \rangle \models \mathbf{Knows}(j, \varsigma) & \textit{iff } \mathbb{M}(\mathcal{G}_i)(\varsigma) \in KS(W_j), \textit{ if } \mathcal{G}_i(j) = (\_, W_j, \_) \\ \langle \pi, i \rangle \models \mathbf{Acts}_\lambda(j, \varsigma) & \textit{iff } \alpha_i = j.\lambda(M), \textit{ where } M = \mathbb{M}(\mathcal{G}_i)(\varsigma) \end{array}$$

Definition 8 (Formulae Interpretation). *Given a formula* f *and a trace* $\pi = q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$*, we have that for* $1 \leq i \leq n$*:*

$$\begin{array}{ll} \langle \pi, i \rangle \models \rho & \textit{iff } \langle \pi, i \rangle \models \rho \\ \langle \pi, i \rangle \models \neg f & \textit{iff } \langle \pi, i \rangle \not\models f \\ \langle \pi, i \rangle \models f_1 \wedge f_2 & \textit{iff } \langle \pi, i \rangle \models f_1 \textit{ and } \langle \pi, i \rangle \models f_2 \\ \langle \pi, i \rangle \models \exists s.f & \textit{iff } \textit{ there exists } j \in \mathcal{I} : \langle \pi, i \rangle \models f[\frac{j}{s}] \\ \langle \pi, i \rangle \models \Diamond_P f & \textit{iff } \textit{ there exists } j, 0 \leq j \leq i : \langle \pi, j \rangle \models f \end{array}$$

The obvious extension of satisfiability over a trace is defined as follows $\pi \models f$ iff $\langle \pi, i \rangle \models f, \forall i : 0 \leq i \leq \text{length}(\pi)$. Finally we say that a protocol Z satisfies a formula f, written $Z \models f$, if f is satisfied over all the traces of the LTS, semantics of Z.

4 Typed spy-calculus

In Section 2 it has been pointed out that the use of $KS(W_\Omega)$ generally creates an infinite amount of input transitions but, in order to perform model checking, we need finite-state LTS. This Section studies the possibility of using *type information* (prevalently in input actions) in order to select only those messages of a certain type among the ones in the environment's knowledge. It is worth to underline that the use of types is here oriented to obtain finite-state models.

Definition 9 (Type). *A* type t*, is either: (1) a* basic type `proc`, `key`, `nounce` *or* `atom`*, (2) a* pair type $\langle\!|\, t, t \,|\!\rangle$*, (3) a* crypto type $\{\!|\, t \,|\!\}_t$*, a* sum of types $t \oplus t'$*. With* $\mathbb{T}$ *we mean the set of all possibles types.*

Types sort is partially ordered, with the following ordering relation:

Definition 10. *Let* $t, t' \in \mathbb{T}$*. We say that* $t \leq t'$ *iff*

$$\begin{cases} t = t' \textit{ or } \left(t' = t'_1 \oplus t'_2 \textit{ and } (t \leq t'_1 \textit{ or } t \leq t'_2)\right) \\ t = \{\!|\, t_1 \,|\!\}_{t_2} \textit{ and } t' = \{\!|\, t'_1 \,|\!\}_{t'_2} \textit{ and } t_1 \leq t'_1 \textit{ and } t_2 \leq t'_2 \\ t = \langle\!|\, t_1, t_2 \,|\!\rangle \textit{ and } t' = \langle\!|\, t'_1, t'_2 \,|\!\rangle \textit{ and } t_1 \leq t'_1 \textit{ and } t_2 \leq t'_2 \end{cases}$$

Types will be used in a light different version of the spy-calculus, called *spyD-calculus*, where names and variables are provided with a type, and whose syntax is expressed with the following grammar:

$$\begin{array}{ll} \textit{protocols} & \tilde{Z} ::= \tilde{Z} \| \tilde{Z'} \mid (\nu N)\tilde{Z} \mid (i, \tilde{P}) \\ \textit{roles} & P ::= \mathbf{0} \mid a(x:t).\tilde{P} \mid \overline{a}(\tilde{T}).\tilde{P} \mid \underline{a}(\tilde{T}).\tilde{P} \mid \\ & \qquad \tilde{P} + \tilde{P'} \mid (\nu N)\tilde{P} \mid [x:t = \tilde{T}]\tilde{P} \end{array}$$

The spyD-calculus syntax differs because of: (a) the set of names $\mathcal{N}$ is partitioned among $\mathcal{C}, \mathcal{K}, \mathcal{O}$ and $\mathcal{Z}$ of resp., atomic messages, process names, keys and nounces/timestamps. We assume that data from $\mathcal{C}$ are of type `proc`, data from $\mathcal{K}$ are of type `key`, data from $\mathcal{O}$ are of data `nounce` and data from $\mathcal{Z}$ are of type `atom`;

(b) variables are indeed written as *typed variables* $x : t$, where x is a variable and t is a type. The set $\tilde{\mathcal{T}}$ of *typed terms* $\tilde{T}$, is then build, over the signature $\Sigma \cup (\mathcal{V} \times \mathbb{T})$. Protocols $\tilde{Z}$ and roles $\tilde{P}$ remain almost unchanged but typed terms are used instead of terms. We assume that only *well typed* protocols are possible, meaning that all instances of a variable bound by the same outermost, input primitive or match, must have the same type.

Starting from basic types, explicitly assigned to names and variables, a *top level type* $\lfloor \tilde{T} \rfloor$ for a typed term $\tilde{T}$ can be easily deduced by structural induction over terms[3]. In the following we say that a message term $\tilde{T}$ has type t if $\lfloor \tilde{T} \rfloor = t$, and we write $|t|$ to indicate the number of basic types appearing in t. Using types we mean to define a bound version of the environment knowledge.

Definition 11 (Bound Knowledge). *Let $W \subset \mathcal{M}$ be a finite set of messages, and t a type. Then $KS^{(t)}$ called the* t-knowledge *is the following set of messages:*

$$KS^{(t)}(W) = \{M \in KS(W) : \lfloor M \rfloor \leq t\}$$

About bound knowledge the following results hold:

Lemma 1. *Let W a finite set of messages. Then for every $t \in \mathbb{T}$, $KS^{(t)}(W)$ is a finite set.*

Lemma 2. *Let W be finite. Given a message M, the question $M \in KS^{(t)}(W)$ is decidable.*

We are now ready to restate the operational semantics of spyD-calculus in order to use $KS^{(t)}(W)$ in input rule. In fact, spyD-calculus semantics is based on labeled transition systems, we call $LTS_\flat$, whose definition is almost the same of the one of LTS (see Definition 5) with the exception of the transition rule $\mathbf{r}_?$, which is re-defined as the rule $\tilde{\mathbf{r}}_?$:

$$\frac{\sigma_i(x) = \perp \ , \ \exists \sigma' \sqsupseteq \sigma_i : \sigma'(x) = M \in KS^{(t)}(W_\Omega)}{\langle \mathcal{G},\ Z \| (i, a(x:t).P) \rangle \xrightarrow{a\langle M \rangle}_\flat \langle \mathcal{G}[\frac{\kappa_i \cup \{M\}}{\kappa_i}][\frac{\sigma'}{\sigma_i}],\ Z \| (i, P) \rangle}\ \tilde{\mathbf{r}}_?$$

In $\tilde{\mathbf{r}}_?$ the bound knowledge $KS^{(t)}$ is used instead of KS. We write $\langle \mathcal{G},\ \tilde{Z} \rangle \xrightarrow{\alpha}_\flat \langle \mathcal{G}',\ \tilde{Z}' \rangle$ to say that a typed protocol $\tilde{Z}$, and the global state $\mathcal{G}$, change as a consequence of action α. It easily follows that:

[3] With a little abuse of notation, the same function symbol $\lfloor \_ \rfloor$ is used for the either the function returning a type given a message M or the function returning a type given a typed term $\tilde{T}$.

Theorem 1. *Given any typed protocol Z the corresponding labeled transition system is finite-state.*

Logic formulae (see Section 3) may be interpreted over traces $\pi_\flat = \rho_0 \cdot \alpha_1 \cdots \alpha_n \cdot \rho_n$, coming from $LTS_\flat$, exactly in the same way they are over traces π coming from LTS. In fact, the satisfiability relation (see Section 3) is defined over a global state *Glob* of a state $\rho = \langle \mathcal{G}, \tilde{Z} \rangle$, whose definition is unchanged.

5 Building Typed Protocols

In this section we explain how to obtain, in a semi-automated way, typed spyD-calculus protocols starting from a spy-calculus specification. In addition we want at least that an attack over the spyD-calculus protocol implies the presence of an attack over the corresponding spy-calculus protocol. In the following we require protocols to be written in *normal form* where all the variables in different role are distinct and, within a role, variables bound by different input primitives or match are different.

Definition 12 (Transformation). *A* transformation *is a partial functions $\mathcal{S} : \mathcal{V} \to \mathcal{V} \times \mathbb{T}$, such that, for all x in the domain of $\mathcal{S}$, $\mathcal{S}(x) = x : t$ where t is a type.*

Based on a transformation $\mathcal{S}$ we can define a mapping $\mathcal{C}_\mathcal{S}$, that simply is a structural extension of $\mathcal{S}$ from variables to message terms, processes and protocols. Although different symbols should be used for those extensions we will used the same symbol $\mathcal{C}_\mathcal{S}$ for all of them. The general idea is that, given a protocol Z, $\mathcal{C}_\mathcal{S}(Z)$ is a typed version of Z with finite-state semantics. To be precise, more than general transformations we are interested on transformations whose domain is the set of variables of a given protocol. Formally:

Definition 13 (Transformation with respect to a Protocol). *Let Z be a protocol specification in normal form. A* transformation with respect to Z *is a transformation $\mathcal{S}_Z$ whose domain is exactly the set variables appearing in Z.*

We can observe that given a protocol Z and a protocol transformation $\mathcal{S}_Z$, the set of bound and free names of Z and the ones of $\mathcal{C}_{\mathcal{S}_Z}(Z)$ is the same. In fact a protocol transformation acts only over variables, while names are left untouched. The same can be observed for bound and free names of processes P's, involved in Z, and processes $\mathcal{C}_{\mathcal{S}_Z}(P)$'s involved in $\mathcal{C}_{\mathcal{S}_Z}(Z)$.

Let us again refer to Example 1, and suppose to have defined the following protocol transformation function, over the $NSSK$:

$$\mathcal{S}_{NSSK}(v) \stackrel{def}{=} \begin{cases} v = x, & x : \langle\!| \{\!| \texttt{key} |\!\}_{\texttt{key}}, \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} |\!\rangle \\ v = x_1, & x_1 : \texttt{key} \\ v = x_2, & x_2 : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} \\ v = y, & y : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} \\ v = y_1, & y_1 : \texttt{key} \\ v = z, & z : \langle\!| \texttt{proc}, \texttt{proc} |\!\rangle \\ \text{else}, & \bot \end{cases}$$

The transformation is built following the intuition that messages received in variable y from agent B indeed are messages of type $\{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}}$, but any other transformation can be defined. The typed version of the protocol is then the following:

$$\begin{aligned}
\mathcal{C}_{NSSK}(NSSK) \stackrel{def}{=}\ & (\nu K_{as})(\nu K_{bs})(\nu A)(\nu B)(\nu S) \\
& (1, p_A\langle A, B, S, K_{as}\rangle) \| (2, p_B\langle A, B, S, K_{bs}\rangle) \\
& \| (3, p_S\langle A, B, K_{as}, K_{bs}\rangle) \\
p_A\langle a, b, s, k_{as}\rangle \stackrel{def}{=}\ & \overline{c_{as}}(\langle a, b\rangle).c_{as}(x : \langle\!| \{\!| \texttt{key} |\!\}_{\texttt{key}}, \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} |\!\rangle) \\
& [x : \langle\!| \{\!| \texttt{key} |\!\}_{\texttt{key}}, \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} |\!\rangle \\
& \langle \{x_1 : \texttt{key}\}_{k_{as}}, x_2 : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} \rangle]. \\
& \overline{c_{ab}}(x_2 : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}}).\mathbf{0} \\
p_B\langle a, b, s, k_{bs}\rangle \stackrel{def}{=}\ & c_{ab}(y : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}}) \\
& [y : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}} = \{\langle A, y_1 : \{\!| \langle\!| \texttt{proc}, \texttt{key} |\!\rangle |\!\}_{\texttt{key}}\rangle\}_{K_{bs}}].\mathbf{0} \\
p_S\langle a, b, k_{as}, k_{bs}\rangle \stackrel{def}{=}\ & c_{as}(z : \langle\!| \texttt{proc}, \texttt{proc} |\!\rangle)[z : \langle\!| \texttt{proc}, \texttt{proc} |\!\rangle = \langle a, b\rangle]. \\
& (\nu K_{ab})\overline{c_{as}}(\langle \{K_{ab}\}_{k_{as}}, \{\langle a, K_{ab}\rangle\}_{k_{bs}}\rangle).\mathbf{0}
\end{aligned}$$

6 Towards a Finite Model Checking

After having proved that (typed) spyD-calculus protocols can be obtained from (untyped) spy-calculus specifications in a semi-automatic way via the use of transformation functions, we want to inquire into the relationship between the relative semantics, *i.e.,* between $LTS_\flat$ and LTS. This section formally shows that there exists at least a trace inclusion relation. In particular, given a transformation $\mathcal{S}_Z$, the set of traces from LTS always includes the set of traces from $LTS_\flat$. This is a surprise, but a more interesting result is that given any trace of LTS a transformation can always be defined in order to have the same trace in $LTS_\flat$. Unfortunately the proof of this latter result is not constructive and we have no general methods to construct this abstraction. We conjecture that a static analysis of the message flow along a protocol specification may help in defining significative transformations , but we have not yet enquired in this direction.

The following definitions define the basic equivalence relations among global states and traces of the relative transition systems.

Definition 14. *Given a transformation function $\mathcal{S}$, let $q = \langle \mathcal{G}, P \rangle$ be a state of a LTS, and $\rho = \langle \mathcal{G}', \tilde{P}' \rangle$ a state of a $LTS_\flat$ We say that they are* equal up to $\mathcal{S}$ *and we write $q =_{\mathcal{S}} \rho$, iff (a) $\mathcal{G} = \mathcal{G}'$; (b) $\mathcal{C}_{\mathcal{S}}(P) = \tilde{P}'$ where the symbol $=$ must be intended as an identity.*

Definition 15. *Given a transformation function $\mathcal{S}$, let $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, and $\pi_\flat = \rho_0 \cdot \alpha'_1 \cdot \rho_1 \cdot \ldots \cdot \alpha'_n \cdot \rho_n$ be two traces. We say that they are* equal up to $\mathcal{S}$ *and we write $\pi =_{\mathcal{S}} \pi_\flat$ iff for all k (a) $\alpha_k = \alpha'_k$; (b) $q_k =_{\mathcal{S}} \rho_k$.*

In the following we intend to investigate in an order relation between LTS models of a protocol Z, with the $LTS_\flat$ models of $\mathcal{C}_{\mathcal{S}_Z}(Z)$.

Lemma 3. *Let suppose that Z is protocol specification in normal form, and $\mathcal{S}_Z$ a protocol transformation with respect to Z. Let be Π_Z the set of traces from the LTS, semantics of Z, and let be $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ the set of traces of the $LTS_\flat$, semantics of the protocol $\mathcal{C}_{\mathcal{S}_Z}(Z)$. Then for each trace $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ there exists a trace $\pi \in \Pi_Z$ such that $\pi =_{\mathcal{S}_Z} \pi_\flat$.*

Lemma 4. *Let suppose that Z is a protocol in normal form, and Π_Z be the set of traces from the LTS, semantics of Z. Then for each trace $\pi \in \Pi_Z$ there exists a transformation S such that in the set of traces $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ of transformed protocol $\mathcal{C}_{\mathcal{S}_Z}(Z)$ there is a trace $\pi_\flat$ such that $\pi_\flat =_{\mathcal{S}_Z} \pi$.*

Now we analyze the impact of protocol transformation on the satisfiability of formulae. In particular given a protocol specification Z, a protocol transformation $\mathcal{S}_Z$, and a formula f to be checked, we want to understand if f can be checked over $\mathcal{C}_{\mathcal{S}_Z}(Z)$. The answer is given by the following theorems.

Theorem 2. *Given a protocol Z, a transformation $\mathcal{S}_Z$ over Z, and a a logic formula f. Let be Π_Z the set of traces from the LTS, semantics of Z, and let be $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ the set of traces of the $LTS_\flat$, semantics of the protocol $\mathcal{C}_{\mathcal{S}_Z}(Z)$, and let be $\pi \in \Pi_Z$ and $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ such that $\pi =_{\mathcal{S}_Z} \pi_\flat$. Then $\pi \models f$ iff $\pi_\flat \models \mathcal{C}_{\mathcal{S}_Z}(f)$.*

Theorem 3. *Given a protocol Z and a protocol transformation $\mathcal{S}_Z$ over Z, let us suppose that f is a logic formula. Then $\mathcal{C}_{\mathcal{S}_Z}(Z) \not\models f$ implies $Z \not\models f$.*

Theorem 4. *Given a protocol Z and a f such that $Z \not\models f$. Then there exists a protocol transformation $\mathcal{S}_Z$ such that $\mathcal{C}_{\mathcal{S}_Z}(Z) \not\models f$.*

Note that Theorem 4 is possible because variables may assume, via $\mathcal{S}_Z$ potentially any types. This makes our type system too general for being used into any type checking framework, but for the same reason type flaw attacks are still possible in our framework.

7 Model Checking

This section presents *spyDer*, the model checking framework we intend to use for verifying security. The kernel of spyDer is implemented by Algorithm 1,

a simple procedure which visits a finite labeled transition system in a depth first search mode. As parameters the algorithm requires a closed protocol Z in normal form, a protocol transformation $\mathcal{S}_Z$, and a formula f to be checked. Informally Algorithm 1 uses two stacks: (a) a stack $\mathcal{S}_A$ containing transitions that implements the depth first visit of the transition system; (b) a stack $\mathcal{S}_\Pi$ for storing prefixes of traces. In particular during the depth first traversal of transitions, prefixes $\rho_0 \cdot \alpha_1 \cdots \alpha_i \cdot \rho_i$ are built and the satisfiability of the formula f is checked over the state q_i. The procedure stops with a counterexample if f is discovered to be unsatisfied in some ρ_i. Algorithm 1 has time complexity $O(|V| \cdot (|f| + |W_{max}| \cdot |M_{max}|))$ where $|V|$ is the number of states of the transition system, $|f|$ is the length of formula f, W_{max} is the greatest W_Ω and M_{max} is the biggest message used within the protocol. It is worth to underline that use of the protocol transformation $\mathcal{S}_Z$ limits the number of traces of the model to a finite number, even if it remains exponential (an exponential number of messages is involved in input actions). This means that our model checker runs in exponential time w.r.t the length of messages involved in the protocol. Anyway, being the depth of (the tree representing) the $LTS_\flat$ linear in the agents steps, by using an on-the-fly generation strategy (in real implementation step 13 is indeed on-demand) and depth first search requires linear space.

8 Conclusions

This paper has presented spyDer, a model checking environment for a typed spi-calculus dialect. A protocol is specified as a term of a formal calculus called spy-calculus describing a parallel composition of a finite number of process instances, each representing a finite-behaviored role running the protocol. More runs a protocol can be described by instantiating more copies of each agent. The spy-calculus has been provided with an operational semantics based on labeled transition systems, where the intruder is described in the Dolev-Yao style. Security properties can be expressed in a linear time temporal logic.

Here types are used to obtain finite-state labeled transition systems. Moreover they are inserted in a flexible framework, where user-defined transformation functions $\mathcal{C}_{\mathcal{S}_Z}$ are applied to a given protocol specification Z to obtain a typed version in $\mathcal{C}_{\mathcal{S}_Z}(Z)$, before running the model checker.

A transformation $\mathcal{C}_{\mathcal{S}_Z}$ is indeed a "view" that a user can introduce to select a finite partition of the state space. Different transformations can be used in order to select different portion of the state space, increasing the confidence of the analysis. As theoretical results we have proven that given a formula f, an attack (that is a trace over which f is not satisfied) over a transformed protocol $\mathcal{C}_{\mathcal{S}_Z}(Z)$ always implies the existence of the same attack over the original protocol Z. Obviously finding an attack via protocol transformations is only a sound methods, *i.e.,* if an attack exists over a protocol Z we have no guarantee that the attack can be found over a specific transformed protocol. Anyway, we prove that a transformation $\mathcal{C}'_{\mathcal{S}_Z}$ that preserves the attack always exists. This mean that our transformations are general enough to maintain type flaws.

Algorithm 1 Model Checking (Z a closed protocol, f formula, S)

1: $\tilde{Z} \leftarrow \mathcal{C}_{\mathcal{S}_Z}(Z)$ {Returned a typed protocol, using $\mathcal{S}_Z$}
2: $\rho_0 = \langle \mathcal{G}_0, \tilde{Z} \rangle$ {Set the initial state}
3: $\mathcal{S}_\Lambda \leftarrow \emptyset$ {an empty stack for containing actions}
4: $\mathcal{S}_\Pi \leftarrow \emptyset$ {an empty stack containing actions and states (i.e., prefix of traces)}
5: $\texttt{push}(\mathcal{S}_\Pi, \epsilon \cdot \rho_0)$
6: **repeat**
7: $\_ \cdot \rho \leftarrow \texttt{head}(\mathcal{S}_\Pi)$ {retrieve the element in the top of the stack (only q is considered)}
8: **if not** $mark(\rho)$ **then** {if ρ has not visited yet}
9: $mark(\rho) \leftarrow \texttt{true}$ {remember that q ha been mark it}
10: **if** $\langle \rho, \mathcal{S}_\Pi \rangle \not\models f$ **then** {if f is not satisfied over ρ along trace $\mathcal{S}_\Pi$}
11: **return** $\texttt{false}, \mathcal{S}_\Pi$ {return counterexample}
12: **else** {f is satisfied}
13: $\Lambda \leftarrow \{\alpha : \alpha$ is an enabled transition from state $\rho\}$
14: **if** $\Lambda = \emptyset$ **then** {no transition is possible}
15: $\texttt{pop}(\mathcal{S}_\Pi)$ {delete head element $\alpha \cdot \rho$ from trace stack (i.e., backtrack)}
16: **else**
17: $\forall \alpha \in \Lambda, \texttt{push}(\mathcal{S}_\Lambda, \alpha)$ {push the enabled actions into $\mathcal{S}_\Lambda$}
18: **end if**
19: **end if**
20: **end if**
21: **if** $\mathcal{S}_\Lambda \neq \emptyset$ **then** {if some transition remains}
22: $\alpha \leftarrow \texttt{pop}(\mathcal{S}_\Lambda)$ {retrieve next transition (i.e., depth first search)}
23: **let** $\rho' : \rho \overset{\alpha}{\longmapsto} \rho'$ **in** $\texttt{push}(\mathcal{S}_\Pi, \alpha \cdot \rho')$
24: $\rho \leftarrow \rho'$
25: **else**
26: $\texttt{pop}(\mathcal{S}_\Pi)$ {backtrack}
27: **end if**
28: **until** $\mathcal{S}_\Pi = \emptyset$ {all states have been visited}
29: **return** $\texttt{true}$

References

1. M. Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, 1999.
2. M. Abadi and A. D. Gordon. Reasoning about Cryptographic Protocols in the Spi Calculus. In *Proc. of CONCUR '97: Concurrency Theory, 8th International Conference*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.
3. M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols. The Spi Calculus. Technical Report 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.
4. R. M. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proc. of 11th International Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394, 2000.
5. M. Boreale. Symbolic trace analysis of cryptographic protocols in the spi-calculus. In *Proc. of ICALP 2001*, 2001.
6. M. Boreale and D. Gorla. Proof techniques for cryptographic processes calculi and the verification of security properties. *Journal of Telecommunications and Information Technology - Special Issue on Cryptographic Protocol Verification*, 2002/4:28–40, 2002.
7. M. Boreale, R. De Nicola, and R. Pugliese. Process Algebraic Analysis Of Cryptographic Protocols. In *Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, pages 375–392. Kluwer Academic Publishers, 2000.
8. P. J. Broadfoot, G. Lowe, and A. W. Roscoe. Automating data independence. In *ESORICS*, pages 175–190, 2000.
9. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proc. of the Royal Society of London*, volume 426 of *Lecture Notes in Computer Science*, pages 233–271. Springer-Verlag, 1989.
10. I. Cervesato, N. A. Durgin, P. D. Lincoln, John C. Mitchell, and Andre Scedrov. A Meta-Notation for Protocol Analysis. In *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*. IEEE Computer Society Press, 1999.
11. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. *ACM Transaction on Programming Languages and Systems*, 8(2):244–263, 1986.
12. E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.
13. H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002. special issue on Cryptographic protocol verification.
14. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transaction on Information Theory*, 29(2):198–208, 1983.
15. L. Durante, R. Sisto, and A. Valenzano. A State-Exploration Technique for Spi-Calculus Testing-Equivalence Verification. In *Proc. of the IFIP TC6 WG6.1 Joint International Conference FORTE XIII and PSTV XX*, pages 155–170. Kluwer Academic Publishers, 2000.
16. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. of the 14th Computer Security Foundation Workshop (CSFW-14)*, pages 160–173. IEEE, Computer Society Press, 2001.

17. R. Focardi, R. Gorrieri, and F. Martinelli. Non Interference for the Analysis of Cryyptographic Protocols. In *Proc. of the ICALP'00*. Springer-Verlag, 2000.
18. U. Frendrup, Hans Hüttel, and J. N. Jensen. Modal logics for cryptographic processes. In *Proc. of EXPRESS 02*, 2002.
19. A. Giani, F. Martinelli, M. Petrocchi, and A. Vaccarelli. A Case Study with PaMoChSA: A Tool for the Automatic Analysis of Cryptographic Protocols. In *Proc. the World Multiconference on Systemics,Cybernetics and Informatics and International Conference on Information Systems, Analysis and Synthesis (SCI/ISAS)*, pages 108–116, 2001.
20. A. D. Gordon and A. Jeffrey. Authenticity by Typing for Security Protocols. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, pages 145–159. IEEE Computer Society, 2001.
21. J. Heather and S. Schneider. Towards Automatic Verification of Authentication Protocols on an Unbounded Network. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW '00)*, pages 132–143. IEEE, 2000.
22. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
23. G. Lowe. Casper: A compiler for the analysis of security protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
24. C. A. Meadows. The NRL protocol analyzer: an overview. In *Proc. of the 2nd International Conference on the Practical Application of PROLOG*, 1994.
25. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40, 1992.
26. J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murϕ. In *Proceeding of the IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
27. L. C. Paulson. Proving Properties of Security Protocols by Induction. In *Proc. of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
28. A. W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.
29. S. Schneider. Security properties and CSP. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 174–187, 1996.
30. S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transaction on Sofware Engineering*, 24(8):743–758, 1998.
31. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. of the 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.
32. T. Woo and S. Lam. A Semantic Model for Authentication Protocols. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, 1993.

A FORMAL MODEL FOR TRUST LIFECYCLE MANAGEMENT

Waleed Wagealla[1], Marco Carbone[2], Colin English[1], Sotirios Terzis[1], and Paddy Nixon[1]

[1] Department of Computer and Information Sciences,
University of Strathclyde in Glasgow,
26 Richmond Street, Glasgow, G1 1XH United Kingdom
firstname.surname@cis.strath.co.uk
[2] BRICS, University of Aarhus,
Ny Munkegade b.540, 8000 Aarhus C, Denmark
carbonem@brics.dk

Abstract. The rapid development of collaborative environments over the internet has highlighted new concerns over security and trust in such global computing systems. The global computing infrastructure poses an issue of uncertainty about the potential collaborators. Reaching a trusting decision in such environments encompasses both risk and trust assessments. While much work has been done in terms of modelling trust, the investigation of the management of trust lifecycle issues with consideration of both trust and risk is less examined. Our previous work addressed the dynamic aspects of trust lifecycle with a consideration of trust formation, exploitation, and evolution. In this paper we provide an approach for formalizing these aspects. As part of the formalization of the trust lifecycle, we introduce a notion of *attraction* to model the effect of new pieces of evidence on our opinion. The formalization described in this paper constitutes the basis of ongoing work to investigate the properties of the model.

1 Introduction

Increasing interest in collaborative applications for accomplishing difficult distributed tasks in the global computing context (Internet and other large networks) has led to an increased awareness of the security issues in such a dynamic and open environment [5], [4]. It is becoming widely acknowledged that traditional security measures fail to provide the necessary flexibility for interactions between mobile autonomous entities, due to statically defined security policies and capabilities. The global computing setting precludes reliance on some central control authority, which in traditional systems would have full knowledge of all the entities within the system. One approach that has been suggested to help alleviate these problems is the concept of trust [7], which in analogous situations in human society has proved to be effective in allowing unknown entities to learn to determine how each other are likely to behave based on evidence that is available for evaluation. In the SECURE project[9], we are providing an approach that considers risk and trust to secure collaborations between entities. At the heart of the SECURE approach lies the formal trust model [2] and risk model [1].

In a global collaboration environment, mobile entities form ad-hoc collaborations in order to achieve their goals. These entities might represent persons, devices, or their software agents. Every entity that needs to make trusting decisions is defined in our model as a principal in a set *P*. A collaboration involves a number of actions belonging to the set *A*, the set of all possible actions. Although a collaboration in the general case might involve multiple actions and principals, in this paper we restrict ourselves to single action collaborations between two principals. We call such collaborations simple. Thus, a simple collaboration is defined as follows:

Definition 1 (Simple Collaboration). *Given a set P of principals and a set of actions A, a simple collaboration C between p_i and p_e elements of $\mathcal{P}$ is defined as a triple (act, p_i, p_e) , where*

- ▷ *p_e is the executor of the action,*
- ▷ *p_i is the initiator of the action,*
- ▷ *act denotes the action (actions could be parameterized) that p_i wants to perform and is an element of A, the set of actions.*

The starting supposition is that one of the two collaborators (p_i) sends a request to p_e asking for permission to perform the action *act*. Since the environment of collaboration is characterized by a high level of uncertainty about the identity and trustworthiness of the collaborators, a trusting decision needs to be made. In such a situation both trust and risk assessment play a crucial role in establishing collaboration, in the sense that no trusting decision is required if there is no risk involved in executing the action. This is the reason for having trust and risk models underpinning the SECURE project.

The structure of the paper is as follows: section 2 describes the formal model of the trust-based decision maker with its components. Section 3 discusses the trust policy languages. We conclude and state the future work in section 4.

2 Trust-based decision maker

The approach we are taking in the SECURE project is to develop a dynamic model of trust to provide entities with the ability to collaborate and make security related decisions autonomously. A trusting decision will be reached by weighing the trust information of the principal and the risk assessment for the requested interaction. The cyclic relationship between trust and risk implies that on one hand, trust is a parameter in the risk assessment, while on the other hand risk is a parameter in the trust evolution process.

Figure 1 illustrates the structure of the decision maker that incorporates four components: *a trust engine, a risk evaluator, a trust lifecycle manager,* and *an evidence repository*. For a trust-based decision to be made, the trust engine supplies the risk evaluator with the trust information that would help in reasoning about the risk involved in the collaboration. For example, a trustworthy principal behaves in a way that increases the probability of high benefits and reduces the probability of high costs. The decision is based on the risk aversion of the deciding principal and the expected costs/benefits of the interaction as they are dictated by risk analysis. The trust information , in the form

of evidence from interactions, collected and processed by the trust lifecycle manager will be fed into the risk evaluator and the trust engine to be used in future interactions. Trust information or values may be stored in memory to represent historical information on the behavioural patterns of specific entities. The functionalities of these components are further discussed in the following subsections.

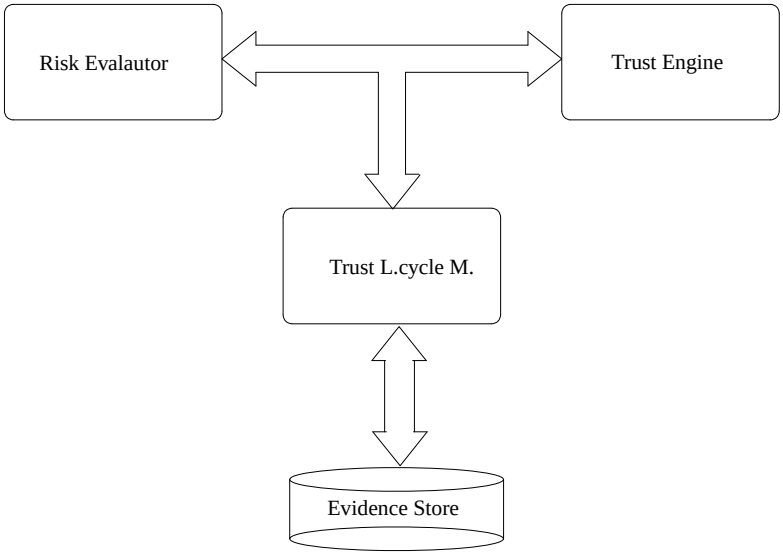

Fig. 1. The trust-based decision maker

2.1 Trust Engine

The aims of the trust engine are to update trust information based on the accumulation of evidence and to supply the risk evaluator with the required trust information for handling requests and reaching decisions.

In the global computing setting, the set of active principals is very large. Therefore, every principal has its own trust values on a finite set of other principals and associates *unknown* to the rest (*unknown* is interpreted as having no evidence for trust or distrust).

In the trust engine [2], the global trust of the system is expressed as a function (m) mapping principals ($\mathcal{P}$) to a function from principals to trust information ($\mathcal{T}$). Formally we have:

$$m : \mathcal{P} \rightarrow \mathcal{P} \rightarrow \mathcal{T}$$

meaning that m applied to a principal a and then to a principal b would return the trust value $m(a)(b) \in \mathcal{T}$ expressing a's trust in b. The set $\mathcal{T}$ is a set of trust values, whose

elements represent degrees of trust. The set $\mathcal{T}$ has two orderings $\preceq$ and $\sqsubseteq$ such that $(\mathrm{T}, \leq)$ is a complete lattice and $(\mathrm{T}, \sqsubseteq)$ is a complete partial order (c.p.o.) with a bottom element. The two orderings ($\preceq$ and $\sqsubseteq$) that defined on the set $\mathcal{T}$ are:

The first ordering ($\preceq$) is the trust ordering, that allows the qualitative comparison of trust values. For example, in a complete lattice $(\mathrm{T}, \leq)$ that expresses a set of trust labels (*low,medium, high*) the value *high* reflects "more trust" in comparison to the value *medium*.

The second ordering ($\sqsubseteq$), the trust information ordering, represents the quantity of available trust information for a principal. In essence this enables the comparison of the precision of trust information.

Following these two orderings, we can construct the triple $(\mathrm{T}, \leq, \sqsubseteq)$ starting with a complete lattice, $(D, \leq)$, and considering the set of intervals of type $[d_0, d_1]$ over D. The $\preceq$ ordering on this set of intervals allows the construction of a complete lattice of intervals which can be allocated as trust values. For example, the interval [*medium, high*] represents higher trust than the interval [*low, medium*].

The $\sqsubseteq$ ordering considers the intervals' width, which can be thought of as representing the amount of uncertainty. For example, the interval [*low,medium*] represents less uncertainty in comparison to the interval [*low, high*]. In fact, the interval expresses that the exact trust label could be either *low* or *high*.

Each principal has a trust policy that expresses how the principal plans to compute its own trust information (for examples of policies see section 3 of this paper). The global trust function *m* will be computed from the collection of all the policies (one for each principal).

There are two operations defined in the trust engine: *update* and *trust*. The *update* method modifies the state of the engine with new observation, and the *trust* method returns a trust value for a principal. Local policies compute the trustworthiness of the principal.

2.2 Risk Evaluator

The function of the risk evaluator is to reason about the risk involved in the interactions between principals. The risk evaluator helps in reaching decision under the circumstances of uncertainty. For this reason the whole process of the risk evaluation is based on investigating the likelihood of potential costs or benefits of the possible outcomes.

Each action is associated to a set of possible *outcomes*. Each outcome has an associated subjective family of cost-probability density functions (*cost-pdfs*)[1]. The cost-pdfs represent the range of possible costs and benefits that may be incurred by the user for each outcome, reflecting variations in principal behaviour. Actions might have parameters associated with them. In this case, the action parameter may affect the values of the costs/benefits.

The trust information that is provided by the trust engine selects one cost-pdf from the family of cost-pdfs for each outcome. The selected cost-pdf for each of the independent outcomes are combined and analyzed according to a security policy, to facilitate a decision on the requested action. The decisions need not be binary, in the sense of "accept or reject", but may also encode some other decisions, *i.e.* asking for more information.

The approach we are taking in the SECURE project is to incorporate trust and risk for trust-based decision making for collaboration between entities. Reasoning about risk and trust indicates the mapping relationship between users' trustworthiness and users' profile of expected behaviour (cost-pdfs).

2.3 Trust Lifecycle Manager

The dynamic aspects of the model such as how trust is formed, how trust evolves over time due to available information and how trust can be exploited are collectively referred to as the trust lifecycle. Therefore, the trust lifecycle manager evaluates trust information and reflects that on the level of principals' trustworthiness. The processes of trust formation and trust evolution are slightly different. Formation differs from evolution in that additional evidence might be actively sought for formation, while evolution refers to the process of changing one's estimation of trust based on gathered trust information from interactions.

There are two main sources of trust information (evidence): *observations* and *recommendations*. Personal observations are gathered after each collaboration is complete. Personal observations of the entity's behaviour are essential for the subjective evaluation of trustworthiness; therefore the outcome of collaborations is recorded and stored as evidence for future collaborations. Recommendations from trusted third parties provide the possibility for trust regarding unknown entities to be propagated.

It is important to state that we would expect personal observations to influence trust to a greater degree than recommendation, therefore it is important to weight the evidence dependent on the source of the information. Since trust is a subjective notion, our own trust information has more merit in the sense that we have first hand knowledge of the whole circumstances of the interaction. Recommendations, however, are passed on as a trust value, which conceals much of the supporting information. The process of recommendation becomes more important in cases where we have no personal interactions with the entity in question.

In [6], we introduced the notion of attraction. The attraction represents the effect of a new piece of evidence on the current trust value. The new evidence (either from observations or recommendations) has some influence on our opinion, in the sense that it "attracts" our opinion towards it.

Definition 2 (Attraction). *An attraction att is a pair* (τ, ι) *where* τ *is the direction of the attraction and* ι *is the power (strength) of the attraction.*

The attraction operates by using the two orderings: trust ordering ($\preceq$) and information ordering ($\sqsubseteq$). Accordingly, we have that:

- ▷ The direction of the attraction, τ, is determined by the trust ordering. Relative positioning of the two trust values according to the trust ordering, i.e. trust positive, trust neutral and trust negative. For example, if T_{curr} = [*low, medium*] and T_{evd} = [*medium, high*] then the trust value from evidence is trust positive.
- ▷ The power (strength) of the attraction, ι, is determined by the information ordering. Relative strength of the two trust values is determined by their positioning on the information ordering c.p.o. For example, if T_{curr}=[*low,medium*] and T_{evd}=[*medium,high*] then the trust value from evidence is conflicting.

The power (strength) has the following properties:

- ▷ The stronger the new piece of evidence, the stronger the attraction (τ and ι) is.
- ▷ The stronger our opinion the weaker the attraction.
- ▷ A supporting evidence reinforces our opinion.
- ▷ A contradicting evidence undermines our opinion.

Before describing the application of the notion of attraction and its two functions to observations and recommendations, we will discuss the evaluation of observations and recommendations.

Observation. Observation is defined as the observed cost-benefit for each outcome upon completion of an interaction (*ocb-outcome*). The evaluation function takes place after completing an interaction and obtaining some observation. Formally the evaluation of observation is achieved by the function `eval()`. This function takes as arguments the current trust value (T_{curr}) and the observed costs-benefits of the outcomes and returns a trust value.

As described above in the risk evaluator, the outcomes are associated with their potential cost and likelihood. Incorporating the trust model and the risk model allows us to choose the best cost-pdf for the outcomes. By reaching a decision, a prediction of the expected cost/benefit of the outcomes will be made. The evaluation function determines the level of trust in the principal based on the one piece of evidence, by comparing the current trust value used to initiate the collaboration with the observed cost/benefit (*ocb-outcome*). The evaluation function must therefore be able to determine the correct trust value which corresponds to the ocb-outcome, to enable the determination of the relevant trust value. This is achieved through the use of a reversible mapping from cost/benefit to trust values.

The evaluation function $\texttt{eval}_p^{act} : \mathcal{T} \times \mathit{Outcomes} \rightarrow \mathcal{T}$, where Outcomes is the set of possible outcomes and T is the set of trust values. The function takes a pair (ocb-outcome, T_{curr}), where ocb-outcome represents the observed cost/benefit and the T_{curr} represent the current trust value that was in use before establishing the interaction. The evaluation function provides an insight into which cost-pdfs would have been the more accurate predictors of the outcome of the interaction (i.e. which set of cost-pdfs would have provided the highest likelihood for the observed costs-benefits). This evaluation function produces a trust value (i.e. the trust value that would have picked this set of cost-pdfs). The evaluation function *eval()* could be defined as,

$$\texttt{eval}_p^{act} = \mathit{eval}(T_{curr}, \mathit{ocb\text{-}outcome}, C) = \mathit{eval}(T_{curr}, \mathit{ocb\text{-}outcome}, act, p_e, p_i)$$

where *act* is the action and p_e and p_i are the principals. Following this approach evidence might be stored with two indexes identifying the action and the principal who performed the action. This allows us to look up the history of evidence by either the principal or the associated action.

Recommendation. Recommendations are defined as,

$$\mathit{rec} : \mathcal{P} \rightarrow \mathcal{P} \rightarrow \mathcal{T}$$

For example if $a, b \in \mathcal{P}$ ($\mathcal{P}$ is the set of principals) we can say that $rec(a)(b)$ is a's trust value for b. We seek recommendation for further information when the amount of observations is insufficient. Accordingly, we request recommendations, with respect to the principal in question, from trusted third parties. Trusted third parties pass on their opinion on the principal in question as a trust value. This trust value summarizes their own observations. Having recommendations based on third parties' observations prevent us from receiving second hand recommendations.

Assumption 1 (Uniform Behaviour).

We only consider recommendations from principals that observe uniform behaviour to us, In the sense that we observe similar profile of cost/benefit probability mass.

The driving force behind this assumption is that we only consider recommendations provide information useful to us. Unless we observe similar behaviour then a recommendation is misleading.

There is an issue of discounting recommendations. The current state of our formalization does not incorporate a notion of discounting recommendations. We are considering this issue in our agenda of future work. The discounting will change the trust value before its evaluation otherwise the approach is the same.

The way we deal with recommendations is similar to observations by calculating their respective trust values' attraction. Each piece of evidence (either from recommendations observations) has its own attraction on our trust value.The core step of the evolution of trust values is the application of the following two attraction-functions to the new piece of evidence.

As said above, the attraction-functions correspond to the notion of the two orderings of the trust engine (*trust ordering* and *information ordering*). Therefore, we have two attraction-functions: trust function (τ-function) and information function (ι-function). These two functions take as arguments the current trust value T_{curr} and the trust value from the piece of evidence (for simplicity called T_{evd}), be that from an observation (after the application of the *eval()* function) or from a recommendation (note that the recommender is just providing a trust value).

τ-function. This function takes the current trust value (T_{curr}) and the trust value from evidence (T_{evd}) and produces a trust-attraction value (τ-*value*). Formally we can describe the function as:

$$\tau(T_{curr}, T_{evd}) = \tau\text{-value}$$

These trust-attraction values belong to a domain that is divided into trust increasing, trust decreasing and trust neutral values. A trust increasing value cannot lower, according to the trust ordering, our current trust value. For example, if T_{curr} is *[medium, medium]* and T_{evd} is *[medium,high]* that means the new trust values cannot be lower than *[medium, medium]*. Similarly a trust decreasing value cannot increase our current trust value. While a trust neutral value can neither increase nor decrease our current trust value. If the trust-attraction value is positive, negative or neutral depends on the relative position on the trust ordering lattice of the two trust values, arguments of the τ-function.

ι-function. This function also takes the current trust value T_{curr} and the trust value from evidence (T_{evd}) and produces an information-attraction value (ι-*value*). Formally we can describe the function as:

$$\iota(T_{curr}, T_{evd}) = \iota\text{-value}$$

These information-attraction values belong to a domain that is divided into conflicting, reinforcing and neutral values. To determine which value of the domain the ι-*value* belongs to, we follow these conditions:

- The maximum reinforcing value is returned when the greatest lower bound (g.l.b.) according to the information ordering of the two arguments is equal to the current trust value(T_{curr}).
- The maximum conflicting value is returned when the g.l.b. of the two arguments is equal to the least element of the information ordering.
- A neutral value is returned when the g.l.b. of the two arguments is equal to the trust value from the evidence.

The exact measure of the ι-attraction value has the following relationship to the strength (number of labels included in interval) of the arguments and the overlap(number of common labels):

(Strength (evidence trust)/Strength (current trust))*(Overlap)

The maximum strength for a trust value is assigned to values that have no other trust value greater than them according to the information order, while the minimum strength is assigned to the least element of the information ordering c.p.o. The rest of the strength values reflect the number of trust values that are greater than the value in question according to the information ordering. The overlap reflects the level of contradiction or reinforcement of the two trust values and as was described above depends on the relative position of our current trust value to the g.l.b. of two arguments of the ι-function.

Applying the two attraction functions (ι-function and τ-function) to the new piece of evidence produces a pair (τ,ι)that will be used to update the current trust value (T_{curr}).

The update function. The update function, *update()*, takes as input the pair (τ, ι) and the current trust value (T_{curr}), producing a new trust value (T_{new}).Formally, we can describe the *update()* function as:

$$\text{update}((\tau, \iota), T_{curr}) = T_{new}$$

The two attraction values are in fact excluding certain trust values as a possibility for the new trust value. The τ-attraction excludes values based on their position on the trust ordering lattice, while the ι-attraction excludes values based on their position on the information ordering c.p.o. For example, a trust increasing value τ-value would exclude all trust values for which the current trust value is greater than according to the trust ordering. Similarly, a contradicting value ι-value will exclude all trust values that are greater than the current trust value according to the information ordering. These

functions will result in two subsets (τ-set and ι-set $\subseteq T$) of trust values each consistent to the evaluation of the new piece of evidence according to the each ordering. The new trust value $T_{new} \in \tau$-set $\bigcap$ ι-set. The exact functions for the determination of the new trust value depends on the application.

2.4 Evidence Store

In our model the position is taken that the trust information is stored in order to allow us to reason about principals' trustworthiness in future interactions. Therefore, we introduce a structure called the *evidence store*. The use of a layered structure (Figure 2) to store trust information provides a greater depth of information upon which to base any decision than merely storing the individual trust values relating to the entity in question.

The following points discuss these layers of the evidence store and the operation functions between the layers:

1. The base layer contains lists of all observations or recommendations. The output of the evaluation function `eval()`will be stored in these lists. There is a fixed size for each one of these lists. When adding a new experience, the size of the list (L) will be checked. If L has some space, then the new piece of evidence will be added to the previous ones. When L is full, the attraction of old observations or recommendations will be calculated and then cleared so as to add the new one. In this way, all evidence in the history of the principal's interactions is reflected in the trust value even though the observations themselves are discarded. The trust update function *update()* produces a trust value (T_{obs})that will be stored on the second layer. The update of the evidence will take place separably, for recommendations and observations, to allow us to determine a trust value that purely based on observations (T_{obs}). This facilitates the process of giving recommendations upon receiving recommendation requests.
2. The second layer in the structure contains a trust value (T_{obs}) specific to observation and a trust value (T_{rec}) reflects the attraction of all the received recommendations. The new final T_{curr} to be used for decision making can be calculated using the attraction notion in two ways. The first approach is to apply the attraction of T_{rec} on T_{obs} to produce a new trust value that will be stored as (T_{curr})in the top layer. In the second approach we apply the attraction of both T_{rec} and T_{obs} on the current trust value (T_{curr})that is stored in the top layer and produce a new trust value. The result of these two operations produce a trust value that will be stored in the top layer.
3. The value in the top layer influences the trust engine, which invokes the trust method. In this way, the local trust policy will be updated in the light of this new trust value.

3 Trust Policy languages

In this section we show how to express what was said in the previous sections using a language for describing policies.

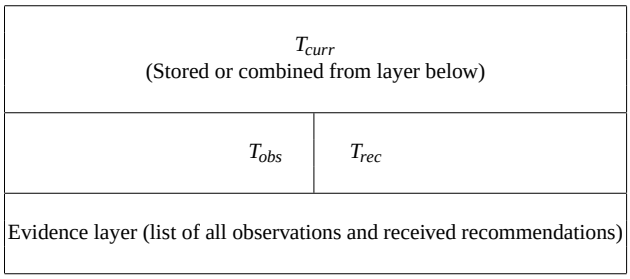

Fig. 2. The Trust Information Structure

3.1 The Policy Language

In [2] we defined a general setting, where policies are specified via a language. Formally, policies are functions of type $\mathcal{P} \to \mathcal{T}$ such that for each principal a they return a trust value. The language permits the specification of these functions. A policy language has the form $\lambda x : \mathcal{P}.\tau$ where τ specifies the returned value of the policy, done by using certain operators. One of these operators is the operator $\ulcorner \cdot \urcorner$, called *delegation*. For instance the policy $\lambda x : \mathcal{P}.\ulcorner a \urcorner(x)$ specifies the function that, for any principal $b \in \mathcal{P}$, returns a's trust in b. Delegation makes policies dependent on the global trust m. The function m is defined as the least fixed point of all policies (see the reference for more details and formal semantics).

The operator of delegation can also be combined with other operators provided from the language. For instance we could write a policy which says that our trust in b is a's trust in b but for any other principal c it will be the least upper bound between a's trust in c and a threshold value $[d_0, d_1]$:

$$\lambda x : \mathcal{P}.(x = b) \mapsto \ulcorner a \urcorner(x); ([d_0, d_1] \sqcap \ulcorner a \urcorner(x))$$

where the operator $\cdot \mapsto \cdot; \cdot$ is semantically equivalent to an if-then-else and $\sqcap$ is the l.u.b. in lattices.

3.2 Encoding in the policy language

The encoding in the policy language is done by exploiting two features: the syntax of the policy and the structure of the set $\mathcal{T}$. When we give a policy π written in the language we give an algorithm for computing a function. Hence, when writing the policy, we "store" some information in the syntax of the language which means that it is possible to record the current trust value.

Representing the various layers of the evidence repository could be handled by giving more structure to the set $\mathcal{T}$. Suppose that the initial set of trust values (as the one of the previous sections) is $\mathcal{T}_{val}$, provided with an information ordering $\sqsubseteq$ and a trust ordering $\preceq$ as required. Then we can define a new set $\mathcal{T}$ as

$$\mathcal{T} = [EVD] \times \mathcal{T}_{obs} \times \mathcal{T}_{rec}$$

where $[EVD]$ is the list of observations, *e.g.* $[evd_1,\ldots,evd_n] \in [EVD]$ for $evd_1,\ldots,evd_n$ evidences. This means that our policy contains a list of evidences (as shown in the base layer of the evidence repository), and two trust values (as shown in the second layer): the first one which refers to the current trust value based on observation (which is then a constant in the syntax), and the second one which refers to the value updated with recommendations.To describe these trust values as a policy, we will initially start from the situation where there is no observations or recommendations. Therefore, the policy could be defined as:

$$\pi_a = \lambda x : \mathcal{P}.([], \bot, \bot)$$

where [] is the empty list and $\bot$ is the bottom of the information ordering (no information).

When we have an observation, say o_1, we just need to update the list of observations and so get the the new policy

$$\pi_a = \lambda x : \mathcal{P}.([evd_1], \bot, \bot)$$

In a situation when the returned trust value that we asked the trust engine for is not enough, we might ask principal b for recommendation about c. Then our policy would be updated to

$$\pi_a = \lambda x : \mathcal{P}.([], [d_0, d_1], (x = c) \mapsto update((\tau_1, \iota_1), [d_0, d_1])$$

where $\tau_1 = \tau((\ulcorner b \urcorner(x).2), \ulcorner a \urcorner(c).2))$ and $\iota_1 = \iota((\ulcorner b \urcorner(x).2), \ulcorner a \urcorner(c).2))$ determines the attraction. The operator .2 (in general .n for any natural number) stands for the projection of the 2-nd element (in general n-th) over the tuple (over set $\mathcal{T}$). Hence every time we update we keep the old value and add new references to other principals.

Notice that when getting an observation we just store it in the list but we do not update the trust value. That is going to be done with a further update which basically cleans the list and updates (with the function `eval`) the trust value in the middle of the triple.

4 Conclusions and Further Work

The primary contribution of this paper is in formalizing a model for a trust-based decision maker that incorporates four components: trust engine, risk evaluator, trust lifecycle, and evidence repository. The model reflects on how important is to take the two factors of trust in risk on board in establishing collaboration between principals. The model showed that reaching decision is a prolonged process to remove some shade of uncertainty issues concerning risk and principals' trustworthiness.

A similar approach for trust management formalization is described in [3]. This model also accommodates and correlates trust and risk in establishing interactions in the context of e-services. In their proposed model, there is no formal definition of trust. Alternatively, they described a classification of trust relationships in the environment of e-commerce. It is also important to mention that subjective logic theory [8] , has influenced our addressing of issues of uncertainty in our model.

There are a number of promising directions for further investigation. One of these issues is how to introduce complexity in the model by considering more complex collaborations (i.e. a collaboration that takes place between more than two collaborators to perform dependent actions). An interesting area of research is how to make a decision based on the relative context and time. This implies encoding these two notions in the model. For the trust engine, one particular area of interest is to complement the denotation model presented here with an operational model where, for instance, we will need to address the question of computing trust information efficiently over the global network.

Our future research will be focused on examining this model by developing a simulation framework. In this framework, we will study the properties of the proposed model. The results of the simulation and the study of the current model properties will guide us to more complex collaboration model. We hope to report on this work shortly.

Acknowledgements The work in this paper is supported by SECURE (Secure Environments for Collaboration among Ubiquitous Roaming Entities, IST-2001-32486) funded by the EU FET Programme under the Global Computing Initiative.

References

1. Jean Bacon, Nathan Dimmock, David Ingram, Ken Moody, Brian Shand, and Andy Twigg. Definition of risk model. *SECURE Deliverable 3.1*, 2002.
2. Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A formal model for trust in dynamic networks. In *Proc. of International Conference on Software Engineering and Formal Methods*, September 2003.
3. Theo Dimitrakos. System models, e-risks and e-trust.towards bridging the gap? In *Proceedings of the 1st Conference on e-Commerce, e-Business, e-Government.Kluwer Academic Publishers*, October 2001.
4. C. English, P. Nixon, S. Terzis, A. McGettrick, and H. Lowe. Security models for trusting network appliances. In *Proc. of the 5th IEEE International Workshop on Networked Appliances*, October 2002.
5. C. English, W. Wagealla, P. Nixon, S. Terzis, A. McGettrick, and H.Lowe. Trusting collaboration in global computing. In *Proc. of the First International Conference on trust management*, May 2003.
6. Colin English, Sotirios Terzis, Waleed Wagealla, Paddy Nixon, Helen Love, and Andrew McGettrick. Trust dynamics for collaborative global computing. In *Proc. of the WETICE conference*, 2003.
7. Tyrone Grandison and Morris Sloman. A survey of trust in internet application. *IEEE Communications Surveys, Fourth Quarter*, 2000.
8. Adun Jøsang. A logic for uncertain probabilities. *Fuzziness and Knowledge-Based Systems*, 9(3), 2001.
9. SECURE Project. Official website. `http://secure.dsg.cs.tcd.ie`, 2002.