Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Einar Snekkenes Dieter Gollmann (Eds.)

Computer Security – ESORICS 2003

8th European Symposium on Research in Computer Security
Gjøvik, Norway, October 13-15, 2003
Proceedings

Springer

Volume Editors

Einar Snekkenes
Gjøvik University College
P.O. Box 191, 2802 Gjøvik, Norway
E-mail: einar.snekkenes@hiq.no

Dieter Gollmann
Microsoft Research, Roger Needham Building
7 JJ Thomson Avenue, Cambridge CB3 0FB, United Kingdom
E-mail: diego@microsoft.com

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

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

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

ISSN 0302-9743
ISBN 3-540-20300-1 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10949931 06/3142 5 4 3 2 1 0

Preface

ESORICS, the European Symposium On Research In Computer Security, is the leading research-oriented conference on the theory and practice of computer security in Europe. The aim of ESORICS is to further the progress of research in computer security by establishing a European forum for bringing together researchers in this area, by promoting the exchange of ideas with system developers and by encouraging links with researchers in related areas.

ESORICS is coordinated by an independent steering committee. In the past it took place every two years at various locations throughout Europe. Starting this year, it will take place annually.

ESORICS 2003 was organized by Gjøvik University College, and took place in Gjøvik, Norway, October 13–15, 2003.

The program committee received 114 submissions, originating from 26 countries on all continents. Half the papers originated in Europe (57). The most dominant countries were: UK (16), USA (14), Germany (6), South Korea (6), Sweden (6), Italy (5), France (4) and Poland (4). Each submission was reviewed by at least three program committee members or other experts. The program committee chair and co-chair were not allowed to submit papers. The final selection of papers was made at a program committee meeting followed by a week of e-mail discussions. Out of the 114 papers received, only 19 got accepted (17%). In comparison, ESORICS 2000 and 2002 received 75 and 83 papers and accepted 19% and 16%, respectively.

The program reflected the full range of security research, including access control, cryptographic protocols, privacy enhancing technologies, security models, authentication, and intrusion detection.

The program committee gratefully acknowledges all authors who submitted papers for their efforts in maintaining the standards of this conference.

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

Gjøvik, October 2003 Einar Snekkenes

General Chair

Jørn Wroldsen (Gjøvik University College)

Program Committee

Tuomas Aura	Microsoft Research, UK
Joachim Biskup	University of Dortmund, Germany
Tønnes Brekne	Telenor, Norway
Frédéric Cuppens	ENST-Bretagne, France
Marc Dacier	Eurecom, France
Sabrina De Captani di Vimercati	University of Milan, Italy
Hervé Debar	France Telecom R&D, France
Yves Deswarte	LAAS-CNRS, France
Simon Foley	University College Cork, Ireland
Dieter Gollmann	Microsoft Research, UK
Trent Jaeger	IBM Research, USA
Kaoru Kurosawa	Ibaraki University, Japan
Heiko Mantel	DFKI, Germany
John McHugh	CERT, USA
Ken Moody	University of Cambridge, UK
Stefano Paraboschi	University of Bergamo, Italy
Birgit Pfitzmann	IBM Research, Switzerland
Avi Rubin	Johns Hopkins University, USA
Peter Ryan	University of Newcastle, UK
Pierangela Samarati (co-chair)	University of Milan, Italy
Tomas Sander	Hewlett-Packard, USA
Einar Snekkenes (chair)	Gjøvik University College, Norway
Michael Steiner	IBM Research, USA
Paul Syverson	NRL, USA
Michael Waidner	IBM Research, Switzerland

Additional Reviewers

Anas Abou El Kalam, Carlos Aguilar, Michael Backes, Jeremy Bryans, Laurent Bussard, Christian Cachin, Pau-chen Cheng, Jason Crampton, Roger Dingledine, Ulrich Flegel, Cédric Fournet, Tetsu Iwata, Klaus Julisch, Jan Jürjens, Yücel Karabulut, Wataru Kishimoto, Thomas Leineweber, Shinichiro Matsuo, Ralf Menzel, Refik Molva, Ira Moskowitz, Vincent Nicomette, Yasuhiro Ohtaki, Jörg Parthe, Benny Pinkas, Gerhard Popp, Fabien Pouget, Michael Roe, Ayda Saïdane, Pasha Shabalin, Ron Steinfeld, Morton Swimmer, Vu Dong Tô, Sandra Wortmann, Yuliang Zheng, Alf Zugenmaier

Organizing Committee

Steering Committee

Table of Contents

Signature and Access Control Policies for XML Documents*

Elisa Bertino[1], Elena Ferrari[2], and Loredana Parasiliti Provenza[1]

[1] Università degli Studi di Milano,
Dipartimento di Scienze dell'Informazione,
{bertino,parasiliti}@dico.unimi.it
[2] Università degli Studi dell'Insubria,
Dipartimento di Scienze Chimiche Fisiche e Matematiche,
elena.ferrari@uninsubria.it

Abstract. Information push is today an approach widely used for information dissemination in distributed systems. Under information push, a Web data source periodically (or whenever some relevant event arises) broadcasts data to clients, without the need of an explicit request. In order to make information push usable in a large variety of application domains, it is however important that the authenticity and privacy requirements of both the receiver subjects and information owners be satisfied. Although the problem of confidentiality has been widely investigated, no comparable amount of work has been done for authenticity. In this paper, we propose a model to specify signature policies, specifically conceived for XML data. The model allows the specification of credential-based signature policies, supporting both single and joint signatures. Additionally, we provide an architecture for supporting the generation of selectively encrypted and authenticated XML document, ensuring at the same time the satisfaction of both access control and signature policies.

1 Introduction

Today the Web is becoming the main means to exchange information. This is not only true for private users, but also for companies and organizations that are today using the Web as the main dissemination means. Information dissemination often takes the form of documents that are made available at Web servers, or that are actively broadcast by Web servers to interested clients. This last dissemination mode, which is usually referred to as *information push*, is today becoming more and more attractive since it fits very well with the Web characteristics. According to the push approach a document source periodically (or whenever some relevant event arises) broadcasts data to clients, without the need of an explicit request. Clearly, information push must satisfy the authenticity and confidentiality requirements of both the receiving subjects and information owners. In this scenario, XML [7] plays a crucial role, since it is today becoming

* The work reported in this paper has been partially supported by the EU under the PANDA IST Thematic Network.

the standard for data representation and exchange over the Web. Although the problem of XML document confidentiality has been widely investigated and several access control models have so far been defined [5], no comparable amount of work has been carried out for providing a comprehensive solution to XML document authenticity. In particular, whereas several signature techniques are today available allowing the data receiver to verify the authenticity of the data, no comprehensive solutions exist encompassing all aspects of the signature process and fitting with different information dissemination strategies. We believe that a first requirement is the definition of a model supporting the specification of *signature policies.* A signature policy has to specify which subject(s) must sign a document (or document portion(s)). The main difference between an access control policy and a signature policy is that the first expresses the possibility of exercising a privilege on a given document, whereas the second expresses the *duty* of signing a document or a document portion. Thus, a first contribution of this paper is the definition of a model for specifying credential-based signature policies. To this purpose we extend $\mathcal{X}$-Sec, an XML-based language previously proposed by us in [4] for the specification of access control policies.
The signature policies we have defined are of two different types: *single* and *joint.* Single signature policies require the signature of a single subject on a document, or document portions. A joint signature is by contrast required when the same document portion must be signed by several subjects. This could be the case for instance of a travel approval that must be signed by both the administrative manager and the technical manager. In the paper, we focus on those scenarios in which independent joint signatures of the same document portion are required. Additionally, we provide an architecture, and related algorithms, supporting the generation of encrypted and authenticated XML documents, which ensures at the same time the satisfaction of both access control and signature policies. The architecture builds on an approach previously proposed by us [3] for access control within information push, which is based on encrypting different portions of the same document according to different encryption keys, on the basis of the specified access control policies, and selectively distributing these keys to the various subjects in such a way that each subject receives all and only the decryption keys corresponding to the authorizations he/she has on the considered document. In this paper, we extend this framework to authenticity. The idea is to sign different portions of the document with different signatures, according to the specified signature policies. This is a non trivial extension, since the portions to which the same signature must be applied can be different from the portions that must be encrypted with the same key for confidentiality purposes, and this must be carefully considered to obtain a document encryption that is correct w.r.t. both access control and signature policies. In the paper, we present two alternative strategies for encrypting and signing a document and we discuss the advantages and the drawbacks of both the approaches.

This paper is organized as follows. The next section briefly reviews the main characteristics of $\mathcal{X}$-Sec. Section 3 introduces signature policies. Section 4 presents the system architecture we have developed in order to support the generation of encrypted and authenticated XML documents, according to the

```
<Employee_dossier Emp_ID="EID">
      <Resume Date="8/8/2000">
            <Personal_Data Name="John" . . . >
                  <Reserved>
                        <Health> . . . </Health>
                        <Criminal> . . . </Criminal>
                  </Reserved>
            </Personal_Data>
            <Education>
                  <Qualification> . . . </Qualification>
                  . . .
            </Education>
            <Activity >
                  <Professional_Experience> . . . </Professional_Experience>
                  . . .
            </Activity>
      </Resume>
      <Evaluation>
            <Manag_Eval> . . . </Manag_Eval>
            <Board_Dir_Eval> . . . </Board_Dir_Eval>
            <HR_Eval> . . . </HR_Eval>
      </Evaluation>
      <Career>
            <Position Role="Secretary" Salary="1000" Date="8/8/2000"/ >
            . . .
      </Career>
</Employee_dossier>
```

Fig. 1. An example of XML document

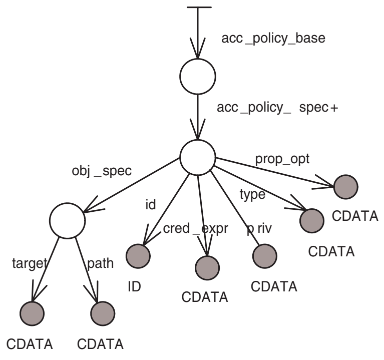
specified access control and signature policies. Section 5 concludes the paper. Finally, Appendix A contains some of the algorithms developed for our system, whereas Appendix B reports the proof of the formal results stated in the paper.

2 Credentials and Access Control Policies

$\mathcal{X}$-Sec is an XML-based language for specifying credentials and credential-based access control policies [4]. In this paper we extend $\mathcal{X}$-Sec with the possibility of specifying signature policies. Before going into the details of this extension, we provide a brief overview of $\mathcal{X}$-Sec [4]. In the following, we assume that an XML source $\mathcal{S}$ is given. In explaining the policies and throughout the paper we use as running example the XML document illustrated in Figure 1. It models an employee dossier, that provides information on employee's activities within an organization. The document contains the employee's resume, the evaluation of the employee activities, and information about his/her career within the organization. The `Resume` element contains both general information on the employee and reserved data, about his/her health and criminal record. The `Evaluation` element contains three evaluations on the employee performance, modeled with three distinct subelements. The first contains the evaluation made by the manager of the employee, the second contains the evaluation of the members of board of directors, and the third the evaluation made by the head of the human resources department. Finally, the `Career` element provides information on the employee's position(s) in the organization hierarchy.

2.1 $\mathcal{X}$-Sec Credentials

To make the task of credential specification easier, $\mathcal{X}$-Sec groups credentials with similar structure into credential types. In $\mathcal{X}$-Sec, a credential type is a



Attribute	Parent_node	Value
`id`	acc_policy_spec	it identifies an access control policy
`cred_expr`	acc_policy_spec	Xpath compliant expression on $\mathcal{X}$-profiles
`priv`	acc_policy_spec	it specifies the access control policy access mode
`type`	acc_policy_spec	it specifies whether the access control policy is positive or negative
`prop_opt`	acc_policy_spec	it specifies the propagation option of the policy
`target`	`obj_spec`	it denotes the DTD/XML-Schema/document to which the policy applies
`path`	`obj_spec`	it denotes selected portions in the `target` document(s)

Fig. 2. (a) Graph representation of the $\mathcal{X}$-Sec access control policy base template and (b) Attribute description

DTD, where simple properties of a credential are modeled as empty elements and composite properties as elements with element content, whose subelements model composite property components. A credential is an instance of a credential type and specifies the set of property values characterizing a given subject against the credential type itself.

To simplify the process of evaluating subject credentials against access control and signature policies, all the credentials a subject possesses are collected into an XML document, called $\mathcal{X}$-*Profile* [4].

2.2 $\mathcal{X}$-Sec Access Control Policies

$\mathcal{X}$-Sec provides an XML based template for specifying credential-based access control policies. Figure 2(a) gives the graph representation of the template. According to such DTD, policy specifications are modeled as empty elements, `acc_policy_spec`, with one subelement, `obj_spec`, and five attributes, whose semantics is described in Figure 2(b). The template allows the specification of both positive and negative access control policies. The subjects to which a policy applies are denoted by imposing conditions on his/her credentials and credential properties. These conditions, called *credential expressions*, are expressed through XPath compliant expressions [7] against $\mathcal{X}$-profiles. By contrast, the `obj_spec` element allows the specification of the protection objects to which a policy applies. It contains two attributes: `target`, denoting the name of an XML document/DTD/XML-Schema, and `path`, which is an XPath compliant expression selecting specific portions within the target on the basis of both its structure and its contents. Access control policies can be of two different types: *browsing policies*, that allow subjects to see the information in a document and/or to navigate through its links, and *authoring policies* that allow the modification of XML documents under different modes. A detailed description of these privilege can be found in [3]. Additionally, the template allows the specification of different *propagation options* in order to reduce the number of access control policies that need to be defined. Propagation options state how policies specified on a given element of a DTD/XML-Schema/document hierarchy propagate (partially or totally) to lower levels. Two different types of propagation are provided: *implicit* and *explicit propagation*. According to default implicit propagation, policies specified on a DTD/XML Schema propagate to all the DTD/XML Schema instances, and policies specified on a given element propagate to all attributes and links of the element. Additionally, the Security Administrator (SA) can state an explicit propagation option in the policy specification by $n \in N \cup \{*\}$. This symbol indicates the "depth" of the propagation: i) if n is equal to '$*$', then the access control policy propagates to all the direct and indirect subelements of the element(s) specified in the object specification; ii) if n is equal to 0, then no propagation of the policy is enacted; iii) if $n \geq 1$, then the policy propagates to the subelements of the element(s) specified in the policy specification, that are at most n levels down in the document/DTD/XML-Schema hierarchy.

An access control policy base, denoted as $\mathcal{ACPB}$, is therefore an XML document instance of the $\mathcal{X}$-Sec access control policy base template. Each policy in the $\mathcal{ACPB}$ is uniquely identified by an *id* attribute assigned when the policy is specified. Given an $\mathcal{ACPB}$, we associate with it the set of access control policy identifiers, defined as follows:

$$\Pi(\mathcal{ACPB}) = \{acp_i \mid i = id, \text{ where } id \text{ is the ID attribute of an } \texttt{acp} \text{ in } \mathcal{ACPB}\}$$

This notation is necessary to distinguish access control policies identifiers from signature policies ones.

Example 1. Figure 3 presents an example of $\mathcal{ACPB}$, defined for a source $\mathcal{S}$ containing the XML document in Figure 1. The first access control policy authorizes the managers to view all the documents in $\mathcal{S}$, having the DTD specified by the *target* attribute. The second policy authorizes an employee to view all the information about his/her resume, his/her position within the organization. The third access control policy authorizes the head of the human resources department to view the corresponding evaluation in the employee dossier and all the information about employee's resume and career within the organization. The fourth policy allows each member of the board of directors to view the corresponding evaluation, and the employee's information contained in the document. Finally, the fifth policy states an exception to the fourth policy preventing the members of the board of directors to view employee's reserved data and salary. In this example, $\Pi(\mathcal{ACPB}) = \{acp_1, acp_2, acp_3, acp_4, acp_5\}$.

3 Signature Policies

In this section, we extend $\mathcal{X}$-Sec to the support of signature policies.

3.1 $\mathcal{X}$-Signature Policy Base Template

Signature policies state which subjects' categories have to sign which document portions within a document source. Signature policies are specified according

```
<acc_policy_base>
   <acc_policy_spec id=1 cred_expr="//Manager" priv=" browse_all" type="grant"
      propt_opt="*"/ >
      <obj_spec target="Employee_dossier.dtd" path="/Employee_dossier//"/ >
   <acc_policy_spec/ >
   <acc_policy_spec id=2 cred_expr="//Employee" priv="browse_all" type="grant"
      propt_opt="*"/ >
      <obj_spec target="Employee_dossier.xml" path="//Resume[Emp_ID=credID]|
         //Position[Emp_ID=credID]"/ >
   <acc_policy_spec/ >
   <acc_policy_spec id=3 cred_expr="//HR_head " priv="view" type="grant"
      propt_opt="*"/ >
      <obj_spec target="Employee_dossier.xml" path="//HR_eval| //Resume|//Career"/ >
   <acc_policy_spec/ >
   <acc_policy_spec id=4 cred_expr="//Board_dir_member" priv="view" type="grant"
      propt_opt="*"/ >
      <obj_spec target="Employee_dossier.xml" path="//Board_dir_eval|//Resume|
      //Career"/ >
   <acc_policy_spec/ >
   <acc_policy_spec id=5 cred_expr="//Board_dir_member" priv="view" type="deny"
      propt_opt="*"/ >
      <obj_spec target="Employee_dossier.xml" path="//Reserved|//@salary"/ >
   <acc_policy_spec/ >
</acc_policy_base>
```

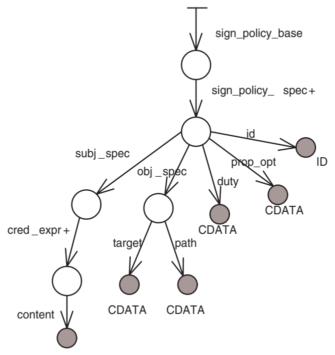
Fig. 3. An example of access control policy base

to the XML policy base template defined in what follows. In the definition of the template we denote with *Label* a set of element tags and attribute names, and with *Label** a set of strings obtained through the concatenation of names in *Label* and a symbol in {*,+,?}.

Definition 1. (**$\mathcal{X}$-Signature policy base template**). *An $\mathcal{X}$-Sec signature policy base template $\mathcal{X}$-*`spt` *is a tuple* $(\overline{v}_{\texttt{spt}}, V_{\texttt{spt}}, E_{\texttt{spt}}, \phi_{E_{\texttt{spt}}})$*, where:*

- $\overline{v}_{\texttt{spt}}$ *is the root element denoting the whole policy base;*
- $V_{\texttt{spt}} = V^e_{\texttt{spt}} \bigcup V^a_{\texttt{spt}}$ *is a set of nodes representing element and attribute types. In particular,* $\overline{v}_{\texttt{spt}}$ *has one or more direct children, belonging to* $V^e_{\texttt{spt}}$*, each of which models a signature policy specification. Such node, called* `sign_policy_spec`*, has two subelements,* `subj_spec` *and* `obj_spec`*, and three attributes:* `id`*,* `duty` *and* `prop-opt`*. Figure 4(b) provides the specification of all the attributes in the signature policy base template;*
- $E_{\texttt{spt}} \subseteq V_{\texttt{spt}} \times V_{\texttt{spt}}$ *is a set of edges, where* $e \in E_{\texttt{spt}}$ *represents an element-subelement or an element-attribute relationship;*
- $\phi_{E_{\texttt{spt}}} : E_{\texttt{spt}} \to Label^*$ *is the edge labeling function. In particular the edge entering* $\overline{v}_{\texttt{spt}}$ *is labeled with* `sign_policy_base`*; the edge entering the (unique) direct child of* $\overline{v}_{\texttt{spt}}$ *is labeled* `sign_policy_spec+`*, to denote the fact that this node is repeatable.*

Figure 4(a) gives the graph representation of the signature policy base template. We omit the description of some attributes specified in the signature policy base template that are analogous to those of the access control one. The only difference is attribute `duty` and element `subj_spec`. Attribute `duty` specifies the kinds of signature policy. It can assume two distinct values: *sign* and *joint_sign* (in the following we denote with $\mathcal{D}$ the set $\{sign, joint\_sign\}$). The *sign* duty imposes that a subject, whose $\mathcal{X}$-profile satisfies the credential expression in the signature policy must sign the document portions to which the policy applies. The *joint_sign* duty can be thought of as an extension of the previous one, to include the possibility of applying a joint signature by more than one subject on



Attribute	Parent_node	Value
`id`	sign_policy_spec	it identifies a signature policy
`duty`	sign_policy_spec	it specifies the signature mode required by the policy
`prop_opt`	sign_policy_spec	it specifies the propagation option of the policy
`target`	obj_spec	it denotes a DTD /XML-Schema/document to which the policy applies
`path`	obj_spec	it denotes selected portions within the target

Fig. 4. (a) Graph representation of the $\mathcal{X}$-Sec signature policy base template, and (b) Attribute description

the same protection objects. It imposes that, for each credential expression specified in the signature policy, one subject, whose $\mathcal{X}$-Profile satisfies the specified credential expression, signs the protection objects to which the policy applies. To support both single and joint signatures, the `subj_spec` element contains one or more `cred_expr` elements, whose content is an XPath compliant expression on $\mathcal{X}$-Profiles. If `duty=sign`, the `subj_spec` element contains only one `cred_expr` element. By contrast, if `duty=joint_sign`, the `subj_spec` contains a list of `cred_expr` subelements. In this paper, we take into account only those scenarios in which independent signatures are required; thus, we do not impose any order in which multiple signatures must be applied on the same nodeset.

3.2 Signature Policy Base

A signature policy base is an instance of the signature policy base template previously introduced, as formalized by the following definition.

Definition 2. (**$\mathcal{X}$-Sec signature policy base**). *Given an $\mathcal{X}$-Sec signature policy base template $\mathcal{X}$-*`sp`*t, an $\mathcal{X}$-Sec signature policy base is a valid XML document with respect to $\mathcal{X}$-*`sp`*t.*

Like access control policies, signature ones are uniquely identified by an *id*, assigned at the time of their insertion. Given a signature policy base $\mathcal{SPB}$, we associate with it the set of signature policies identifiers, defined as follows:

$$\Pi(\mathcal{SPB}) = \{sp_i \mid i = id, \text{ where } id \text{ is the ID attribute of a } \texttt{sp} \text{ in } \mathcal{SPB}\}$$

Example 2. Figure 5 presents an example of a $\mathcal{SPB}$, defined for a source $\mathcal{S}$ containing the XML document in Figure 1. The first signature policy imposes that each employee signs his/her resume; according to the second policy, the manager must sign the employee's evaluation. The third signature policy requires that two members of the board of directors sign the employee evaluation made by the board of directors. Similarly, the fourth policy imposes that the head

```
<sign_policy_base>
  <sign_policy_spec id=1 , duty="sign" propt_opt="*">
    <subj_spec>
      <cred_expr>//Employee</cred_expr>
    < /subj_spec>
    <obj_spec target="Employee_dossier.xml" path="//Resume[Emp_ID=credID]"/ >
  </sign_policy_spec>
  <sign_policy_spec id=2 , duty="sign" propt_opt="*">
    <subj_spec>
      <cred_expr> //Manager </cred_expr>
    < /subj_spec>
    <obj_spec target="Employee_dossier.dtd" path="//Evaluation"/ >
  </sign_policy_spec>
  <sign_policy_spec id=3 , duty="joint_sign" propt_opt="*">
    <subj_spec>
      <cred_expr> //Board_dir_member </cred_expr>
      <cred_expr> //Board_dir_member </cred_expr>
    < /subj_spec>
    <obj_spec target="Employee_dossier.xml" path="//Board_dir_eval"/ >
  </sign_policy_spec>
  <sign_policy_spec id=4 , duty="sign" propt_opt="*">
    <subj_spec>
      <cred_expr> HR_head </cred_expr>
    < /subj_spec>
    <obj_spec target="Employee_dossier.xml" path="//hr_eval"/ >
  </sign_policy_spec>
  <sign_policy_spec id=5 , duty="sign" propt_opt="*">
    <subj_spec>
      <cred_expr>//Payroll_Depart_head|//HR_head </cred_expr>
    < /subj_spec>
    <obj_spec target="Employee_dossier.xml" path="//career"/ >
  </sign_policy_spec>
</sign_policy_base>
```

Fig. 5. An example of signature policy base

of the human resources department signs his/her evaluation. Finally, the last policy states that a subject, who is both the head of the payroll department and the head of the human resources department, must sign the portion of the `Employee_dossier` related to employee's career. In this example, $\Pi(\mathcal{SPB}) = \{sp_1, sp_2, sp_3, sp_4, sp_5\}$.

3.3 Signature Policy Semantics

The term *signature policy semantics* denotes the set of signature duties enacted by a given signature policy. A signature duty can be represented as a triple $(\overline{s}, o, d)$, where $\overline{s}$ is a tuple of n subjects, with $n \geq 1$, $d \in \mathcal{D}$ is a duty, and o is a protection object. Note that if $d \equiv sign$, then $n = 1$, whereas if $d \equiv joint\_sign$, then $n \geq 2$. Before formally defining the semantics of a signature policy, we introduce some preliminary notations and definitions. In what follows, given a signature policy `sp`, *subj_spec(*`sp`*)*, *obj_spec(*`sp`*)*, *duty(*`sp`*)*, *prop_opt(*`sp`*)* denote, respectively, the subject specification, the object specification, the duty, and the propagation option in `sp`. Moreover, *cred_expr(*`sp`*)*, *target(*`sp`), *path(*`sp`) denote, respectively, the credential expression in the *cred_expr* subelement of *subj_spec(*`sp`), the target document/DTD/XML-Schema and the XPath compliant expression on target. Additionally, we define a function *Eval*, that receives as input a name of an XML document/DTD/XML-Schema and an XPath compliant expression, and returns *true* if the input XML document/DTD/XML-Schema satisfies the XPath compliant expression, *false* otherwise. Formally, let *target* be the name of an XML document, or a DTD, or an XML-Schema, and *path* an XPath compliant expression. Then:

$$Eval(target, path) = \begin{cases} true & \text{if } target \text{ satisfies } path \\ false & \text{otherwise} \end{cases}$$

Now, we define the sets of subjects and protection objects by a policy.

Definition 3. (**Credential expression semantics**). *Let* `sp` *be a signature policy,* `subj_spec(sp)` *be the subject specification in* `sp`*, and* `cred_expr(sp)` *be a credential expression in* `subj_spec(sp)`*. The semantics of* `cred_expr(sp)` *is defined as follows:*

$$\Phi(\texttt{cred\_expr(sp)}) \overset{def}{=} \{s \in S | \text{ Eval } (\mathcal{X}\textit{-profile,}\texttt{cred\_expr(sp)}) = true, \\ /\mathcal{X}\textit{-profile@sbjID=s}\}.$$

Definition 4. (**Subject specification semantics**). *Let* `sp` *be a signature policy,* `subj_spec(sp)` *be the subject specification in* `sp`*, having n* `cred_expr` *subelements. The semantics of* `subj_spec(sp)` *is defined as follows:*

$$\Psi(\texttt{subj\_spec(sp)})\overset{def}{=} \{(s_1, \ldots, s_n) \in \times_{i=1}^{n} \Phi(\texttt{cred\_expr[i]})| \ s_i \neq s_j, \ \forall i \neq j\}$$

where $cred\_expr[i]$ denotes the i-th *cred_expr* subelement of `subj_spec(sp)`.

Whereas the object specification semantics of a signature policy is the set of portions, denoted by the `path` attribute of a document d or all the documents, instances of a DTD/XML Schema, depending on that the `target` attribute of the policy denotes the document d or a DTD/XML Schema. Let *d_name* be the name of an XML document d and *path* an XPath compliant expression, then:

$$node\_set(d\_name, path) \overset{def}{=} \{n\_id | n\_id \text{ is the id of a node in } d \text{ satisfying } path\}$$

We use the notation $Inst(dtd\_name/X\_name)$ to denote the set of names of documents instance of the DTD/XML Schema with name dtd_name/X_name.

Definition 5. (**Object specification semantics**). *Let* `sp` *be a signature policy and let* `obj_spec(sp)` *be the object specification in* `sp`*. The semantics of* `obj_spec(sp)` *is defined as follows:*

$$\delta(\texttt{obj\_spec(sp)}) \overset{def}{=} \begin{cases} node\_set(d\_name, path(\texttt{sp})) = A & \textit{if } \text{target}(\texttt{sp}) \textit{ denotes } d \\ \bigcup_{d\_name \in Inst(\texttt{target(sp)})} A & \textit{otherwise} \end{cases}$$

In the following, we make use of a particular function, defined as follows: given $n \in \mathbb{N} \cup \{*\}$, let $\mathcal{I}_\mathcal{E}$ be a set of element identifiers, $Succ^n : \mathcal{I}_\mathcal{E} \longrightarrow \mathbb{P}(\mathcal{I}_\mathcal{E})$ s.t. $Succ^n(e\_id) = X \subseteq \mathcal{I}_\mathcal{E}$, where 1) if $n = *$, X contains the identifiers of all the direct and indirect subelements of e; 2) if $n = 0$, $X = \emptyset$; 3) if $n \geq 1$, X contains the identifiers of all the subelements of *e_id*, that are at most n level down in the document/DTD/XML Schema hierarchy with respect to *e_id*. Now, we need to introduce the notion of *target nodeset of a signature policy.*

Definition 6. (**Target nodeset of a signature policy**). *Let* `sp` *be a signature policy, the target nodeset of* `sp` *is defined as follows:*

$$O(\texttt{sp}) = \delta(\texttt{obj\_spec(sp)}) \cup \left(\bigcup_{e\_id \in \delta(\texttt{obj\_spec(sp)})} Succ^{prop-opt(\texttt{sp})}(e\_id)\right)$$

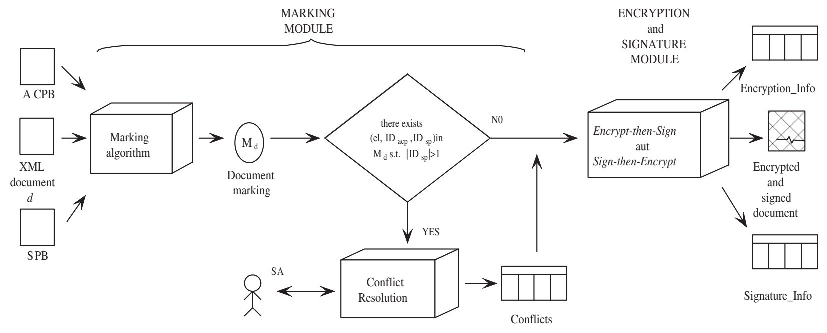

MARKING MODULE
ENCRYPTION and SIGNATURE MODULE
A CPB
XML document d
SPB
Marking algorithm
M d
Document marking
there exists (el, ID acp ,ID sp)in M d s.t. |ID sp|>1
N0
YES
Encrypt-then-Sign aut Sign-then-Encrypt
Encryption_Info
Encrypted and signed document
Signature_Info
SA
Conflict Resolution
Conflicts

Fig. 6. System architecture

We are now ready to introduce the semantics of a signature policy.

Definition 7. (**Signature policy semantics**). *Let* sp *be a signature policy. The semantics of* sp*, denoted as* $\mathcal{E}(\texttt{sp})$*, is defined as follows:*

$$\mathcal{E}(\texttt{sp})=\{(\overline{s},\texttt{o},\texttt{d}) \mid \overline{s} \in \Psi(\texttt{subj\_spec(sp)}),\ \texttt{o}\in O(\texttt{sp}),\ \texttt{d}=\texttt{duty(sp)}\}.$$

Example 3. Consider the third signature policy $\texttt{sp}_3$ in the $\mathcal{SPB}$ in Figure 5. The semantics of the subject and object specification of $\texttt{sp}_3$ are: $\Psi(\text{subj\_spec}(\texttt{sp}_3)) = \{(s_i, s_j) \in S^2 | \ s_i \neq s_j, \text{ with } i \neq j \ \}$, where S is the set of identifiers of members of the board of directors, $\delta(\text{obj\_spec}(\texttt{sp}_3)) = \{\&13\}$. Then, $\mathcal{E}(\texttt{sp}_3) = \{((s_i, s_j), \&13, \text{joint\_sign}) | \ (s_i, s_j) \in \Psi(\textit{subj\_spec}(\texttt{sp}_3))\}$.

4 System Architecture

In this section, we present the architecture we have designed for the generation of selectively encrypted and authenticated XML documents according to the stated access control and signature policies. The architecture, illustrated in Figure 6, consists of two main modules: the *Marking Module* and the *Encryption and Signature Module*. The *Marking Module* receives as input an XML document d, the access control policy and the signature policy base, and marks each portion of the document with the access control and signature policies, applied to that portion. When more than one signature policy applies to the same document portion (for instance a single and a joint signature policy), then a *Conflict Resolution* submodule is activated, that allows the SA to obtain only one signature policy for each document node, according to a strategy that will be illustrated in the following section; otherwise the *Encryption and Signature Module* is immediately activated. This module receives as input the document d, the *marking* of d, generated by the *Marking Module*, and the *Conflicts* table, generated by *Conflict Resolution* module, if it is activated; then, it executes both the encryption and signature operations. The result is an encrypted and signed document and two tables, *Encryption_Info* and *Signature_Info*, storing information that allows the receiver subjects to correctly decrypt and verify the authenticity of the

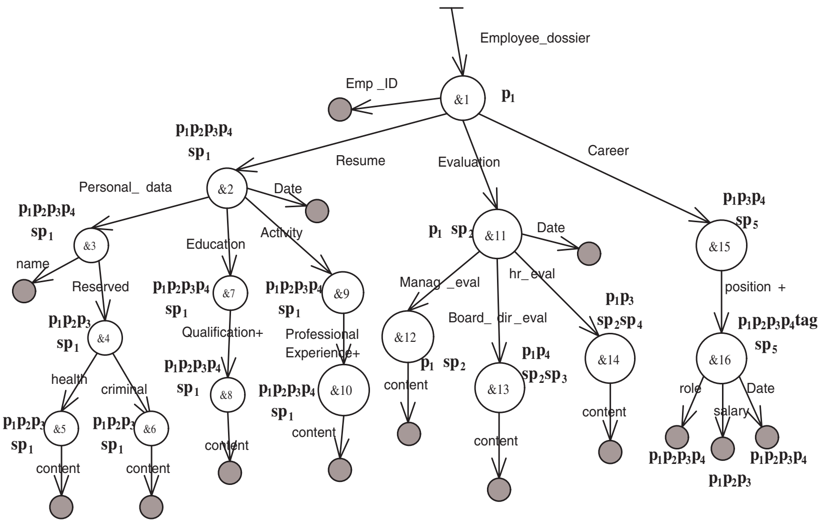


Fig. 7. Marking of the XML document in Figure 1

received document. The *Encryption and Signature Module* can operate according to two different strategies: *encrypt-then-sign* and *sign-then-encrypt*. If the first strategy is applied, the input document is first encrypted and then signed. By contrast, if the second strategy is adopted, the signatures are applied on the clear document and then the encryption process takes place.

4.1 Marking Module

The Marking Module generates a document marking which is defined as follows.

Definition 8. (**Document marking**). *Let $d = (V_d, \bar{v}_d, E_d, \phi_{E_d})$ be an XML document. A marking M_d of d is a set of triples (el, ID_{acp}, ID_{sp}), where:*

1. *the first component el can be one of the following:*
   - *an identifier of an element in d, that is, $el = e\_id, e\_id \in V_d^e \cup \{\bar{v}_d\}$;*
   - *an expression denoting an attribute in d or in one of its elements;*
   - *an expression of the form $el = e_id.$TAG, $e\_id \in V_d^e \cup \{\bar{v}_d\}$;*
2. *ID_{acp} is a set of identifiers of positive access control policies in $\mathcal{ACPB}$, applying on el, which are not overwritten by negative policies;*
3. *ID_{sp} is a set of identifiers of signature policies in $\mathcal{SPB}$, applying on el.*

An algorithm performing the marking is reported in Appendix A.

Example 4. Consider the XML document in Figure 1, the $\mathcal{ACPB}$, and $\mathcal{SPB}$ illustrated in Figures 3, and 5, respectively. The resulting marking is shown in Figure 7. In this figure, we denote an access control policy acp_i with p_i. Additionally, keyword TAG is used to denote that the policy marking applies only to the start and end tags of the corresponding element.

4.2 Conflict Resolution

If two different signature policies apply to the same nodeset, then a *Conflict Resolution* submodule is invoked. This submodule interacts with the SA, to decide how different signature policies, applying on the same nodeset, can be combined. This is a necessary step for signature policies because, differently from an access control policy, a signature policy states a duty instead of the granting of an access authorization. As an example, suppose that two different signature policies apply to the same protection object o: the first requires a signature by a technical manager, whereas the second requires the signature of the head of department. The problem is then, who must sign o. An option can be to combine the duties enacted by the two policies and requiring a joint signature by both the technical manager and the head of department. An alternative solution can be to consider one of the two policies as prevailing with respect to the other. However, whichever solution is adopted, there is the need of a conflict resolution phase, which decides how different signature policies applying to the same nodeset must be combined. A necessary step is, thus, to formally state the notion of *conflict* between two or more signature policies on a given document. First, we introduce the notion of *view of a document w.r.t. a signature policy*, which will be also necessary to formally state the notion of correct signature. This view is the set of portions of a document that must be signed according to the given policy.

Definition 9. (**View of a document w.r.t. a signature policy**). *Let d be an XML document and* sp *be a signature policy. The view of d w.r.t.* sp *is defined as follows:*

$$V_d(\mathtt{sp}) \overset{def}{=} \begin{cases} A \cup (\bigcup_{e\_id \in A} Succ^n(e\_id)) & \textit{if } \text{target}(\mathtt{sp}) \textit{ denotes } d \textit{ or its} \\ & \textit{DTD/ XML-Schema} \\ \emptyset & \textit{otherwise} \end{cases}$$

where $A = node\_set(d\_name, path(\mathtt{sp}))$ and n=$prop\_opt(\mathtt{sp})$.

Definition 10. (**Conflict on an XML document**). *Let d be an XML document and $\mathcal{SPB}$ be a signature policy base, specified for the source $\mathcal{S}$ containing d. Let $\{sp_k\}_{k\in I}$ be a set of signature policies identifiers applying on d, where I is a set of some id attributes of signature policies in $\mathcal{SPB}$. We say that there is a* conflict *among the policies $\{sp_k\}_{k\in I}$ on d if the following hold:*

1. $\bigcap_{k\in I} V_d(\mathtt{sp_k}) \neq \emptyset$;
2. $\Psi(subj\_spec(\mathtt{sp_i})) \neq \Psi(\mathtt{subj\_spec}(\mathtt{sp_j}))$, $\forall i, j \in I, i \neq j$.

In order to support *conflict resolution*, our system provides the SA with a set of operators on sets of signature policies, which allow him/her to associate with them a single signature policy. Given a set of conflicting signature policies, $\{sp_k\}_{k\in I} \subseteq \Pi(\mathcal{SPB})$, applying on the same nodeset of a document d, we define the following operators: the *choice* function f_k, with $k \in I$, the *concatenation* '$||$', and the *and* operator '$\wedge$'. The choice function f_k, $k \in I$ allows the SA to associate with the policies in X that signature policy, whose identifier *id* is equal to k; the concatenation operator '$||$' allows the SA to associate with policies in X a signature policy, whose duty attribute is *joint_sign* and whose subject specification contains the concatenation of the credential expressions of each component policy. The *and* operator '$\wedge$' applies only on signature policies whose duty attribute is *sign* and gives as result a signature policy, whose duty is still *sign* and whose cred_expr is the composition through the XPath symbol '|' of each credential expression of the component policies. Note that, the resulting signature policy must apply only on the nodeset of the document on which a conflict is verified, that is, that resulting from the intersection of the views of the document with respect to the conflicting policies.

Definition 11. (**Conflict resolution operators**). *Given a set of conflicting signature policies on an XML document d, $X = \{sp_k\}_{k \in I}$, $|I| = n$:*

(i) *if* $\forall k \in I$, $\mathtt{duty}(\mathtt{sp_k}) = \mathtt{sign}$, $'\wedge'(X) = \mathtt{sp}_{k_1} \wedge \ldots \wedge \mathtt{sp}_{k_n} \overset{def}{=} \bar{\mathtt{sp}}$, *where* $\bar{\mathtt{sp}}$ *is a signature policy whose objects specification element denotes* $\bigcap_{k \in I} V_d(\mathtt{sp_k})$, *propt_opt*$(\bar{\mathtt{sp}})$*=0 duty*$(\bar{\mathtt{sp}})$*= sign and cred_expr*$(\bar{\mathtt{sp}})$ *contains cred_expr*$(sp_{k_1})|\ldots|$*cred_expr*(sp_{k_n})*;*

(ii) $\forall k \in I$, $f_k(X) \overset{def}{=} sp_k$*;*

(iii) $\forall k \in I$, $'||'(X) = \mathtt{sp}_{k_1}||\ldots||\mathtt{sp}_{k_n} \overset{def}{=} \bar{\mathtt{sp}}$, *where* $\bar{\mathtt{sp}}$ *is a signature policy whose objects specification element denotes* $\bigcap_{k \in I} V_d(\mathtt{sp_k})$, *propt_opt*$(\bar{\mathtt{sp}})$*=0, duty*$(\bar{\mathtt{sp}})$*= joint_sign and subj_spec*$(\bar{\mathtt{sp}})$ *contains the credential expressions of all the component policies.*

Such modules notifies the SA of conflicts among signature policies and displays the nodes on which more than one signature policy applies. Then, the SA can decide how to solve the conflict, choosing among the operators previously introduced. Note that the Conflict resolution phase is not limited to the definition of the conflict resolution operators, since it is also necessary to modify the nodeset of conflicting policies to exclude those nodes that are covered by the resulting signature policy. Thus, the '*restrict*' operator is defined as follows: given a set of conflicting signature policies on d, $\{sp_k\}_{k \in I}$, if $V_d(\mathtt{sp_k}) - \bigcap_{\mathtt{i} \in \mathtt{I}} \mathtt{V_d}(\mathtt{sp_i}) \neq \emptyset$ for some $k \in I$, the operator '*restrict*' applies on $\{\mathtt{sp_k}\}$ and gives as result that signature policy `sp'` having the same semantics of sp_k with the only exception of the view of the document w.r.t. the signature policy $V_d(\mathtt{sp}') = V_d(\mathtt{sp}_k) - A$, where A is the nodeset on which the signature policy $\mathtt{sp}_k$ is in conflict with the other signature policies in $\mathcal{SPB}$ applying on d. Moreover, to uniformly treat both conflicting and non conflicting policies, the system uses the operator *identity* for those signature policies that are not in conflict with other policies on document d; *identity*$(\{\mathtt{sp}_k\}) = \mathtt{sp}_k$, $\forall \mathtt{sp}_k \in \Pi(\mathcal{SPB})$. The output of the Conflict resolution phase is a table, called **Conflicts**, with *Config*, and *Operator* as attributes: this table contains an entry for each signature policy configuration $ID_{sp} \neq \emptyset$. For each ID_{sp} with $|ID_{sp}| > 1$ the table contains operator specified by the SA in order to solve the conflict among the policies in ID_{sp}; for each ID_{sp} with $|ID_{sp}| = 1$, if there exists an ID'_{sp} with $|ID'_{sp}| > 1$ such that $ID_{sp} \cap ID'_{sp} \neq \emptyset$ the Operator attribute is set to *restrict*, otherwise it is set to *identity*.

Example 5. With reference to the document marking in Figure 7, we observe that there are two different signature policies applying to the same nodeset, having a different subject specification semantics: $\mathtt{sp}_2$, $\mathtt{sp}_3$ and $\mathtt{sp}_2$, $\mathtt{sp}_4$. The SA must decide how these signature policies must be combined. For example, the SA can decide to combine $\mathtt{sp}_2$, $\mathtt{sp}_3$ and $\mathtt{sp}_2$, $\mathtt{sp}_4$ through the concatenation operator. As a result, node &13 must be signed by two members of the board of directors and by the manager, node &14, instead, must be signed by the head of the human resouces department together with the manager. The resulting *Conflicts* table is represented in Table 1, where $ID^{i}_{sp} = \{\mathtt{sp}_1\}$, $ID^{ii}_{sp} = \{\mathtt{sp}_2\}$, $ID^{iii}_{sp} = \{\mathtt{sp}_2, \mathtt{sp}_3\}$, $ID^{iv}_{sp} = \{\mathtt{sp}_2, \mathtt{sp}_4\}$, $ID^{v}_{sp} = \{\mathtt{sp}_5\}$.

Table 1. Conflicts Table

Config	Operator
ID^{i}_{sp}	*identity*
ID^{ii}_{sp}	*restrict*
ID^{iii}_{sp}	*concatenation*
ID^{iv}_{sp}	*concatenation*
ID^{v}_{sp}	*identity*

4.3 Correct Signature

An important requirement is to formally state when a document conforms to the signature policies and, in case of conflicts among the policies on the given document, to the conflict resolution operators specified for the document. For this reason, we introduce the notion of correct signature, that is, a signature that allows the authentication of the correct view of a document.

Definition 12. (**Correct signature**). *Let d be an XML document, $\mathcal{SPB}$ be the signature policy base specified for the source $\mathcal{S}$ containing d. The signature of document d is said to be* correct *iff both the following hold:*

1. *for each signature policy specified in $\mathcal{SPB}$, applying on d, whose identifier is sp_i, $\exists \overline{s} \in \Psi(\mathtt{subj\_spec}(sp_i))$ s.t. $V_d(sp_i)$ is authenticated by all the subjects in $\overline{s}$, if there are not conflicts on d among the signature policies in $\mathcal{SPB}$;*
2. *for each signature policy in $\mathcal{SPB}$, applying on d, whose identifier is sp_i, $\exists \overline{s} \in \Psi(\mathtt{subj\_spec}(f(X)))$ s.t. $V_d(f(X))$ is authenticated by all the subjects in $\overline{s}$, $\forall X$, set of signature policies identifiers containing sp_i applying on the same nodeset of d, and f is the operator in the $Conflicts$ Table corresponding to X, if there are conflicts among the signature policies in $\mathcal{SPB}$ on d.*

4.4 Encryption and Signature Module

The *Encryption and Signature Module* can work according to two different and alternative strategies: (1) all document portions to which the same access control policies apply are encrypted with the same key. Then, the signatures are applied on the encrypted document; (2) the document is first signed according to the defined signature policies. Then, all the document portions to which the same access control policies apply are encrypted with the same key.

Both the strategies have their advantages and drawbacks. The encrypt-then-sign strategy has the main advantage of limiting the number of unnecessary decryption operations. Indeed, the receiver subject can verify the correct signature of the received document without decrypting it. But, in this case the receiver subject has to store also the encrypted portion he/she can access in order to be able to validate the signature subsequently [6]. By contrast, the sign-then-encrypt strategy has the advantage that the document server is not forced to apply signatures on an encrypted content as in the other strategy. On the other side, security considerations impose on us the encryption of digital signatures along with the signed data. The SA can select the strategy that better fits the characteristics of the considered application domain, such as the security requirements, the time costs of the operations and so on. The algorithms for both strategies are presented in Figures 9, and 10, respectively in Appendix A. Both algorithms receive as input a document, its marking and the `Conflicts` table, whenever some of the signature policies applying on the document are in conflicts; they generate the encrypted and signed document along with the tables `Encryption_Info` and `Signature_Info`, which contains information allowing the receiver subjects to correct decrypt and verify the authenticity of the document portions for which he/she has the authorization.

The notion of correct signature has been introduced in Section 4.3, whereas the notion of correct encryption [3] refers to the possibility for each subject of obtaining all and only the keys necessary to decrypt the document portions for which he/she has a proper authorization. When we selectively encrypt a document that will be authenticated or that has been signed according to the stated signature policies, we have to operate in such way that the receiving subject is able to verify the authenticity of the document portions for which he/she has the authorization. This is a non trivial issue. Indeed, consider the sign-then-encrypt strategy and suppose that a subject signs a document portion which must be encrypted with different keys. Suppose moreover than one of the receiver subjects has the authorization to view only a part of the signed portion. Thus, he/she is not able to validate the signature. For this reason, we have to introduce the following definition, that always make a subject able to verify the signatures applying on document portions he/she is authorized to access.

In the following, $\mathcal{SPB}'$ denotes an XML document, structured as $\mathcal{SPB}$, containing the signature policies specified during the Conflict Resolution phase.

Definition 13. (**Correct composition of encryption and signature operations**). *Let $\mathcal{S}$ be an XML source, $\mathcal{ACPB}$ and $\mathcal{SPB}$ be the access control and signature policy base respectively, defined for $\mathcal{S}$ and let d be an XML document in $\mathcal{S}$. Let S_d be the identifiers of subjects to which the encryption of d should be delivered. The composition of encryption and signature operations is said to be* correct *iff the following requirements hold:*

1. *the encryption of document d is correct;*
2. *the signature of document d is correct;*

3. $\forall s \in S_d$ *and for each access control policy in* $\mathcal{ACPB}$ *with identifier* $\texttt{acp}_i$ *such that* $s \in \Phi(cred\_expr(\texttt{acp}_i))$*, and for each signature policy in* $\mathcal{SPB}$ *or in* $\mathcal{SPB}'$ *with identifier* $\texttt{sp}_j$ *such that* $V_d(\texttt{acp}_i) \cap V_d(\texttt{sp}_j) \neq \emptyset$*, then* s *must be able to verify the authenticity of* $V_d(\texttt{acp}_i) \cap V_d(\texttt{sp}_j)$*.*

We can prove that the proposed algorithms generate a correct composition of signature and encryption operations.

Theorem 1. *Let* $\mathcal{S}$ *be an XML source,* $\mathcal{ACPB}$ *and* $\mathcal{SPB}$ *be the access control and signature policy base respectively, defined for* $\mathcal{S}$*. The composition of encryption and signature operations realized through Algorithms 1, 2 or 3 is correct.*

5 Concluding Remarks

In this paper, we have proposed an XML-based language for specifying credential-based signature policies. The language allows the specification of both single and joint signatures. Additionally, we have described the system architecture supporting the generation of selectively encrypted and signed XML documents, according to the stated access control and signature policies. Future work includes the development of a prototype implementation of the proposed system, and an extensive investigation of the performance of the two strategies we have proposed (i.e., sign-then-encrypt and encrypt-then-sign). Moreover, we plan to apply and, possibly extend, our approach to SOAP. Finally, we plan to address the problem of signature policy conflicts, to devise semiautomatic conflict resolution and analysis techniques for policy bases in order to detect potential conflicts when policies are specified, and to allow conflicts resolution also at DTD/XML Schema level.

References

1. J. H. An, Y. Dodis, and T. Rabin. On the Security of Joint Signature and Encryption. *EUROCRYPT, 2002.*
2. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT 2000.*
3. E. Bertino and E. Ferrari. Secure and Selective Dissemination of XML Documents, *ACM Transactions on Information and System Security*, vol. 5, n. 3, 2002.
4. E. Bertino, S. Castano and E. Ferrari. On Specifying Security Policies for Web Documents with an XML-based Language. In *Proc. of the 1st ACM Symposium on Access Control Models and Technologies (SACMAT'01)*, Chantilly, Virginia, USA, Maggio 2001. ACM Press.
5. C. Geuer Pollmann. The XML Security Page. http://www.nue.et-inf.uni-siegen.de/˜geuer-pollmann/xml_security.html
6. W. Stallings. Network Security Essentials: Applications and Standards. *Prentice Hall, 2000.*
7. Word Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. Available at `http://www.w3.org/TR/REC-xml`

A Algorithms

The marking algorithm receives as input a document, the access control policy base and the signature policy and generates the corresponding marking. This algorithm makes use of function 'Add()'. The result of the statement 'Add $(el, \texttt{acp}_i, \ldots)$ to M_d' is the addition of element $(el, ID_{acp}, \emptyset)$ to marking M_d, where $ID_{acp} = \{\texttt{acp}_i\}$, if there does not exist a triple containing el in M_d. Otherwise, it is the addition of $\texttt{acp}_i$ to the set of access control policies identifiers, ID_{acp}, in that triple. Moreover the statement 'Add $(el, \ldots, \texttt{sp}_i)$ to M_d' is the addition of element $(el, \emptyset, ID_{sp})$ to marking M_d, where $ID_{sp} = \{\texttt{sp}_i\}$, if there does not exist a triple containing el in M_d. Otherwise, it is the addition of the signature policy identifier $\texttt{sp}_i$ to the set of signature policies identifiers, denoted with ID_{sp}, third component of that triple, whose first component is el.

The Algorithms presented in Figures 9, and 10, respectively, first, generate the sets MARK1 and MARK2, whose elements are sets of access control and signature policies identifiers, belonging to the document marking. For each element l in MARK1, they generate a family of sets, $\{\mathrm{E}_l\}_{l \in MARK1}$, each of those contains elements and attributes marked with the access control policy configuration identified by l; moreover, they generate the set of all elements which are not covered by any access control policy, identified by *DEFAULT* string. Then, if the `Encrypt-then-Sign` Algorithm is selected, the family $\{\mathrm{T}_m\}_{m \in MARK2}$ is computed, where T_m is the set of elements and attributes to which the same signature policy configuration m applies. The algorithm builds a table, named `Signature_Info`: each entry of this table has the form (m, O_m), where O_m is a list of offsets, which allows one to identify the portion T_m in the encrypted document, which is signed according to m. Then, the encryption operation is executed: for each distinct group E_l, the algorithm generates a different key, which is used to encrypt the nodes members of the group. The algorithm builds a second table, named `Encryption_Info`, that contains information on the keys implied by each access control policy configuration that can be applied to the input document. Each entry of the table has the form (l, E_l, key), where l belongs to MARK1, E_l is the set of nodes marked with the access control policy identifier l, and *key* is an encryption key generated for this portion. Finally, the signing process takes place using the strategies defined by the W3C for the XML Signature [7]. For each signature policy identifier m in MARK2 an XML document, called Signatures with m as identifier, is built: it will contain all the XML Signatures of T_m, according to the signature policies specified in $\mathcal{SPB}$ or resulting from the Conflict Resolution phase, whenever there are conflicts among the policies in $\mathcal{SPB}$. Differently from Algorithm 2, the `Sign-then-Encrypt` Algorithm first generates the signatures of the document and then encrypts each block E_l along with all the signatures of portion of E_l, using the strategies illustrated above. In order to allow a correct composition of encryption and signature operations, each subject, who has to sign a document portion X, must affix one signature on each non empty portion $X \cap E_l$, where l belongs to MARK1. Thus, given a document portion X whose nodes are marked with the signature policy identifier m, $m \in$ MARK2, Algorithm 3 generates the families $\{T_{lm}\}_{l \in MARK1}$,

Algorithm 1 *The Marking Algorithm*

```
INPUT:   An XML document $d = (V_d, \bar{v}_d, E_d, \phi_{E_d})$, $\mathcal{ACPB}$ and $\mathcal{SPB}$.
OUTPUT:  The marking $M_d$ of document $d$

 1. Let $\Pi_d(\mathcal{ACPB}) = \{\texttt{acp}_i \mid i = id$, where $id$ is the attribute of a browsing policy in $\mathcal{ACPB}$
                        s.t. $\exists$ (s,o,p) $\in \mathcal{E}(\texttt{acp}_i)$ [1] where the $id$ of $d$ appears in o$\}$
 2. $M_d$ is initialized to be empty
 3. For each $\texttt{acp}_i \in \Pi_d(\mathcal{ACPB})$:
      (a) If path($\texttt{acp}_i$) denotes a set of elements of $d$:
            Let $E' = \{id \mid id$ is the identifier of an element in $d$ denoted by path($\texttt{acp}_i$)$\}$
            Let $E^* = \{e\_id \mid e'\_id \in E'$ and $e\_id \in Succ^{\texttt{prop-opt}(\texttt{acp}_i)}(e'\_id)\}$
            For each $e\_id \in E' \cup E^*$:
                case priv($\texttt{acp}_i$) of
                browse-all: Add ($e\_id$,$\texttt{acp}_i$,...) to $M_d$
                view: If $e\_id$ does not have IDREF(s)/URI(s) attributes: Add ($e\_id$,$\texttt{acp}_i$,...)
                        to $M_d$
                      else: For each non IDREF(s)/URI(s) attributes $a$ in $e\_id$:
                            Add ($e\_id.a$,$\texttt{acp}_i$,...) to $M_d$
                navigate: If $e\_id$ does not have non IDREF(s)/URI(s) attributes:
                            Add ($e\_id$,$\texttt{acp}_i$,...) to $M_d$
                          else: For each IDREF(s)/URI(s) attributes $a$ in $e\_id$:
                            Add ($e\_id.a$, $\texttt{acp}_i$,...) to $M_d$
      (b) If path($\texttt{acp}_i$) denotes a set of attributes $A$:
            For each $a \in A$:
                Let $e\_id$ be the identifier of the element to which $a$ belongs to
                Add ($e\_id.a$,$\texttt{acp}_i$,...) to $M_d$
    endfor
 4. For each $e\_id \in \mathcal{I}_{\mathcal{E}}$ such that $\exists(el, ID_{acp}, \ldots) \in M_d$, $el = e\_id.a$:
      Let $A$ be the set of attributes in $e\_id$
      Let $S = \bigcup_{a \in A} ID_{acp}$, where $ID_{acp} \subseteq \mathcal{ACPB}$ s.t. $(e\_id.a, ID_{acp}, \ldots) \in M_d\}$
      If $\exists ID'_{acp} \subseteq \mathcal{ACPB}$ such that $(e\_id, ID'_{acp}, \ldots) \in M_d$
         Replace $(e\_id, ID'_{acp}, \ldots)$ with $(e\_id.\textsc{tag}, ID'_{acp} \cup S, \ldots) \in M_d$
         For each $a \in A$: Add $(e\_id.a, ID'_{acp}, \ldots)$ to $M_d$
      else: Add$(e\_id.\textsc{tag}, S, \ldots)$ to $M_d$
    endfor
 5. Let $\Pi_d(\mathcal{SPB}) = \{\texttt{sp}_i \mid i = id$, where $id$ is the attribute of a policy in $\mathcal{SPB}$
                        s.t. $\exists$ ($\bar{s}$,o,d) $\in \mathcal{E}(\texttt{sp}_i)$ where the $id$ of $d$ appears in o$\}$
 6. For each $\texttt{sp}_i \in \Pi_d(\mathcal{SPB})$:
      (a) If path($\texttt{sp}_i$) denotes a set of elements of d:
            Let $E' = \{id \mid id$ is the identifier of an element in $d$ denoted by path($\texttt{sp}_i$)$\}$
            Let $E^* = \{e\_id \mid e'\_id \in E'$ and $e\_id \in Succ^{\texttt{prop-opt}(\texttt{sp}_i)}(e'\_id)\}$
            For each $e\_id \in E' \cup E^*$: Add ($e\_id$,...,$\texttt{sp}_i$) to $M_d$
      (b) If path($\texttt{sp}_i$) denotes a set of attributes $A$:
            For each $a \in A$:
                Let $e\_id$ be the identifier of the element to which $a$ belongs to
                Add ($e\_id.a$,...,$\texttt{sp}_i$) to $M_d$
    endfor
```

Fig. 8. The Marking Algorithm

where $T_{lm} = X \cap E_l$, that is, the document portion encrypted with the same key to which the signature policy m applies. Then, for each element l in MARK1, an XML document, called Signatures with l as identifier, is built: it will contain all the XML Signatures of T_{lm}, according to the signature policies specified in $\mathcal{SPB}$ or resulting from the Conflict Resolution phase, whenever there are conflicts among the policies in $\mathcal{SPB}$. Finally, each block E$_l$ along with the XML document and the Signatures with l as ID attribute, is encrypted.

Algorithm 2 *The Encrypt-then-Sign Algorithm*

```
INPUT:   An XML document $d$, its marking $M_d$ [the table *Conflicts*].
OUTPUT:  $d^e$, the encrypted and signed version of $d$, *Encryption_Info* and *Signature_Info*.

 1. MARK1 $= \emptyset$, MARK2 $= \emptyset$
 2. Let $d'$ be a copy of $d$
 3. For each $(el, ID_{acp}, ID_{sp}) \in M_d$:
        If $((ID_{acp} \neq \emptyset)$ and $(ID_{acp} \notin$ MARK1$))$: Add $ID_{acp}$ to MARK1
        If $((ID_{sp} \neq \emptyset)$ and $(ID_{sp} \notin$ MARK2$))$: Add $ID_{sp}$ to MARK2
 4. For ēach $l \in$ MARK1: Let $E_l = \{el \mid (el, ID_{acp}, ID_{sp}) \in M_d$ and $ID_{acp} = l\}$
 5. l = DEFAULT
 6. Let $E_l$ be the complementary set of $\bigcup_{r \in MARK1} E_r$ in $V_d^e \cup V_d^a$
 7. If $E_l \neq \emptyset$: Add l to MARK1
 8. For each $m \in$ MARK2:
        Let $T_m = \{el \mid (el, ID_{acp}, ID_{sp}) \in M_d$ and $ID_{sp} = m\}$
        Compute the list $O_m$ of sequences of offsets of $T_m$
        Insert $(m, O_m)$ in table *Signature_Info*
 9. For each $l \in$ MARK1:
        Generate a key $k$
        Add $(l, E_l, k)$ to *Encryption_Info*
        For each $el \in E_l$:
            If $el$ denotes a node $n$: Encrypt $n$ in $d'$ with $k$
            If $el$ denotes the start and end tags of an element $e$:
               Encrypt the tags of $e$ in $d'$ with $k$
10. If there are conflicts in $\mathcal{SPB}$:
      For each $m \in$ MARK2:
          Generate an XML document Signatures with $m$ as signature id
          Select Operator from table *Conflicts* where Config attribute is equal to $m$
          Case Operator = $identity$ or $restrict|choice_\alpha$
            For each $i \in [1, count(cred\_expr(sp_k))]$ with $sp_k \in m|k = \alpha$
                Insert a detached XML Signature into Signatures document
                Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
                If $s \neq s_j$ with $1 \leq j \leq i-1$
                  Require the signature of SignedInfo element by the subject $s$
                  $s_i = s$
                  Insert the public key of $s$ in KeyInfo
          Case Operator = $and$
            Insert a detached XML Signature into Signatures document
            Select a subject $s \in \bigcap_{sp_k \in m} \Phi(cred\_expr(sp_k))$:
            Require the signature of SignedInfo element by the subject $s$
            Insert the public key of $s$ in KeyInfo
          Case Operator = $concatenation$
            For each $sp_k \in m$ :
                For each $i \in [1, count(cred\_expr(sp_k)]$
                    Insert a detached XML Signature into Signatures document
                    Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
                    If $s \neq s_j$ with $1 \leq j \leq i-1$
                     Require the signature of SignedInfo element by the subject $s$
                     $s_i = s$
                     Insert the public key of $s$ in KeyInfo
    else
      For each $m \in$ MARK2:
          Generate an XML document Signatures with $m$ as signature id
          For each $i \in [1, count(cred\_expr(sp_k))]$ with $sp_k \in m$
              Insert a detached XML Signature into Signatures document
              Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
              If $s \neq s_j$ with $1 \leq j \leq i-1$
                Require the signature of SignedInfo element by the subject $s$
                $s_i = s$
                Insert the public key of $s$ in KeyInfo
    endif
```

Fig. 9. The Encrypt-then-Sign Algorithm

Algorithm 3 *The Sign-then-Encrypt Algorithm*

```
INPUT:   An XML document $d$, its marking $M_d$ [and the Table $Conflicts$].
OUTPUT:  $d^e$, the encrypted and signed version of $d$, $Encryption\_Info$ and $Signature\_Info$.

 1. MARK1 = $\emptyset$, MARK2 = $\emptyset$
 2. Let $d'$ be a copy of $d$
 3. For each $(el, ID_{acp}, ID_{sp}) \in M_d$:
        If $((ID_{acp} \neq \emptyset)$ and $(ID_{acp} \notin$ MARK1$))$: Add $ID_{acp}$ to MARK1
        If $((ID_{sp} \neq \emptyset)$ and $(ID_{sp} \notin$ MARK2$))$: Add $ID_{sp}$ to MARK2
 4. For ēach $l \in$ MARK1: Let $E_l = \{el \mid (el, ID_{acp}, ID_{sp}) \in M_d$ and $ID_{acp} = l\}$
 5. l = DEFAULT
 6. Let $E_l$ be the complementary set of $\bigcup_{r \in MARK1} E_r$ in $V_d^e \cup V_d^a$
 7. If $E_l \neq \emptyset$: Add l to MARK1
 8. For each $l \in$ MARK1:
        Let $T_{lm} = \{el \mid el \in E_l, ID_{sp} \neq Null, m = ID_{sp}\}$
        Generate an XML document Signatures with $l$ as ID attribute
 9. For each $m \in$ MARK2:
        For each $T_{xy}$, where $y = m$
            Compute the list $O_{xm}$ of sequences of offsets of $T_{xm}$
            Insert $((x, m), O_{xm})$ in Table $Signature\_Info$
            If there are conflicts in $\mathcal{SPB}$:
             Select Operator from table $Conflicts$ where Config attribute is equal to $m$
             Case Operator = $identity$ or $restrict|choice_\alpha$
                For each $i \in [1, count(cred\_expr(sp_k))]$ with $sp_k \in m|k = \alpha$
                    Insert a detached XML Signature into Signatures document with $x$ as id
                    Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
                    If $s \neq s_j$ with $1 \leq j \leq i-1$
                      Require the signature of SignedInfo element by the subject $s$
                      $s_i = s$
                      Insert the public key of $s$ in KeyInfo
             Case Operator = $and$
                Insert a detached XML Signature into Signatures document with $x$ as id
                Select a subject $s \in \bigcup_{sp_k \in m} \Phi(cred\_expr(sp_k))$:
                Require the signature of SignedInfo element by the subject $s$
                Insert the public key of $s$ in KeyInfo
            Case Operator = $concatenation$
             For each $sp_k \in m$ :
                 For each $i \in [1, count(cred\_expr(sp_k)]$
                     Insert a detached XML Signature into Signatures document with $x$ as id
                     Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
                     If $s \neq s_j$ with $1 \leq j \leq i-1$
                      Require the signature of SignedInfo element by the subject $s$
                      $s_i = s$
                      Insert the public key of $s$ in KeyInfo
            else
             For each $i \in [1, count(cred\_expr(sp_k))]$ with $sp_k \in m$
                 Insert a detached XML Signature into Signatures document with $x$ as id
                 Select a subject $s \in \Phi(cred\_expr[i](sp_k))$
                 If $s \neq s_j$ with $1 \leq j \leq i-1$
                     Require the signature of SignedInfo element by the subject $s$
                     $s_i = s$
                     Insert the public key of $s$ in KeyInfo
10. For each $l \in$ MARK1:
        Generate a key $k$
        Add $(l, E_l, k)$ to $Encryption\_Info$
        For each $el \in E_l$:
            If $el$ denotes a node $n$: Encrypt $n$ in $d'$ with $k$
            If $el$ denotes the start and end tags of an element $e$:
                Encrypt the tags of $e$ in $d'$ with $k$
                Encrypt the Signatures document with $l$ as id
    endfor
```

Fig. 10. The Sign-then-Encrypt Algorithm

B Proof

Proof of Theorem 1. Let $\mathcal{S}$ be an XML source, $\mathcal{ACPB}$ and $\mathcal{SPB}$ be the access control and signature policy base respectively, defined for $\mathcal{S}$, and let d be an XML document in $\mathcal{S}$ and let S_d be the identifiers of subject to which the encryption of d should be delivered. By Definition 13 we have to prove:

1. the encryption of document d is correct;
2. the signature of document d is correct;
3. $\forall s \in S_d$ and for each access control policy in $\mathcal{ACPB}$ with identifier $\texttt{acp}_i$ such that $s \in \Phi(cred\_expr(\texttt{acp}_i))$, and for each signature policy in $\mathcal{SPB}$ or in $\mathcal{SPB}'$, in case of conflicts among the policies in $\mathcal{SPB}$ on document d, with identifier $\texttt{sp}_j$ such that $V_d(\texttt{acp}_i\ ) \cap V_d(\texttt{sp}_j) \neq \emptyset$, then s must be able to verify the authenticity of $V_d(\texttt{acp}_i) \cap V_d(\texttt{sp}_j)$.

for each document d belonging to $\mathcal{S}$ or istance of a DTD/XML Schema in $\mathcal{S}$.

We do not report here the correctness proof for encryption since it is very similar to the proof of Theorem 4.1 of [3]. First of all, we concentrate on the correctness proof for signature. Let d be an XML document in source $\mathcal{S}$. We consider a signature policy specified in $\mathcal{SPB}$, applying on d, whose identifier is $sp_i \in \Pi(\mathcal{SPB})$, since $\texttt{target(sp}_i\texttt{)}$ denotes the document d or its DTD/XML Schema. Thus, set $\Pi_d(\mathcal{SPB})$ computed by step 5) of Algorithm 1 includes sp_i. Thus, sp_i is considered during the execution of step 6) of the algorithm. Several cases can arise, based on the form of $\texttt{path(}sp_i\texttt{)}$. Let us consider all these cases in turn:

1. $\texttt{path(}sp_i\texttt{)}$ denotes a set of elements in d. Let E be the set of the identifiers of the elements of d denoted by $\texttt{path(}sp_i\texttt{)}$. Step 6a) of Algorithm 1 is executed. By Definition 9, $V_d(sp_i)$ depends on the propagation option in sp_i. If $\texttt{prop\_opt(}sp_i\texttt{)} = \texttt{*}$, then the view contains all the elements in E and all the subelements of elements in E. If $\texttt{prop\_opt(}sp_i\texttt{)} = \texttt{0}$, then the view contains all the elements in E, whereas if $\texttt{prop\_opt(}sp_i\texttt{)} = \texttt{n}$, $\texttt{n} \geq 1$, then the view contains all the elements in E and all the subelements of elements in E from the direct subelements going down $\texttt{n}$ levels in the document hierarchy. In all the above cases, the set of identifiers of the elements belonging to the view, is equal to set $E^{'} \cup E^{*}$, where E^{*} and $E^{'}$ are the sets computed at the beginning of step 6a). Thus, at the end of step 6a) for all and only $e\_id \in E^{'} \cup E^{*}$, $\exists(el, \ldots, ID_{sp}) \in M_d$ such that $el = e\_id$ and $\texttt{sp} \in ID_{sp}$, that is elements in $E^{'} \cup E^{*}$ are marked with ID_{sp}. No other element of d is marked with ID_{sp}.
2. $\texttt{path(}sp_i\texttt{)}$ denotes a set of attributes of document d. Let A be the set of such attributes. In this case, by Definition 9, $V_d(\texttt{sp})$ is equal to the set of elements that contain at least an attribute in A, from which all the attributes which do not appear in A have been removed. Since $\texttt{path(}sp_i\texttt{)}$ denotes a set of attributes, step 6b) of Algorithm 1 is executed. At the end of this step, for all and only the attributes $a \in A$, $\exists(el, \ldots, ID_{sp}) \in M_d$ such that $el = e\_id.a$, $e\_id \in \mathcal{I_E}$, $sp_i \in ID_{sp}$, where e_id is the identifier of the element that contains attribute a. It is easy to verify that for all and only the attributes $a \in A$, $\exists(el, \ldots, ID_{sp}) \in M_d$ such that el equal to $e\_id.a$, $\texttt{sp} \in ID_{sp}$, where e_id is the identifier of the element that contains a. Thus, the marking phase takes place correctly.

Then, we prove that the signature operation is performed according to the document marking and the At first we suppose that there are conflicts among the signature policies in $\mathcal{SPB}$ on document d. Since $sp_i \in \Pi_d(\mathcal{SPB})$, there exists at least a signature policies configuration ID_{sp} such that $ID_{sp} \supseteq sp_i$. Let f be the operator corresponding to ID_{sp} in table *Conflicts*, if $|ID_{sp}| > 1$, for each $\bar{sp} = f(ID_{sp})$ $V_d(\bar{\mathtt{sp}}) = \bigcap_{\mathtt{sp}_k \in ID_{sp}} V_d(\mathtt{sp}_k)$ and it is easy to prove that $V_d(\bar{\mathtt{sp}})$ is equal to the following set $\{el|\ (el, ID_{acp}, ID'_{sp}) \in M_d$ s.t.$ID'_{sp} = ID_{sp}\}$. The *Encrypt-then-Sign* Algorithm selects the set $T_m = \{el|\ (el, ID_{acp}, ID_{sp}) \in M_d$ s.t.$ID_{sp} = m\}$, which is equal to $V_d(\bar{\mathtt{sp}})$, whereas the *Sign-then-Encrypt* Algorithm selects for each l the set T_{lm}, such that $\bigcup_l T_{lm}$ is equal to $V_d(\bar{\mathtt{sp}})$. Both the algorithms require for this nodeset the signature of a tuple of subjects according to the operator corresponding to ID_{sp} in table *Conflicts*, specified by the SA. If the Operator attribute is equal to '*and*', the system selects a subject s belonging to $\bigcap_{sp_k \in ID_{sp}=m} \Phi(\mathtt{cred\_expr}(sp_k))$. If the Operator attribute is equal to *concatenation*, for each $\mathtt{sp}_k \in ID_{sp} = m$ and for each cred_expr in subj_spec($\mathtt{sp}_k$), the system requires the signature of a subject s belonging to $\Phi(\mathtt{cred\_expr})$. Finally, if the Operator attribute is equal to $choice_\alpha$, where α is an id attribute of a signature policy in ID_{sp}, for each cred_expr in subj_spec($\mathtt{sp}_k$), where $k = \alpha$, the system requires the signature of a subject s such that his/her $\mathcal{X}$-Profile satisfies cred_expr. In all these cases, the selected subjects belong to $\Psi(subj\_spec(\bar{\mathtt{sp}}))$. If $|ID_{sp}| = 1$, that is $ID_{sp} = \{\mathtt{sp}_i\}$ for some policy identifier i, and $\exists ID'_{sp}$, with $|ID'_{sp}| > 1$ such that $ID_{sp} \cap ID'_{sp} \neq \emptyset$, the Operator attribute f is equal to *restrict*. It is evident that $V_d(\bar{\mathtt{sp}})$, with $\bar{\mathtt{sp}} \in f(ID_{sp})$ is equal to $\{el|\ (el, ID_{acp}, ID'_{sp}) \in M_d$ s.t.$ID'_{sp} = ID_{sp}\}$. Both the algorithms 2, and 3 require the signature of all $V_d(\mathtt{sp})$ or of a partition of its by a tuple of subjects $\bar{s} \in \Psi(subj\_spec(\bar{\mathtt{sp}})) = \Psi(subj\_spec(\mathtt{sp}_i))$. Otherwise, the Operator attribute f is set to *identity* in the *Conflict Resolution* phase. As above, the algorithms 2, and 3 require the signature of all $V_d(\bar{\mathtt{sp}}) = \{el|\ (el, ID_{acp}, ID'_{sp}) \in M_d$ s.t.$ID'_{sp} = ID_{sp}\}$, with $\bar{\mathtt{sp}} \in f(ID_{sp})$ or of a partition of its by a tuple of subjects $\bar{s} \in \Psi(subj\_spec(\bar{\mathtt{sp}})) = \Psi(subj\_spec(\mathtt{sp}_i))$. Analogously, if there are not conflicts among the signature policies in $\mathcal{SPB}$ on document d, the signature of document d is correct.

Now, we have to prove the third condition. Let s be a subject to which the encrypted and signed document d should be delivered. Suppose that s belongs to $\Phi(\mathtt{cred\_expr}(\mathtt{acp_i}))$, where acp_i is the identifier of an access control policy in $\mathcal{ACPB}$, both the Algorithms 2, and 3 allow the receiver subject s to verify the authenticity of $V_d(acp_i) \cap V_d(sp_j)$, for each signature policy $\mathtt{sp}_j$ in $\mathcal{SPB}$ or $\mathcal{SPB}'$. Indeed, according the strategy `encrypt-then-sign` s can verify the authenticity of all $V_d(sp_j)$ and, thus, the authenticity of $V_d(acp_i) \cap V_d(sp_j)$. By contrast, according to the sign-then-encrypt strategy, the receiving subject s can verify the authenticity of all the partitions of $V_d(acp_i) \cap V_d(sp_j)$. □

Authorization and Access Control in Adaptive Workflows

Dulce Domingos[1], António Rito-Silva[2], and Pedro Veiga[1]

[1] Informatics Department,
University of Lisbon, Faculty of Sciences
{dulce,pmv}@di.fc.ul.pt
[2] INESC-ID Software Engineering Group
Technical University of Lisbon
Rito.Silva@inesc-id.pt

Abstract. In recent years we have witnessed the development of adaptive workflow management systems. These systems offer an extended set of features to support both ad-hoc and evolutionary changes, while ensuring correctness of process definition and their running instances. Ad-hoc and evolutionary changes impose new access control requirements, which have been neglected by adaptive workflow research and are not met by existing models for traditional workflow management systems (WfMSs). In this paper, we extend the role-based access control model for adaptive workflows. This extension is done by defining authorizations for adaptive WfMSs and adaptive authorizations.

Keywords: Access control, adaptive workflow, adaptive authorizations

1 Introduction

Workflow management systems (WfMSs) are being used to re-engineer, streamline, automate, and track organizational processes [2]. There is a growing acceptance of workflow technology in numerous application domains such as telecommunications, software engineering, manufacturing, production, finance and banking, healthcare, shipping, and office automation [2]. However traditional WfMSs suffer from the lack of flexibility [1], which is a limitation for supporting more demanding applications, more dynamic environments and better human involvement in organizational activities [2].

Indeed, one of the most important requirements emerging from new application areas of WfMSs is the ability to handle adaptable and dynamic changes. As a consequence, in the last years we have witnessed the development of new types of WfMSs supporting adaptive workflows [2,3]. In this way, it is possible to change process definitions as well as process instances while they are being executed.

However, this flexibility raises new access control requirements, which are not met by traditional WfMSs access control models. In traditional WfMSs all instances are assumed to be alike, with no special handling of specific instances.

Moreover, users' participation is always passive; it is assumed that users only need to know about the activity they are personally involved in. Workflow participants get a work item from their worklist, handle it and check it in when it is finished. Indeed, it is assumed they do not need to know about the workflow as a whole. There are workflow administrators that deal with process definition. Consequently, access control requirements of traditional WfMSs focus mainly on task assignment.

Considering access control requirements, adaptive workflow literature agrees [2,14,29] that it must be possible to control at a very fine level of granularity who is allowed to perform which kind of changes and under which conditions. Therefore, the model should support the definition of who have the authority to introduce changes in the workflow and where. It should also distinguish the scopes of the adaptation, which can be done at instance level or at process definition level. Another important aspect stated by these authors points out that the workflow should be flexible enough to be able to adapt to changes in the organizational model, such as changes in persons or resources for a given role and changes in roles for given tasks.

In this paper, we adapt and extend the well-accepted role-based access control model [20] for adaptive WfMSs in two directions. One direction deals with the definition of authorizations considering the objects of adaptive WfMSs, such as: process definition, process instances, activities and activity instances; and operations on them like execute, change, read and grant. Moreover, subjects can be defined as dynamic roles using expressions that could refer to external resource information systems. This mechanism makes the model more adaptive to dynamic changes in organizations, by separating the role model from the organizational model. Implication rules are also defined, which are used to derive implicit authorizations from explicit ones.

The second direction extends the model in order to deal with adaptive authorizations. Our approach to support the evolution of authorizations is based on the following three mechanisms: (1) the definition of a set of operations that allows the change of authorizations, (2) the definition and enforcement of a correctness criterion, which guarantees the correctness of the access control aspect within a process definition, without neglecting side effects of change operations on other workflow aspects and (3) the definition and enforcement of a migration condition, which ensures the correctness of process instances in terms of access control.

The remainder of this paper is structured as follows: section 2 presents a brief overview of adaptive WfMSs. Section 3 surveys some related work in this area. In section 4 we discuss access control requirements of adaptive WfMSs and we illustrate them with an application scenario. Our access control model for adaptive WfMSs is presented in section 5 and section 6 describes the application of this access control model in the WorkSCo WfMSs. Finally, the last section presents some conclusions and provides a brief look for our future work.

2 Adaptive WfMS

This section reviews basic workflow concepts, according to the model and terminology defined by the Workflow Management Coalition [5], introduces adaptive workflows and classifies the two types of workflow adaptations.

A workflow management system is a system that supports the modelling and enactment of workflows. Workflow modelling creates a workflow process definition, which is a computerized representation of a business process and, normally, comprises a number of activities, whose chronological and logical order is given through the control flow. These activities can be atomic (basic work steps) or composite. Composite activities include one or more activities (designated by sub-activities) defining a hierarchy of activities. Workflow modelling requires a workflow meta-model that comprises a set of modelling concepts. A concrete workflow process definition is expressed in a workflow modelling language, which is a formal language offering constructs for the modelling concepts of the meta-model.

Workflow enactment involves the process definition interpretation by the WfMS, which creates and controls process instances, scheduling the various activities and invoking the appropriate human and application resources. Each process instance represents one individual enactment of the process definition, using its own process instance data. The next figure illustrates these concepts and their relationships, using the Unified Modelling Language (UML).

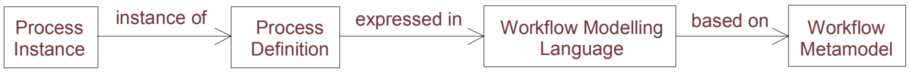


Fig. 1. Workflow modelling concepts

In recent years, we have witnessed the development of new types of WfMSs, which deal with adaptive workflows. Typically, adaptive WfMSs consider two types of changes:

- ad-hoc changes affect only one workflow instance or a selected group of instances. Changes occur on an individual or selective basis. The change is the result of an error, a rare event, or a special demand from the customer.
- evolutionary changes are of a structural nature: from a certain moment in time, the workflow definition changes for all new instances. Existing instances may also be influenced by an evolutionary change. The change can be the result of a new business strategy, reengineering efforts, or a permanent alteration of external conditions (e.g. a change of law).

Accordingly to Casati et al. [11] the problem of evolutionary changes has two facets:

- static evolution facet refers to the issue of modifying the process definition. In order to support the change of process definitions, adaptive WfMSs provide a complete and minimal set of primitives that allows changing the various aspects of the process definition. Typically, these primitives guarantee that the new version of the workflow definition is syntactically correct [11,15,14].
- dynamic evolution refers to the problem of managing running instances of a process, whose definition has been modified. The dynamic evolution facet requires mechanisms to support the migration of existing instances to new process definitions, without affecting their correctness. A process instance is correct if it is compliant with its definition, i.e., if it has been executed according to this new version up to the moment of the migration [11]. A simple but inefficient approach to determine whether a process instance can be migrated is to check its history for compatibility with the destination process definition [11,15]. A more efficient approach, presented by Kradolfer [15], defines a migration condition for each of the provided model change operations and determines whether migration is possible by considering the migration conditions of the operations by which the destination process definition is derived from its source. To deal with process instances that cannot be migrated to the new process definition and have to remain with the old one, some adaptive WfMSs support different versions of process definitions [15,12].

Note that ad-hoc changes are typically supported by adaptive WfMSs as particular cases of evolutionary changes.

As far as we know, research work in adaptive workflow focuses their approaches on specific workflow aspects (mainly the behavioural aspect) and, despite their consensual opinion about the need of access control, this aspect has not been considered yet.

3 Access Control in WfMSs

In recent years, role-based access control (RBAC) has gained a great acceptance. RBAC models have been developed as an alternative to traditional approaches to handling access control in information systems and their main purpose is to facilitate security administration and review [20]. The central notion of RBAC is that authorizations are associated with roles, and users are assigned to appropriate roles. Roles are created for the various job functions in an organization and users are assigned to roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Authorizations can be granted or revoked to roles as needed.

RBAC models have been widely applied to both commercial and research workflow systems [9,24]. Several extensions to the basic models have been presented in order to increase expressiveness and to propose appropriate tools and mechanisms to support it. These extensions include team-based authorizations [4,22,28], history constraints such as separation of duties [8,9,19], temporal and instance-based authorizations [9], mechanisms for inter-organizational workflows

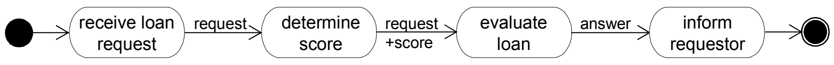


Fig. 2. Workflow access control objects

[13] and task-based authorizations [27,23], as well as workflow data access control [25,18].

However, to our knowledge, no access control model can be found in the literature that addresses access control requirements of adaptive workflows. In addition, only Casati et al. [10] propose in their work authorization rules that allow the definition of who is authorized to perform given types of changes. However, these authorization rules only apply to process instances and they are not assembled in an access control model.

4 Access Control Requirements in Adaptive Workflow

In this section we discuss some access control requirements of adaptive WfMSs in the context of a loan workflow application.

The top-level activities of our simplified loan workflow are illustrated in Figure 2. The activities *receive loan request* and *evaluate loan* are manual activities, while the others are automatic activities. Within this process definition, the activity *receive loan request* can be done by *clerks* and the activity *evaluate loan* can be done by *account managers* if the requestor is a bank client, or by *bank manager* otherwise.

4.1 Process Definition Authorizations

Adaptive WfMSs provide operations to handle process definitions such as: read, change and instantiate. Therefore, for security reasons, adaptive WfMSs should support the definition of authorizations in order to ensure that only authorized users can perform those operations on process definitions. Authorizations should also be defined at activity definition level.

4.2 Process Instance Authorizations

Maria's process instance of the loan process definition is illustrated in Figure 3 (grey activities have already been done). Adaptive WfMSs also provide operations to handle process instances. This instance can be executed, read and changed. As we have stated for process definition, adaptive WfMSs should also support the definition of authorizations in order to ensure that only authorized users can perform those operations on process instances and on activity instances.

4.3 Adaptive Authorizations

Within a WfMS, process definitions specify all the behaviour of their process instances. Therefore, to comprise the access control aspect, the process definition

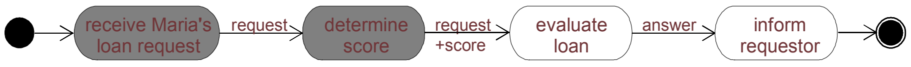


Fig. 3. An instance of the loan workflow application

has to specify, not only its activities, control flow and data flow, but also the authorizations that will be applied to the process instances. Consequently, we have to deal with adaptive authorizations, i.e., we have to support evolutionary and ah-hoc changes of authorizations.

To illustrate the need to support evolutionary changes of authorizations we use the simplified loan workflow illustrated in Figure 2, by considering that, as a result of the definition of more restricted rules for loan process definitions, authorizations defined for activity *evaluate loan* have to be changed: the activity *evaluate loan* can be done by *bank manager* if the score is less than 5 or by *loan manager of central bank* otherwise. Therefore, adaptive WfMSs should also ensure that only authorized users can change authorizations and that correctness of authorizations within the process definition is maintained (static evolution of authorizations facet).

Considering the process instance illustrated in Figure 3, if it needs to be migrated to the new process definition, then the adaptive WfMSs has to ensure the process instance correctness by evaluating if it is compliant with the new definition, taking into account its execution history (dynamic evolution of authorizations facet).

Finally, we point out that, ad-hoc changes are typically supported by adaptive WfMSs as particular cases of evolutionary changes, by creating and defining a specific process definition and migrating the instance to it.

4.4 Conclusions

From the loan workflow application, we conclude that an access control model for adaptive WfMSs needs to protect not only run-time objects (process and activity instances) but also build-time objects (process and activity definitions). The set of access modes that can be exercised on these objects needs to be identified, by taking into account the set of operations provided by adaptive WfMSs.

Finally, within an adaptive WfMSs, authorizations should also be adaptive. To deal with adaptive authorizations, our model should identify who is authorized to change authorizations. Additionally, changing authorizations needs special support in adaptive workflows in order to maintain the correctness of the process definition and its process instances.

5 An Access Control Model for Adaptive WfMSs

In this section we present an access control model for adaptive WfMS. This model is based on the RBAC model. This option is justified because RBAC provides flexibility and efficiency over traditional approaches, as stated before.

The RBAC model is extended and adapted by providing the definition of authorization states (authorizations) and then by describing mechanisms to support adaptive authorizations.

5.1 Authorizations

An authorization is represented as a quadruple (s, o, a, p) [26] and states that a subject s has an access mode a to perform an operation on an object o if the predicate p holds true (where p is optional). In this section we map the general notion of each component of an authorization to the adaptive workflow domain.

We also define rules for the derivation of implicit authorizations. Implication rules determine how implicit authorizations are derived from authorizations explicitly defined. The concept of implicit authorizations simplifies the definition of authorizations because it makes unnecessary to define and store all authorizations explicitly.

5.1.1 Subject. In RBAC models, subjects are defined as roles rather than individual users. This way, if the predicate p holds true, a user u has an access mode a on object o, if there exists the authorization (r, o, a, p), such that $u \in r$. Roles are created for the various activities of the process definition and users are assigned to roles based on their responsibilities and qualifications. However, since we are in the presence of a system that should be flexible enough in order to respond to changes efficiently, the workflow role model should be as independent as possible from the organizational model, in such a way that changes in the organizational model do not affect the workflow role model and vice-versa.

Therefore, subjects can be defined as dynamic roles. Roles and their hierarchy are defined locally to a process definition and roles are defined dynamically, i.e., instead of enumerating for each role the users that belong to it, the user to role assignment can be done as an expression on user attributes (designed by role definition rule). This expression is used, at runtime for user's session role resolution and activation, which can be used to query an external information system, such as, a directory service or a human resource application. Therefore, changes in the organizational model are reflected in the role model transparently and automatically.

Formally, let R be the set of the roles of a process definition, S the set of subjects of the organization model and $r_i dr$ the role definition expression for $r_i \in R$. Our role to user mapping is defined by the function:

- *role_resolution_function*$(r_i dr) \rightarrow 2^S$, this function maps each role r_i to a set of subjects by querying the information system that supports the organizational model.

Roles are hierarchically organized in the workflow role model. Let $r_i, r_j \in R$ be roles. We say that r_i dominates r_j in the hierarchy $(r_i > r_j)$ if r_i precedes r_j in the ordering.

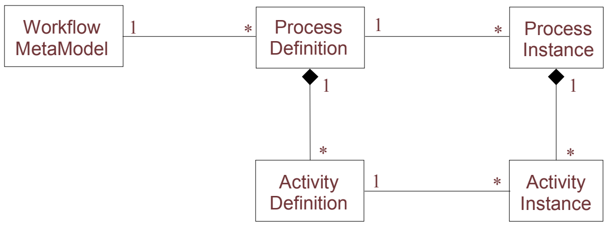


Fig. 4. Workflow access control objects

5.1.2 Object. The second component is object. Authorizations can be applied to each of the following objects: the workflow meta-model, process definitions, process instances, activity definitions and activity instances.

We explicitly define the relationships existing between these objects, as shown in Figure 4, since they imply implicit authorizations. The process definition includes activity definitions, which are hierarchically organized. We use AD to indicate the set of activity definitions with a relation $>_{AD}$ to indicate inclusion relationship, PD to indicate the set of process definitions and the relation $>_{PD}$ between PD and AD to indicate inclusion relationship. Let $ad_k, ad_j \in AD$ be activity definitions, we say that ad_k includes ad_j ($ad_k >_{AD} ad_j$) if ad_j is a sub-activity of ad_k and we say that $pd \in PD$ includes ad_k ($pd >_{PD} ad_k$) if ad_k is an activity of pd.

Similarly, process instances include activities instances, which are hierarchically organized. We use AI to indicate the set of activity instances with a relation $>_{AI}$ to indicate inclusion relationship, PI to indicate the set of process instances and the relation $>_{PI}$ between PI and AI to indicate inclusion relationship. Let $ai_k, ai_j \in AI$ be activity instances and $pi \in PI$ be a process instance, we say that ai_k includes ai_j ($ai_k >_{AI} ai_j$) if ai_j is a sub-activity of ai_k in $pi \in PI$ and we say that pi includes ai_k ($pi >_{PI} ai_k$) if ai_k is an instance activity of pi.

5.1.3 Access Modes. The set of access modes AM we consider in this model are: read, change, execute and grant. These access modes correspond to the read, write, execute, and owner access modes widely used in operating systems. The read access mode allow read-only accesses, as follows, considering each object of our model, (1) read access mode for the workflow meta-model authorizes users to list process definitions, (2) read access mode for process definitions and for activity definitions authorizes users to read the definition, and (3) read access mode for process instances and for activity instances authorize users to see their states.

The execute access mode is used to create new objects or to execute instances. Therefore, we have the following semantics for the execute access mode: (1) the execute access mode for the workflow meta-model authorizes users to create process definitions, (2) the execute access mode for process and activity definitions authorizes users to create (instantiate) instances of them (process instances and activity instances, respectively), and (3) the execute access mode for process instances and for atomic activity instances authorizes users to execute these instances (the execute access mode cannot be applied to composite activity instances because the unit of execution is an atomic activity instance).

Of special importance to adaptive workflow are the change access modes, which cover the following authorizations: (1) change access mode within the workflow meta-model authorizes users to remove process definitions, (2) the change access mode for process definitions authorizes users to define the process and to perform evolutionary changes, i.e., to change the process definition by, for example, adding or deleting activities (the same is applied for activity definitions), and (3) the change access mode for process instances authorizes users to perform ad-hoc changes, i.e., to change the process definition of this particular process instance (the same is applied for activity instances). Finally, the grant access mode is used to authorize users to change authorizations within the same object, i.e., the grant access mode for the workflow meta-model authorizes users to change the authorizations of the workflow meta-model, the grant access mode for a process definition authorizes users to change the authorizations of this process definition (the same is applied for activity definitions) and the grant access mode for a process instance authorizes users to change the authorizations of this process instance (the same is applied for activity instances).

In section 6, we exemplify how operations provided by an adaptive WfMSs can be mapped to this set of access modes.

5.1.4 Predicates. Predicates can be associated with authorizations to restrict their applicability. Predicates can use the following variables:

- Attributes of the system, such as time and location of access,
- Attributes of the user, for instance, his age,
- Attributes of the objects that differ according to the objects, such as instantiation time of process instances and activity instances.

Predicates used in authorizations defined for activity instances can also use the following variables:

- Workflow history, i.e., information about activity instances already executed, such as the username of the executor,
- Values of their input data.

Predicates have the following syntax:

```
<predicate> ::= <conjunctive-predicate>
                {OR <conjunctive-predicate>}
<conjunctive-predicate > ::= <compare-predicate>
                {AND <compare-predicate>}
<compare-predicate> ::= <left-value> <operator> <right-value>
<left-value> ::= <variable>
<right-value> ::= <constant> | <variable>
<operator> ::= '=' | '!= ' | '<' | '>' '<=' | '>='
```

5.1.5 Implication Rules. Implication rules are used to simplify the definition of authorizations. By using implication rules to derive implicit authorizations from explicit ones, the number of authorizations that need to be specified can decrease significantly.

Within our model we define the following implication rules:

Rule 1: the implication authorization rule on roles states that an authorization given to a role r propagates to all roles, which precede r in the role hierarchy (that is, to all roles r' such that $r' > r$).

Rule 2: the implication authorization rule on access modes states that change access mode implies read access mode.

Rule 3: the implication authorization rule on objects states that authorizations are propagated to the included activities, according to the inclusion relationships defined in section 5.1.2 ($>_{PD}, >_{AD}, >_{PI}, >_{AI}$). However, the authorization to execute a process instance should not imply authorizations to execute all its activity instances, therefore, the implication rule 3 does not apply to execute access mode defined for process instances.

Rule 4: the implication authorization rule on grant access mode states that change access mode implies grant access mode according to the inclusion relationships defined in section 5.1.2 ($>_{PD}, >_{AD}, >_{PI}, >_{AI}$). To define this implication rule, we take into account that, for example, the change access mode for process definition authorizes users to define the process by creating and defining activities or changing them. When these users are creating and defining activities, they also specify the authorizations of these activity definitions, exercising the grant access mode on them. The same can be applied to the change access mode for process instances, considering that the change access mode for process instances authorizes users to perform ad-hoc changes, i.e., to change the process definition of a particular instance.

5.2 Adaptive Authorizations

Within our access control model, process definitions specify not only the behavioural and informational aspects of the workflow, but also the access control aspect. Therefore, adaptive workflows that comprise the access control aspect cannot neglect mechanisms to support adaptive authorizations. Considering the access modes defined in section 5.1.3, evolutionary changes of authorizations can be done by exercising the change access mode of process definitions (or of activity definitions). Actually, the authorization to define or change the process definition also comprises the access control aspect. Similarly, ad-hoc changes of authorizations can be done by exercising the grant access mode of process instances as well as the change access mode of process instances (or of activity instances).

Subsequently, we describe how our model supports adaptive authorizations while maintaining both process definition and process instances correctness.

5.2.1 Static Evolution of Authorizations. As stated by Casati [11] the problem of evolutionary changes has two facets: the static evolution facet and the dynamic evolution facet. The static evolution of authorizations facet refers to the issue of modifying the authorizations defined in the process definition, while maintaining its syntactical correctness.

In order to support this facet, the model must provide primitives to allow change operations and must guarantee that the new version of the workflow definition is syntactically correct.

By this way, our model provides two types of primitive operations, which allow access control information to be changed, as follows: (1) operations to add and delete authorizations and (2) operations to manage roles (add and delete workflow roles, add and delete hierarchical relationships between roles, and add and delete role definition rules).

We define the notion of access control aspect model correctness based on model invariants that must be preserved (this is similar to the approach taken in [15], defined for the other workflow aspects). Invariants are conditions over the process definition that have to hold before and after change operations. Following is a list of invariants for the access control aspect:

- Authorization invariant: an authorization can only specify roles defined in the workflow role model, access modes defined in the model (section 5.1.3), objects of the system and valid predicates (see predicate invariant).
- Predicate invariant: requires that predicates can only refer to existent workflow relevant data and, in case of history predicates, defined workflow activities. This invariant can be violated by a change operation on other workflow aspect if that operation affects variables used by the predicate. Therefore, we have identified the operations that can have side effects on predicate validity, which are: the operation that removes an input data of an activity and the operation that removes an activity.
- Role invariant: role hierarchical relationships and role definition can only refer to roles defined in the workflow role model.

5.2.2 Dynamic Evolution of Authorizations. The dynamic evolution facet refers to the problem of managing running instances of a process, whose access control aspect definition has been modified. As stated before, the dynamic evolution facet requires mechanisms to support the migration of existing instances to another process definition, without affecting their correctness, by ensuring their compliance with the new process definition, i.e., by ensuring that the process instance history is compliant with the new process definition.

Considering the access control aspect, the process instance history comprises information about the user that has initiated the execution of the process instance and the users that have executed activity instances already performed. Therefore, a process instance is compliant with a process definition if (1) the user that has initiated the execution of the process instance is valid taking into account authorizations to execute process instances and if (2) the users that have performed already executed activity instances are valid taking into account authorizations to execute these activities.

Following the approach presented by Kradolfer [15], we have defined migration conditions for each of the access control change operations that can affect instances compliance, as follows:

Condition 1: delete an authorization specifying execute access mode for process instances: the process instance can be migrated to the new process definition if the subject that instantiated it is still authorized considering the remaining authorizations.

Condition 2: delete an authorization specifying execute access mode for a manual activity instance: the process instance can be migrated if (1) this activity instance has not been executed yet or, (2) it has already been executed and the subject that has executed it is still authorized considering the remaining authorizations.

6 Implementation in WorkSCo

In this section we overview the WorkSCo project[1] architecture and we describe how operations provided by this adaptive WfMS can be mapped to the set of access modes defined in section 5. This description is illustrated using the simplified loan workflow application example.

The Workflow with Separation of Concerns (WorkSCo) project is being developed based on the new workflow architecture presented in [6] designated by micro-workflow. The WorkSCo framework was developed using techniques specific to object systems and compositional software reuse. It targets software developers and provides the type of workflow functionality necessary in object-oriented applications. WorkSCo has a lightweight kernel that provides basic workflow functionalities and offers advanced workflow features as components that can be added to the kernel. Software developers select the features they need and add the corresponding components to the kernel through composition.

The evolution component uses a meta-model approach. It explicitly supports process definition versioning and workflow instances migration [15]. The idea behind process definition versioning is not to update process definitions in place, but version them. Those workflows that cannot be migrated can continue their execution under the old version.

Within this approach, evolutionary changes are supported by creating a new process definition version and performing changing operations on this new version. Ad-hoc changes to specific process instance are supporting by creating a variant of the process definition and migrating the process instance to this variant.

In the following we illustrate the application of the access model presented in the previous section to the WorkSCo evolution component, using the simplified workflow application example described in section 4 and shown in Figure 2. Initially, the WorkSCo WfMS is configured with the workflow meta-model authorizations, which states who can create, remove and list process definitions and who can change these authorizations.

[1] http://www.esw.inesc-id.pt/worksco

Considering a life cycle of process definitions and process instances, next we exemplify (1) the creation of the process definition, (2) the definition of a process definition, (3) an instantiation of a process, (4) an evolutionary change on the process definition and (5) a migration operation.

(1) Firstly our bank decides to define a new process for the loan workflow application. The WfMS's manager creates a new process definition, by exercising the execute access mode of the workflow meta-model. This process definition will be defined by the role *loan workflow designer*. Therefore, the process definition has to be created with the authorization (*loan workflow designer*, definition of *loan workflow*, *change*).

(2) Next the process is defined. A user playing the role *loan workflow designer* creates and defines its four activities, the control flow, the data flow and the access control aspect information. The access control aspect includes the roles of the process: *clerk, account manager, bank manager, loan manager of central bank* and *loan workflow designer* and its authorizations: (*clerk*, instances of *loan workflow*, *execute*). A user playing the role *loan workflow designer* can also define and change the activities that he will create without needing more authorizations. Indeed, implication rules defined in our model let these authorizations be omitted. This user also defines the following authorizations: (*clerk*, instances of *receive loan request*, *execute*, (*account manager*, instances of *evaluate loan*, *execute*, "requestor is a bank client") and (*bank manager*, instances of *evaluate loan*, *execute*, "requestor is not a bank client").

(3) Execution of process instances can be initiated by role *clerk* as stated by the authorization (*clerk*, instances of *loan workflow*, *execute*). Instances of activity receive loan request can be performed by role *clerk*, defined by authorization (*clerk*, instances of *receive loan request*, *execute*), while instances of activity *evaluate loan* can be performed by *account managers* if the requestor is a bank client or by *bank manager* otherwise, defined by authorizations: (*account manager*, instances of *evaluate loan*, *execute*, "requestor is a bank client") and (*bank manager*, instances of *evaluate loan*, *execute*, "requestor is not a bank client").

(4) As a result of the definition of more restricted rules for loan process definitions, authorizations defined for activity *evaluate loan* have to be changed. This change should be done by users playing role *loan manager of central bank*. Therefore, firstly, the authorization (*loan manager of central bank*, definition of *evaluate loan*, *change*) should be added. Users playing the role *loan workflow designer* can add this authorization exercising the authorization (*loan workflow designer*, definition of *loan workflow*, *change*), which, applying implication rule 4, derives the implicit authorization (*loan workflow designer*, definition of *evaluate loan*, *grant*). Then, a user playing role *loan manager of central bank* can remove the two authorizations defined within the activity *evaluate loan* and can add the two new authorizations: (*loan manager of central bank*, instances of *evaluate loan*, *execute*, "score greater or equal than 5") and (*bank manager*, instances of *evaluate loan*, *execute*, "score less than 5"). The operation that adds authorizations evaluates their invariants, ensuring correctness.

(5) Finally, a user playing role *loan manager of central bank* has to migrate running process instances to the new process definition, such as the Maria's

process instance shown in Figure 3. Therefore, two authorizations have to be defined: (*loan manager of central bank*, instances of *evaluate loan*, *change*) or (*loan manager of central bank*, instances of *evaluate loan*, *grant*), and (*loan manager of central bank*, definition of *evaluate loan*, *execute*). Users playing role *loan workflow designer* can add these authorizations by exercising the authorization (*loan workflow designer*, definition of *loan workflow*, *change*) and applying implication rules 3 and 4, respectively.

Both authorizations (*loan manager of central bank*, instances of *evaluate loan*, *change*) and (*loan manager of central bank*, instances of *evaluate loan*, *grant*) state that the role *loan manager of central bank* can change authorizations of *evaluate loan instances*, while the other authorization (*loan manager of central bank*, definition of *evaluate loan*, *execute*) states that the same role can instantiate the *evaluate loan* activity definition, i.e., can migrate instances to it.

Next, the migration operation has to ensure instance correctness by ensuring its compliance with the new process definition. Considering Maria's instance process, it is compliant with the new process definition because all the migration conditions hold. If, for example, the instance of the activity *evaluate loan* had already been executed by an *account manager*, it would not be compliant with the new process definition and the migration could not be performed.

7 Conclusions and Future Work

Recently, WfMSs have been extended in order to support both ad-hoc and evolutionary changes. These features raise new access control requirements that are not met by traditional workflow access control models.

In this paper, we identify access control requirements of adaptive WfMSs and present an access control model that meets these requirements. Our model extends the RBAC model by defining authorizations and mechanisms that support their evolution. Authorizations are defined taking into account the objects of adaptive WfMSs that need to be protected and the access modes that users can exercise on them. The presented model also supports dynamic roles and defines implication rules to derive implicit authorizations. To support the evolution of authorizations, this model defines an administrative policy and mechanisms to maintain the correctness of process definition as well as process instances.

The development of our model has been influenced by the following works:

- Adaptive WfMSs [11,15,14] – these works propose mechanisms to support adaptive workflows, by ensuring process definition correctness and process instances correctness. Our approach adapts these mechanisms in order to deal with access control aspects.
- Access control models for object oriented databases [26] – access control models for object oriented databases differentiate two types of protected objects: class and set of instances and define rules for the implication between authorizations along each of the three dimensions. These concepts have been adapted to the adaptive WfMSs domain.

- Dynamic roles [7] – within this work, dynamic role resolution is done when authorizations are evaluated. In our approach, dynamic role resolution is done when users establish sessions.

At present, we are also working on testing the applicability of our model in large workflow applications, such as health care workflow applications and emergency planning workflow applications for civil protection.

References

1. Agostini, A., Michelis, G.: A Light Workflow Management System Using Simple Process Models. In Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers, Vol.9, No.3-4 (2000), 335-363.
2. Sheth, A., Kochut, K.J.: Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems. Eds Dogaç, A., Kalinichenko, L., Ozsu, M.T., Sheth, A., NATO Advanced Study Institute, Istanbul, Turkey (1997).
3. Sheth, A.: From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Co-ordination and Collaboration. In Proceedings of the Workshop on Workflow Management in Scientific and Engineering Applications, Toulouse, France (1997).
4. Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K.: Flexible team-based access control using contexts. In Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies (2001), 21-27.
5. Hollingsworth, D.: The Workflow Reference Model. Document Number TC-00-1003. Issue 1.1. 19 January 1995.
6. Manolescu, D.: Micro-workflow: a workflow architecture supporting compositional object-oriented software development. PhD Thesis. University of Illinois at Urbana-Champaign (2001).
7. Cheng, E.C.: An Object-Oriented Organizational Model to Support Dynamic Role-based Access Control in Electronic Commerce Applications. In Proceedings of the 32nd Hawaii International Conference on System Sciences (1999).
8. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraint in workflow management systems. In ACM Transactions on Information and System Security, Vol.2, No.1 (1999) 65-104.
9. Casati, F., Castano, S., Fugini, M.: Managing Workflow Authorization Constraints through Active Database Technology. Information Systems Frontiers, Vol.3, No.3 (1999).
10. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.: Adaptive and Dynamic Service Composition in eFlow. In Proceedings of CAISE 2000, Stockholm, Sweden (2000).
11. Casati, F.: Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions. Ph.D.Thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano (1998).
12. Joeris, G., Herzog, O.: Managing Evolving Workflow Specifications with Schema Versioning and Migration Rules. TZI Technical Report 15, Center for Computing Technologies, University of Bremen (1999).
13. Kang, M.H., Park, J.S., Froscher, J.N.: Access Control Mechanisms for Inter-Organizational Workflow. In Proceedings of the 6th ACM Symposium on Access Control Models and Technologies SACMAT 2001, Chantilly, VA (2001), 66-74.

14. Reichert, M., Hensinger, C., Dadam, P.: Supporting Adaptive Workflows in Advanced Application Environments. In Proceedings of EDBT Workshop on Workflow Management Systems, Valencia, Spain, (1998).
15. Kradolfer, M.: A Workflow Metamodel Supporting Dynamic, Reuse-Based Model Evolution. Dissertation, Institut für Informatik, Universität Zürich (2000).
16. Miller, J.A., Fan, M., Wu, S., Arpinar, I.B., Sheth, A.P., Kochut, K.B.: Security for the METEOR Workflow Management System. Uga-cs-1sdis technical report, University of Georgia (1999).
17. Samarati, P., Vimercati, S.: Access Control: Policies, Models, and Mechanisms. In Focardi, R., Gorrieri, R. (eds): Foundations of Security Analysis and Design, LNCS 2172, Springer-Verlag (2001).
18. Holbein, R., Teufel, S., Morger, O., Bauknecht, K.: Need-to-Know Access Control for Medical Systems. In: Information Security - the Next Decade IFIP/SEC '97 (1997).
19. Botha, R.A., Eloff, J.H.P.: Separation of Duties for Access Control in Workflow Environments. In IBM Systems Journal Vol. 40, No. 3 (2001) 666-682.
20. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. IEEE Computer, Vol. 29, No. 2 (1996).
21. Siebert, R.: An Open Architecture for Adaptive Workflow Management Systems. In International Workshop on Issues and Applications in Database Technology (IADT'98), Berlin (1998).
22. Thomas, R. K.: Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments. In Proceedings of the second ACM Workshop on Role-based Access Control (1997) 13-19.
23. Thomas, R., Sandhu, R.: Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. In Proceedings of the IFIP WG11.3 Workshop on Database Security, Lake Tahoe, California (1997).
24. Kandala, S., Sandhu, R.: Secure Role-Based Workflow Models. In Database Security XV: Status and Prospects, Kluwer (2002).
25. Wu, S., Sheth, A., Miller, J., Luo, Z.: Authorization and Access Control of Application Data in Workflow Systems. In Journal of Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies (JIIS), Vol. 18, No. 1 (January 2002) 71-94.
26. Castano, S., Fugini, M., Martella, G., Samarati, P.: In Database Security. Addison-Wesley (1994).
27. Atluri, V., Huang, W.: An authorization Model for workflows. In Proceedings of the 5th European symposium on research in computer security, Rome, Italy (1996) 44-64.
28. Wang, W.: Team-and-Role-Based Organizational Context and Access Control for Cooperative Hypermedia Environments. In Proceedings of the tenth ACM Conference on Hypertext and Hypermedia (1999) 37-46.
29. Han, Y., Sheth, A., Bussler, C.: A Taxonomy of Adaptive Workflow Management. In Workshop on Adaptive Workflow Systems at the 1998 Conference on Computer-Supported Cooperative Work, Seattle, USA (1998).

Flexible Regulation of Distributed Coalitions[⋆]

Xuhui Ao[⋆⋆] and Naftaly H. Minsky

Department of Computer Science,
Rutgers University, Piscataway, NJ 08854, USA
{ao,minsky}@cs.rutgers.edu

Abstract. This paper considers a coalition C of enterprises $\{E_1,..., E_n\}$, which is to be governed by a *coalition policy* P_C, and where each member-enterprise E_i has its own internal policy P_i that regulates its participation in the coalition. The main question addressed in this paper is how can these three policies be brought to bear, on a single transaction—given that the two internal policies P_i and P_j may be formulated independently of each other, and may be considered confidential by the respective enterprises. We provide an answer to this question via a concept of policy-hierarchy, introduced into a regulatory mechanism called Law-Governed Interaction (LGI).

Keywords: Distributed coalition, Security policy, Decentralized regulatory mechanism, Law-Governed Interaction, Policy hierarchy, Policy interoperability.

1 Introduction

There is a growing tendency for organizations to form coalitions in order to collaborate—by sharing some of their resources, or by coordinating some of their activities. Such coalition are increasingly common in various domains, such as business-to-business (B2B) commerce, under names such as "virtual enterprises" of "supply chains;" and in *grid computing* [16,19,18]; and among educational institutions and governmental agencies. All such coalitions need to be regulated, in order to ensure conformance with the policy that is supposed to governs the coalition as a whole, and in order to protect the interest of member organizations. This need triggered a great deal of recent access-control research for coalitions [8,11,12,16,18,19], exhibiting several different views of the problem, and employing different techniques for its solution. Our own view of this problem, which is more general than most of the above, is based on the following definition of coalitions.

Definition 1. *A coalition C is a set $\{E_1,..., E_n\}$ of enterprises*[1]*, which interoperate under an ensemble of policies $[P_C, \{P_i\}]$, where P_C is the* coalition policy

[⋆] Work supported in part by NSF grant No. CCR-98-03698

[⋆⋆] Work supported in part by DIMACS under contract STC-91- 19999

[1] The term "enterprise" in this definition refers to educational and governmental institutions, as well as to commercial ones.

that governs the coalition as a whole, and P_i is the internal *policy of enterprise E_i, which governs its participation in the coalition.*

In Section 2 we will explain the need for a coalition to be governed by such an ensemble of policies, and illustrate the nature of such an ensemble. Here we just point out that this definition means that every interaction between an agent x_i of enterprise E_i and an agent x_j of E_j must comply with the internal policies P_i and P_j, as well as with the coalition policy P_C.

Two issues need to be addressed when establishing a regulatory framework for such a coalition: (a) the specification of the policy-ensemble that is to govern a coalition, and its evolution over time; and (b) the policy-enforcement mechanism. Regarding the latter issue, we adopt here our own access-control mechanism called Law-Governed Interaction (LGI)[13,14,15,2]. We will be mostly concerned, in this paper, with the former issue above. That is, with the structure and specification of the policy-ensemble of a coalition, requiring it to satisfy the following principle of *flexibility*:

Principle 1 (flexibility) *Each member-enterprise E_i should be able to formulate its internal policy P_i, and to change it at will, independently of the internal policies of other enterprises in the coalition, and without any knowledge of them.*

Such a flexibility is important for several reasons. First, it provides each enterprise with the *autonomy* to define its own policy at will, subject only to the pre-agreed coalition policy P_C. Second, the mutual independence of the internal policies of member enterprises simplifies their formation and their evolution. Finally, this principle allows an individual enterprise to keep its own policy *confidential*, since policies of other enterprises do not depend on it.

To appreciate some of the difficulties in satisfying this principle, consider the following question: how does one ensure that an interaction between an agent x_i of enterprise E_i and an agent x_j of E_j conforms to all three policies P_i, P_j and P_C—which govern it? A seemingly natural answer to this question is to *compose* policies P_i, P_C, and P_j into a single policy, to be enforced by a reference monitor mediating all coalition-relevant interactions between the two enterprises E_i, E_j. This approach has, indeed, been attempted by several researchers [9,5,12,6,20] concerned with the interoperability between agents subject to different policies. But such composition of policies has several serious drawbacks in the context of coalitions.

First, composition of policies could be computationally hard. According to [12], in particular, such composition is intractable for more than two policies, even for a relatively simple policy language. This is particularly serious problem for a coalition, which would require a quadratic number (in terms of its membership size) of compositions of triples of policies—a truly daunting prospect. Second, composition violates our principle of flexibility. This is because for an enterprise E_i to formulate, or change, its internal policy P_i, it will have to be aware of the internal policies of every other member-enterprise, say P_j—lest its new policy will prove to be inconsistent, and thus not composable, with P_j.

(We will later review other attempts to regulate coalitions, not based on policy composition.)

The Proposed Approach: We adopt a *top-down* approach to the specification of the policy ensemble of a coalition. That is, we propose to start by formulating the global coalition policy P_C, which specifies its constraints over interoperability between different coalition members. We then allow individual members to formulate their own internal policies, subject to P_C, but independent of each other.

This approach is supported by a hierarchical organization of policies, recently implemented into LGI. Such a hierarchy is formed via a *superior/subordinate* relation between policies, where a *subordinate* policy is constrained, by **construction**, to conform to its *superior* policy. Under such an organization of policies, the internal policies P_i could be defined as subordinate to P_C. A pair of internal policies P_i and P_j, thus defined, are consistent with each other *by definition*, although they have been formed with no knowledge of each other, because both are guaranteed to conform to the same global policy P_C. As we shall see, the enforcement of such a hierarchical policy-ensemble can be carried out in a completely decentralized manner, without ever having to compose two independently defined policies, such as P_i and P_j, into a single policy.

The remainder of the paper is organized as follows: Section 2 motivates the proposed approaches to the governance of coalitions, illustrating it via a fairly realistic example; this section also discusses related work. Section 3, provides an overview of the concept of law-governed interaction (LGI). Section 4 shows how policies under LGI can be organized into hierarchies, and Section 5 presents a formalization of the example policy-ensemble introduced in Section 2, via the LGI hierarchy model. We conclude in Section 6.

2 On the Nature of the Proposed Regulatory Framework for Coalitions

We have just proposed a regulatory framework for coalitions, under which a coalition C of enterprises $\{E_1,..., E_n\}$ is to be governed by an ensemble of policies $[P_C, \{P_i\}]$, which satisfies the *flexibility principle.* In this section we will attempt: (1) to motivate this coalition model and to illustrate it by means of a detailed example; and (2) to compare our model with other approaches to the governance of coalitions. But we first offer some general remarks about the roles that the various policies in this ensemble are expected to play, and about what they are expected to regulate.

First, we expect the coalition policy P_C to represent prior agreement, or contract, between its member enterprises—as well as possible governmental regulation of such coalitions. For example, P_C might specify such things as the coalition membership, and the required interaction protocol between agents of different member enterprises. It might also provide certain coalition officers with a limited ability to regulate dynamically the interaction between member enterprises. We will not be concerned in this paper with how such a policy is established. This issue has been addressed elsewhere, such as in [8], where a negotiation process between member enterprises to establish a compromised P_C has been presented.

Second, we expect the internal policy P_i of an enterprise E_i to be concerned with internal matters of this enterprise, as they are related to coalition activities. Thus, P_i might specify such things as: (a) which of its *agents* (i.e., people or system-components) are allowed to participate in the coalition activity, either as servers for fellow coalition members, or as clients for services provided by fellow members; (b) the amount of usage a given agent of this enterprise is allowed to make, of services provided by fellow member enterprises, and the amount of services a given server is allowed to provide to the coalition; and (c) the required behavior of agents of this enterprise, when serving requests from other coalition members, or when making such requests—for example, P_i may require certain coalition activities of its agents to be monitored.

2.1 An Example of a Policy-Ensemble

Consider a set $\{E_1,..., E_n\}$ of enterprises that form a coalition C in order to collaborate by using some of each other's services—as is increasingly common in grid computing. Suppose that the coalition as a whole has a distinguished agent D_C called its *director*, and that each member enterprise E_i has its own director D_i. The function of these directors is, in part, to serve as *certification authorities* (CAs), as follows: D_C would certify the directors of the member enterprises, which, in turn, would certify the names and roles of agents within their respective enterprises. These and other roles of the directors, along with other aspects of this coalition are governed by the global coalition policy P_C, and by the internal policies of individual enterprises defined as *subordinate* to P_C, forming a two-level hierarchy depicted in Figure 1(a). These policies would now be described informally, and discussed. They would be formalized as "laws" under LGI in Section 5.

The Coalition Policy P_C: This policy deals with two issues, as described informally below:

1. Director's control over the usage of enterprise-resources:
   (a) The total amount of services offered by an enterprise E_i to other coalition members is determined by director D_i, by offering to the coalition director D_C a budget *B_i for using its services. This budget is expressed in terms of what we call E_i-currency, which can be created (minted, in effect) only by the director D_i, as described above.*
   (b) The coalition director D_C can distribute E_i-currencies he got from D_i among the directors of other member-enterprises.
   (c) E_i-currency can be used to pay for services provided by agents of E_i. Such currency can also be moved from one agent to another within an enterprise, subject to the internal policy of that enterprise—but it cannot be forged.

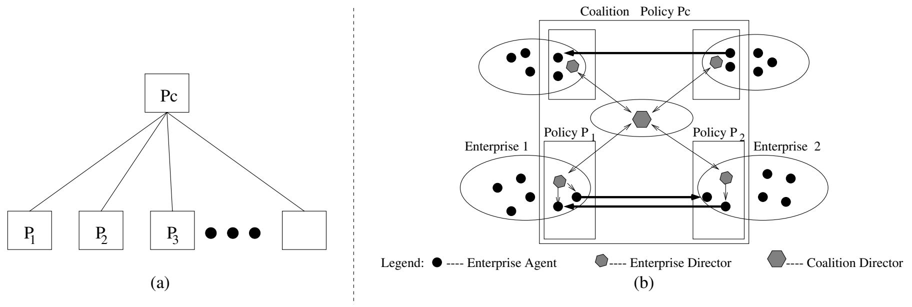


Fig. 1. (a) The policy ensemble governing coalition C; (b) The distributed coalition composed of multiple enterprises and their policies

2. Regulation over service requests by agents at enterprise E_i to agents in E_j:
   (a) each such request must carry the name of the sender, as authenticated by director D_i, and must be sent to an agent authenticated as a `server` *by the director D_j.*
   (b) Each such request must carry a payment in E_j-currency, which would be moved by this request from the client to the server. However, if the request is rejected, for whatever reason[2]*, then the payment will be refunded to the client.*

Thus, policy P_C mandates various global aspects of the coalition, including: (a) the authority, and implied responsibilities, of the various directors; (b) the requirement that service budgets minted by directors should be treated as currency, which can be *moved* from one agent to another, in particular as part of a service request, but cannot be copied or forged; and (c) the requirement that service requests must carry the proper authentication of the clients and be accepted by authenticated servers.

But policy P_C does not address many aspects of the coalition, leaving them for the internal policies of member enterprises to specify. These include, for an enterprise E_i, say, such things as: (a) how is E_j-currency, provided to D_i by the coalition director, to be distributed among the various agents of E_i; (b) how much should servers in E_i charge for their services, and what should they do with payments received; and (c) various conditions that must be satisfied, for agents of E_i to use services of other enterprises, or to provide services to others—such as time constraints and auditing requirements. The following are two example of such internal policies, for member enterprises E_1 and E_2.

[2] The following are among the reasons for a service request to be rejected: (1) the receiver is not an authenticated server; (2) the type of currency used for payment doesn't match the server's enterprise; and (3) the request is not allowed to be served, according to the server enterprise's internal policy.

Policy P_1 of Enterprise E_1:

1. *The director D_1 of this enterprise (E_1) can move service currencies obtained from D_C to any other authenticated agent in its enterprise.*
2. *Copies of all service requests sent by agents of this enterprise, and of all requests received by its servers, must be sent to a designated audit-trail server.*

Policy P_2 of Enterprise E_2:

1. *Any agent in this enterprise (E_2) can give service currencies it has, to any other agents that has been duly authenticated by the director D_2. (That is, currencies can move freely from one agents to another, unlike under P_1, where currencies can only be granted by the director to regular agents.) Also, every transfer of currency is to be audited by a distinguished audit-trail server.*
2. *Agents of this enterprise are allowed to provide services to other coalition members, or to request such services, only after normal working hours (namely, from 6:00PM to 8:00AM).*

Note that it is an essential aspect of the policy-ensemble of this coalition that it is hierarchical, as depicted in Figure 1(a). As we shall see in Section 4, this means that all internal policies P_i *conform* to P_C, so that none of them would be able to violate any of the provisions of P_C. For example, it would not be possible to write a internal policy for an enterprise E_j, say, that allows agents of E_j to forge E_i-currency. This provides an assurance for the director D_i of enterprise E_i that the total amount of E_i-currency circulating among the agents of the various coalition members—and, thus, the total amount of services its agents may be asked to perform—does not exceed the service budget B_i he originally sent to the coalition director D_C.

Therefore, the various member enterprises have the flexibility of being able to write their policies independently of the policies of other members, and they can change their policies at will. And yet, each enterprise can be confident that its interlocutors conform to the same common coalition-policy P_C.

Figure 1(b) provides an overview of the governance of coalition C by its hierarchical policy-ensemble. The outer box in this figure represents policy P_C, which governs the internal policies P_i— represented by boxes nested in it—as well as the coalition director, which operates directly under P_C. The directors of member-enterprises, represented by shaded shapes, along with other agents of such enterprises, are governed directly by their own policies, but indirectly by P_C. This figure attempts to illustrate two notable aspects of these policies: (1) the coalition director D_C is only involved in the distribution of service currency among the directors of member enterprises. He (she, or it) is not involved in the actual sharing of services (represented by the thick arrows) which is done directly between the agents of different member enterprises. (2) the internal policy P_i does not, necessarily, govern the entire enterprise E_i, but only the involvement of E_i in the working of coalition C.

2.2 Other Approaches to the Governance of Coalitions

Our view that coalitions need to be governed by an ensemble of policies $[P_C, \{P_i\}]$ is not the common view of the governance of coalitions—not to speak of our hierarchical structure of such an ensemble, which is quite unique. In this section we review several other approaches to this issue—starting with projects that employ only internal policies, and ending with those that employ both internal and global policies, but in different manners from ours. All the projects to be reviewed below also differ from ours in their enforcement mechanisms, and in their expressive power—which is generally based on stateless RBAC models, and are much weaker than ours. We will comment about these issues, wherever appropriate.

(I) Shands et al. [18], employ only internal P_i policies, with no global coalition policy. Moreover, the only control that these policies may have over the ability of agents to issue service requests, is by assigning them to roles. The server enforces its own policy, as well as that of its client, for each service request. In the approach of Thomson et al. [19], enterprises define their policies regarding a resource in which they have a stake, by issuing digitally-signed certificates. Only when all the stakeholders' access requirements are met can the access be performed. Similarly to [18], all policies are enforced by the server asked to perform a service.

There are several problems with this kind of *server-centric* policy enforcement approach in the coalition context: First, the client enterprise needs to trust the distributed heterogeneous servers to implement and enforce correctly not only their internal enterprise policy, but also that of the client enterprise. Furthermore, the server needs to know the internal policy of the client enterprise, which violates the *confidentiality* requirement of our principle of flexibility. Finally, it is virtually impossible for a server to enforce a *stateful* client's policy, such as our policies regarding the movement of currency on the client side.

(II) Several projects take a bottom-up approach. They start with the internal P_i policies, but attempt, in various ways, to form a global coalition policy from them. These include the already mentioned work of McDaniel et al. [12], which attempted to compose the internal policies automatically into a single common policy, finding this task to be computationally hard. Another approach to this problem is that of Gligor et al. [8], which tries to establish the common access policy via negotiation among the coalition members.

(III) Finally, we are aware of two projects that, like us, view a coalition as being governed by a global coalition policy as well as by the internal policies of individual coalition members. One of these is the project of Pearlman et al. [16], where servers are expected to take into account certificates issued by a centralized enforcer, driven by the coalition policy. The other such project, which is philosophically closest to our own, is that of Belokosztolszki and Moody [4]. They introduce the concept of *meta policy*, which is expected to be conformed to by all internal policies of member enterprises, and is, thus, analogous to our coalition policy P_C. However, this work is based on the RBAC model for access control, which is much weaker than LGI. In particular, a statful policy, which is sensitive to "service currency," such as our example policy, cannot be represented via RBAC. It is not clear to us how their construction could be extended to such policies.

3 Law-Governed Interaction (LGI) – An Overview

LGI is a message-exchange mechanism that allows an *open group* of distributed agents to engage in a mode of interaction *governed* by an explicitly specified policy, called the *law* of the group. The messages thus exchanged under a given law $\mathcal{L}$ are called $\mathcal{L}$-messages, and the group of agents interacting via $\mathcal{L}$-messages is called a *community* $\mathcal{C}$, or, more specifically, an $\mathcal{L}$-community $\mathcal{C}_{\mathcal{L}}$. This mechanism has been originally proposed by one of the authors (Minsky) in 1991 [13], and then implemented, as described in [15,1].

By the phrase "open group" we mean (a) that the membership of this group (or, community) can change dynamically, and can be very large; and (b) that the members of a given community can be heterogeneous. here All the members are treated as black boxes by LGI, which deals only with the interaction between them via $\mathcal{L}$-messages, ensuring conformance to the law of the community.

We now give a brief discussion of the concept of law, emphasizing its local nature, a description of the decentralized LGI mechanism for law enforcement, and its treatment of digital certificates. We do not discuss here several important aspects of LGI, including its concepts of *obligations* and of *exceptions*, the expressive power of LGI, and its efficiency. For these issues, and for implementation details, the reader is referred to [15,1].

3.1 On the Nature of LGI Laws, and Their Decentralized Enforcement

The function of an LGI law $\mathcal{L}$ is to regulate the exchange of $\mathcal{L}$-messages between members of a community $\mathcal{C}_{\mathcal{L}}$. Such regulation may involve (a) restriction of the kind of messages that can be exchanged between various members of $\mathcal{C}_{\mathcal{L}}$, which is the traditional function of access-control policies; (b) transformation of certain messages, possibly rerouting them to different destinations; and (c) causing certain messages to be emitted spontaneously, under specified circumstances, for monitoring purposes, say.

A crucial feature of LGI is that its laws can be *stateful.* That is, a law $\mathcal{L}$ can be sensitive to the dynamically changing *state* of the interaction among members of $\mathcal{C}_{\mathcal{L}}$. Where by "state" we mean some function of the history of this interaction, called the *control-state* (CS) of the community. The dependency of this control-state on the history of interaction is defined by the law $\mathcal{L}$ itself. For example, under law $\mathcal{P}_C$ to be introduced in section 5, as a formalization of our example P_C policy, the term `budget(`B_i`,`E_i`)` in the control-state of an agent denotes the amount of budget this agent gets from its director or other agents.

But the most salient and unconventional aspects of LGI laws are their strictly local formulation, and the decentralized nature of their enforcement. To motivate these aspects of LGI we start with an outline of a centralized treatment of interaction-laws in distributed systems. Finding this treatment unscalable, we will show how it can be decentralized.

On a Centralized Enforcement of Interaction Laws: Suppose that the exchange of $\mathcal{L}$-messages between the members of a given community $\mathcal{C}_{\mathcal{L}}$ is mediated by a reference monitor $\mathcal{T}$, which is trusted by all of them. Let $\mathcal{T}$ consist of the following three part: (a) the law $\mathcal{L}$ of this community, written in a given language for writing laws; (b) a generic *law enforcer* $\mathcal{E}$, built to interpret any well formed law written in the given law-language, and to carry out its rulings; and (c) the control-state ($\mathcal{CS}$) of community $\mathcal{C}_{\mathcal{L}}$ (see Figure 2(a)).

The structure of the control-state, and its effect on the exchange of messages between members of $\mathcal{C}_{\mathcal{L}}$ are both determined by law $\mathcal{L}$. For example, under law $\mathcal{P}_{\mathcal{C}}$, a message `giveCurrency(...)` will cause the specific service currency to be reduced from the CS of the sender and added to that of the receiver.

This straightforward mechanism provides for very expressive laws. The central reference monitor $\mathcal{T}$ has access to the entire history of interaction within the community in question. And a law can be written to maintain any function of this history as the control-state of the community, which may have any desired effect on the interaction between community members. Unfortunately, this mechanism is inherently unscalable, as it can become a bottleneck, when serving a large community, and a dangerous single point of failure.

Moreover, when dealing with stateful policies, these drawbacks of centralization cannot be easily alleviated by replicating the reference monitor $\mathcal{T}$, as it is done in the Tivoli system [10], for example. The problem, in a nutshell, is that if there are several replicas of $\mathcal{T}$, then any change in $\mathcal{CS}$, like the reduction of the service currency from a sender x of message `giveCurrency(...)`, in the example above, would have to be carried out *synchronously* at all the replicas; otherwise x may be able to send more service currency to other agents than what it actually has, via different replicas. Such maintenance of consistency between replicas is very time consuming, and is quite unscalable with respect to the number of replicas of $\mathcal{T}$.

Fortunately, as we shall see below, law enforcement can be genuinely *decentralized*, and carried out by a distributed set $\{\mathcal{T}_x \mid x \in \mathcal{C}\}$ of, what we call, *controllers*, one for each members of community $\mathcal{C}$ (see Figure 2(b)). Unlike the central reference monitor $\mathcal{T}$ above, which carries the CS of the entire community, controller $\mathcal{T}_x$ carries only the *local control-state* $\mathcal{CS}_x$ of x—where $\mathcal{CS}_x$ is some function, defined by law $\mathcal{L}$, of the history of communication between x and the rest of the $\mathcal{L}$-community. In other words, changes of $\mathcal{CS}_x$ are strictly local, not having to be correlated with the control-states of other members of the $\mathcal{L}$-community.

The Local Nature of LGI Laws: An LGI law is defined over a certain types of events occurring at members of a community $\mathcal{C}$ subject to it, mandating the effect that any such event should have. Such a mandate is called the *ruling* of the law for the given event. The events subject to laws, called *regulated events*,

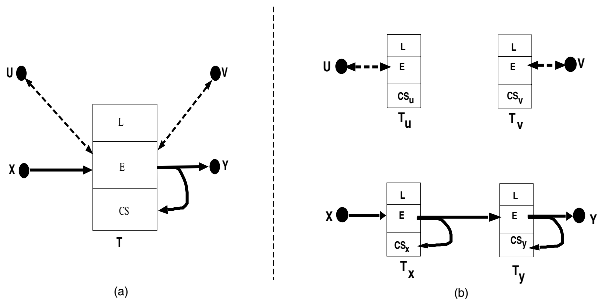


Fig. 2. Law Enforcement: (a) centralized version; (b) decentralized law enforcement under LGI

include (among others): the *sending* and the *arrival* of an $\mathcal{L}$-message; and the *submission of a digital certificate.* The operations that can be included in the ruling for a given regulated event, called *primitive operations*, are all local with respect to the agent in which the event occurred (called, the "home agent"). They include, operations on the control-state of the home agent and operations on messages, such as `forward` and `deliver`. To summarize, an LGI law must satisfy the following locality properties:

(a) a law can regulate explicitly only *local events* at individual agents; (b) the ruling for an event `e` at agent `x` can depend only on `e` itself, and on the *local control-state* $\mathcal{CS}_x$; and (c) the ruling for an event that occurs at `x` can mandate only *local operations* to be carried out at `x`.

Decentralization of Law-Enforcement: As has been pointed out, we replace the central reference monitor $\mathcal{T}$ with a distributed set $\{\mathcal{T}_x \mid x \in \mathcal{C}\}$ of controllers, one for each members of community $\mathcal{C}$. Structurally, all these controllers are generic, with the same law-enforcer $\mathcal{E}$, and all must be trusted to interpret correctly any law they might operate under. When serving members of community $\mathcal{C}_{\mathcal{L}}$, however, they all carry the *same law* $\mathcal{L}$. And each controller $\mathcal{T}_x$ associated with an agent x of this community carries only the *local control-state* $\mathcal{CS}_x$ of x (see Figure 2(b)).

Due to the local nature of LGI laws, each controller $\mathcal{T}_x$ can handle events that occur at its client x strictly locally, with no explicit dependency on anything that might be happening with other members in the community. It should also be pointed out that controller $\mathcal{T}_x$ handles the events at x strictly sequentially, in the order of their occurrence, and atomically. This, and the locality of laws, greatly simplifies the structure of the controllers, making them easier to use as our *trusted computing base* (TCB).

Finally, we point out that the LGI model is silent on the placement of controllers *vis-a-vis* the agents they serve, and it allows for the sharing of a single controller by several agents. This provides us with welcome flexibilities, which can be used to minimize the overhead of LGI under various conditions.

On the Structure and Formulation of Laws: Broadly speaking, the law of a community is a function that returns a *ruling* for any possible regulated event that might occur at any one of its members. The ruling returned by the law is a possibly empty sequence of primitive operations, which is to be carried out locally at the location of the event from which the ruling was derived (called the *home* of the event). (By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.)

More formally, an LGI law $\mathcal{L}$ is a function (called the *ruling* function) of the following form:

$$r = L(e, cs), e \in E, cs \in CS, r \in R \qquad (1)$$

where E is the set of regulated events, CS is the set of control states, and R is the set of all possible sequences of operations that constitute the ruling of the law.

Concretely, such a function can be expressed in many languages. Our middleware currently provides two languages for writing laws: Java, and a somewhat restricted version of Prolog [7]. We employ Prolog in this paper. In this case, the law is defined by means of a Prolog-like program `L` which, when presented with a goal `e`, representing a regulated-event at a given agent `x`, is evaluated in the context of the control-state of this agent `cs`, producing the list of primitive-operations `r` representing the ruling of the law for this event. In addition to the standard types of Prolog goals, the body of a rule may contain two distinguished types of goals that have special roles to play in the interpretation of the law. These are the *sensor-goals* (in the form `t@CS`), which allow the law to "sense" the control-state of the home agent, and the *do-goals* (in the form `do(p)`) that contribute to the ruling of the law.

On the Basis for Trust between Members of a Community: For a member of an $\mathcal{L}$-community to trust its interlocutors to observe the same law, one needs the following assurances: (a) that the exchange of $\mathcal{L}$-messages is mediated by correctly implemented controllers; (b) that these controllers are interpreting the *same law* $\mathcal{L}$; and (c) that $\mathcal{L}$-messages are securely transmitted over the network. If these conditions are satisfied, then it follows that if `x` receives an $\mathcal{L}$-message from some `y`, this message must have been sent as an $\mathcal{L}$-message; in other words, that $\mathcal{L}$-messages cannot be forged.

Broadly speaking, these assurances are provided as follows: Controllers used for mediating the exchange of $\mathcal{L}$-messages authenticate themselves to each other via certificates signed by a certification authority specified by the value of the `ca` attribute in the `law` clause of law $\mathcal{L}$ (see, for example, Figure 4, in the case of law P_C). Note that different laws may, thus, require different certification levels for the controllers used for its enforcement. Messages sent across the network are digitally signed by the sending controller, and the signature is verified by the receiving controller. To ensure that a message forwarded by a controller $\mathcal{T}_x$ under law $\mathcal{L}$ would be handled by another controller $\mathcal{T}_y$ operating under the *same* law, $\mathcal{T}_x$ appends a one-way hash [17] `H` of law $\mathcal{L}$ to the message it forwards to $\mathcal{T}_y$. $\mathcal{T}_y$ would accept this as a valid $\mathcal{L}$-message under $\mathcal{L}$ if and only if `H` is identical to the hash of its own law.

The Deployment of LGI. All one needs for the deployment of LGI is the availability of a set of trustworthy controllers, and a way for a prospective client to locate an available controller. This can be accomplished via one or more *controller-services*, each of which maintains a set of controllers, and one or more certification authorities that certifies the correctness of controllers. For an agent x to engage in LGI communication under a law $\mathcal{L}$, it needs to locate a controller, via a controller-service, and supply this controller with the law $\mathcal{L}$ it wants to employ. Once x is operating under law $\mathcal{L}$ it may need to distinguish itself as playing a certain role, or having a certain unique name, which would provide it with some distinct privileges under law $\mathcal{L}$. One can do this by presenting certain digital certificates to the controller. For the details of how to deal with the certificate, including its expiration and revocation in LGI, the reader is referred to [2].

4 The LGI Law-Hierarchy

We will introduce here the concept of law-hierarchy that formalizes the policy-hierarchy described in Section 2. Each such hierarchy, or tree, of laws $t(\mathcal{L}_0)$, is rooted in some law $\mathcal{L}_0$. And each law in $t(\mathcal{L}_0)$ is said to be (transitively) *subordinate* to its parent, and (transitively) *superior* to its descendents. (As a concrete example of such a hierarchy we will use the two-level tree $t(\mathcal{P}_C)$ which is the formalization under LGI of the policy hierarchy depicted in Figure 1(a).)

Generally speaking, each law $\mathcal{L}'$ in a hierarchy $t(\mathcal{L}_0)$ is created by *refining* a law $\mathcal{L}$, the parent of $\mathcal{L}'$, via a *delta* $\overline{\mathcal{L}'}$, where a delta is a collection of rules defined as a refinement of an existing law (we will, generally denote a delta via an overline above the name of the law they create). The root $\mathcal{L}_0$ of a hierarchy is a normal LGI law, except that it is created to be *open* for refinements, in a sense to be explained below. This process of refinement is defined in a manner that guarantees that every law in a hierarchy *conforms* to its superior law—as we shall see later.

For example, in coalition C, the local law $\mathcal{P}_1$ of enterprise E_1 would be created by refining the coalition law $\mathcal{P}_C$ by means of delta $\overline{\mathcal{P}_1}$, defined in Figure 5. This is how the hierarchy $t(\mathcal{P}_C)$—the ensemble of policies governing coalition C—is formed.

We will introduce here an abstract—language-independent—model for law-refinement. This is followed by a description of a concrete realization of this model, for laws written in Prolog. Finally, we will discuss the basis for trust between members of different communities under our law-hierarchy model.

4.1 An Abstract Model

Recall that an LGI law $\mathcal{L}$ as described in Section 3 is essentially a *ruling* function defined in Equation 1, and illustrated in Figure 3(a). We will now *open up* this law, by: (a) allowing it to consult a collection of rules designed to refine it—called a *delta*; and (b) by taking the advice returned by this delta into account, when computing its ruling. This is done by replacing the L function defined in Equation 1 with a pair of functions: a *consultation function* L_C, defined in Equation 2, which computes a *pseudo event pe* to be presented to the refining delta for evaluation; and a *ruling function* L_R, defined in Equation 3, which takes the *proposed ruling pr* returned by the refining delta, and computes the final ruling of the law. The open law is illustrated in Figure 3(b), and this law with a specific delta is depicted in Figure 3(c).

$$pe = L_C(e, cs) \tag{2}$$

$$r = L_R(e, cs, pr) \tag{3}$$

These two functions are evaluated in succession, as follows: when an event e is submitted to law $\mathcal{L}$ for evaluation, the function L_C is evaluated first, producing a pseudo event pe, which is passed to delta $\overline{\mathcal{L}'}$ for *consultation.* The delta operates just like the standard LGI law, producing its ruling—called, in this context, a "proposed ruling" pr—which is then fed as an input to the ruling function L_R. The result r produced by L_R is, finally the ruling of law $\mathcal{L}$.

Suppose now that unlike the open law $\mathcal{L}$, its delta $\overline{\mathcal{L}'}$ is closed; that is, it is a single function, defined by Equation 4, which computes its proposed ruling pr without consulting anything.

$$pr = \overline{L'}(pe, cs), pe \in E, cs \in CS, pr \in R \tag{4}$$

Then the refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$, produces a regular LGI law $\mathcal{L}'$—which is not open to further refinements—as illustrated by Figure 3(c).

Now, note that the ruling function of this *closed* law $\mathcal{L}'$ is a composition of functions 2, 3 and 4 above, which has the following form:

$$r = L'(e, cs) = L_R(e, cs, \overline{L'}(L_C(e, cs), cs)) \tag{5}$$

Law $\mathcal{L}'$ defined by this ruling function is called a *subordinate* to law $\mathcal{L}$.

It should be pointed out that we limited ourselves here to closed deltas, which do not consult anything when computing their rulings. Such deltas can produce hierarchies of depth two, at most—which is all we need in this paper. However, this model can be extended to open deltas that consult other deltas at a lower level, thus producing a cascade of refinements, and a hierarchy of arbitrary depth. Such a model is outlined in [3], but is beyond the scope of this paper.

Finally, we can now explain the sense in which a law $\mathcal{L}'$, which is subordinate to $\mathcal{L}$ in a hierarchy, is said to *conform* to its superior law $\mathcal{L}$: Law $\mathcal{L}'$ is created by refining $\mathcal{L}$ via a delta $\overline{\mathcal{L}'}$. As is evident from both Figure 3 and Equation 5, this is done by having the ruling of the delta submitted as an input to the ruling function of $\mathcal{L}$, which finally produces the ruling. Thus, the final decision about the ruling of law $\mathcal{L}'$ is made by its superior law $\mathcal{L}$, leaving to its deltas only an advisory role.

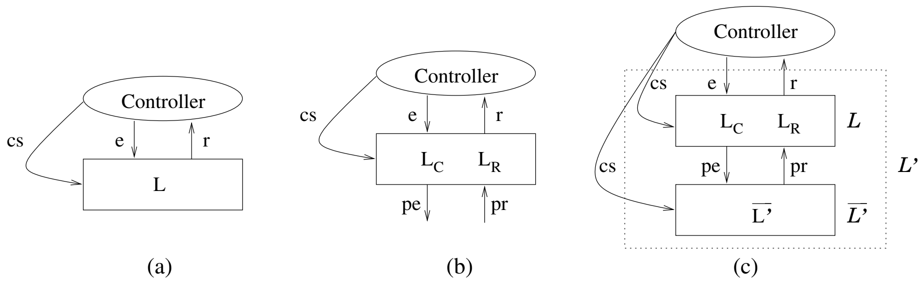


Fig. 3. (a) An closed LGI law $\mathcal{L}$, in context; (b) An open-up law $\mathcal{L}$; (c) Refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$, producing law $\mathcal{L}'$

4.2 A Concrete Realization of Law Hierarchy, for Laws Written in Prolog

We now describe how our current Prolog-based language for writing laws has been extended to implement the abstract model of law hierarchies. We start with the manner that a law can consult its deltas, then we describe the structure of the deltas themselves, and we conclude with the way that a law can decide on the disposition of ruling proposals returned to it by its deltas.

Consulting Deltas: Suppose that we are operating under law $\mathcal{L}'$, which is a refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$. An arbitrary consultation function L_C (see Equation 2) can be defined into a law $\mathcal{L}$, by inserting a clause of the form `delegate(g)`, anywhere in the body of any rule of $\mathcal{L}$, where `g` is an arbitrary Prolog term. The presence of a `delegate(g)` clause serves to invite refining deltas to propose operations to be added to the ruling being computed.

To see the effect of the `delegate(g)` clause, more specifically, suppose that law $\mathcal{L}$ contains the following rule r:

```
h :- ..., delegate(g), ...
```

If the evaluation of $\mathcal{L}$ gets to the `delegate(g)` clause of rule `r`, then goal `g` is submitted to the delta $\overline{\mathcal{L}'}$ for evaluation, playing the role of the pseudo event *pe* in Equation 2. This evaluation by the delta will produce a list of operations *pr*, which would be fed back to $\mathcal{L}$, as the ruling proposal of the delta, for goal `g`. The operations thus proposed are provisionally added to the ruling, but their final disposition will be determined by $\mathcal{L}$, as we shall see later.

Finally, note that `delegate(g)` clause in a law $\mathcal{L}$ has no effect when $\mathcal{L}$ has no delta.

The Structure of Law Deltas: A refining delta $\overline{\mathcal{L}'}$ of a law $\mathcal{L}$ looks pretty much like the root-law $\mathcal{L}_0$ of the law-tree, with two distinctions:

First, the top clause in the delta is

```
law(name(L'),ca(pk)) refines L,
```

where `L` is the name of the law being refined, `L'` is the name of this delta, and `pk` is the public key of the certifying authority to certify the controllers enforcing law `L'`.

Second, the heads of the rules in $\overline{\mathcal{L}'}$ need to match the goals delegated to it by law $\mathcal{L}$, and not the original regulated events that must be matched by the rules of the root-law $\mathcal{L}_0$. Although the goals delegated to a refining delta can, and often do, take the form of regulated events, like `sent(...)`, and `arrived(...)`, as is the case in our case study in Section 5.

Finally, note that each delta has read access to the entire control-state (CS) of the agent. That is, the rules that constitute a given delta can contain arbitrary conditions involving all the terms of the CS.

The Disposition of Ruling Proposals: A law $\mathcal{L}$ can specify the disposition of operation in the ruling proposal returned to it by any refining delta $\overline{\mathcal{L}'}$. This is done via *rewrite rules* of the form:

```
rewrite(O) :- C,replace(Olist)
```

where `O` is some term, `C` is some condition, and `Olist` is a possibly empty list of operations. The effect of these rules are as follows. First let *rp* be the set of terms proposed by a refining delta in response to the execution of a delegate clause in $\mathcal{L}$. For each term p in *rp*, a goal `rewrite(p)` is submitted for evaluation by law $\mathcal{L}$. If this evaluation fails, which happens, in particular, if none of the `rewrite(O)` rules in $\mathcal{L}$ matches this goal, then term p is added to the ruling of law $\mathcal{L}$.

If, on the other hand, the evaluation succeeds by matching one of the `rewrite` rules and condition `C` of this rule evaluates to true, then p is replaced by the list `Olist`. `Olist` is then added to the ruling of $\mathcal{L}$. Note that if `Olist` is empty, then term p would be discarded in spite of its inclusion in the ruling proposal made by the refining delta. For example, the *rewrite rule* of law $\mathcal{P}_{\mathcal{C}}$ in Figure 4, which is the implementation of our example coalition policy, will discard any proposed `forward` operations of `service currency` movement from its refining delta to make sure no service currency can be forged. Further, `C` cannot contain a `delegate` clause so that no further consultation is possible with the refining delta on the disposition of p; we believe that this constraint keeps the model easy to understand without real loss of flexibility.

So, the `rewrite` rules of a law $\mathcal{L}$ determine what is to be done with each operation proposed by a refining delta: whether it should be blocked, included in the ruling, or replaced by some list of operations. Note that each `rewrite` rule is applied to the ruling proposal returned by a refining delta, regardless of which `delegate` clause originally led to the consultation with the refining delta.

Finally, LGI features another technique to regulate the effect of a refining delta on the eventual ruling of the law. It can protect certain terms in the control-state from modification by refining deltas of a given law $\mathcal{L}$. This is done by including the clause

```
protected(T)
```

in the `Preamble` clause of law $\mathcal{L}$ (see Figure 4 for example), where `T` is a list of terms. For example, if the following statement appears in $\mathcal{L}$,

```
protected([name(_),role(_)])
```

then no refinement of $\mathcal{L}$ can propose an operation that modifies the terms `name` and/or `role`. Strictly speaking, such protection of terms in the control state can be carried out via `rewrite` rules, but the `protected` clauses are much more convenient for this purpose.

4.3 On the Basis for Trust between Members of Different Communities

Recall that a law may require that all the controllers used to interpret it are certified by a specific CA. And different, independent, laws may require different CAs for this purpose (see Section 3.1). The situation with laws related via the *subordinate* relation is as follows: Consider a law $\mathcal{L}$ that requires certification by ca, and suppose that law $\mathcal{L}'$, which is subordinate to $\mathcal{L}$, requires certification by ca'. In this case we require the controllers interpreting law $\mathcal{L}'$ to be certified by both CAs: ca and ca'.

Now, consider agents x and y operating under laws $\mathcal{L}_x$ and $\mathcal{L}_y$, respectively. And suppose that these laws permit these agents to exchange messages, provided that both laws above are *subordinate* to a common law, say $\mathcal{L}$. When these two agents communicate, it would be necessary to ensure that: (a) the exchange is mediated by properly certified controllers; and (b) that these controllers are interpreting laws that have a common superior.

The first assurance above is obtained by having the controller of x (and similarly for that of y) authenticated by the CA required by the common superior law $\mathcal{L}$, as well as by the CA required by law $\mathcal{L}_x$, if any. The second assurance is obtained by checking the *subordinate* relationship of the laws (all the laws are identified by their one-way hashes). In our law-language this kind of check is carried out by the predicate

`conforms(`L'`,`L`),`

which is true if and only if law $\mathcal{L}'$ is identical, or subordinate, to law $\mathcal{L}$. (See Figure 4 for an example of the use of this predicate.)

5 A Case Study

We now show how the policy hierarchy described in Section 2 can be specified in LGI, using the mechanisms just introduced. We start by formalizing the coalition policy P_C, making it into an LGI law $\mathcal{P}_{\mathcal{C}}$. We then formalize policies P_1 and P_2 via deltas that refine law $\mathcal{P}_{\mathcal{C}}$.

5.1 Establishing the Coalition Policy P_C

Law $\mathcal{P}_C$, which implements policy P_C, is shown in Figures 4. This, like any other LGI laws, has two parts: (a) a `Preamble`, which specifies such things as certifying authorities acceptable to this law, and the initial control state of agents operating under it; and (b) the set of rules, whose formal statement is followed by informal comments, in italics.

$\mathcal{P}_C$'s `Preamble` has several clauses. The `law` clause indicate that this is a root-law, specifying its name, and the public key of the CA that is to be used for certifying the controllers interpreting this law. The `authority` clause specifies the public key of a CA—the `coalition director` D_c, whose certification would be accepted by this law for the authentication of the `member enterprise directors` D_i—as we shall see. The `initialCS` clause specifies that the initial control state of everybody who adopt this law would be empty. Finally, the `protected` clause specifies four control-state terms that would be protected from change by refining deltas. They are: `myDirector(_,_)`, `budget(_,_)`, `name(_)`, and `role(_)`, which we will discuss later.

Our discussion of the rules of this law is organized as follows: We start with how directors of enterprises, and agents working for them, are authenticated; and how an agent can claim his official names and the role he is to play within its enterprise. Second, we show how service budgets are allocated by enterprise directors, how they are distributed by the coalition director, and service currency is to be moved from one agent to another. Third, we discuss how service request are sent, and what happens if a request is rejected. Finally, we show how the coalition law $\mathcal{P}_C$ limits the power of the deltas that might be used to refine it.

(a) Coalition Membership, and Agent Authentication: For the director D of enterprise E to claim his role under this law, he needs to present a certificate issued by the coalition director D_C, with the attribute `role(directorOf(E))`. By Rule $\mathcal{R}1$, this would cause the term `directorOf(E)` to be added to his control-state.

For any other agent `x` of the enterprise E to join the coalition, it needs to present two certificates: One, issued by coalition director D_C to certify its director D, and getting its public key ; this is done via Rule $\mathcal{R}1$, which will add the term `myDirector(D,E)` to the CS of x. The other, issued by its director `D` to certify the name and role of `x`; this is done via Rule $\mathcal{R}2$, which will add the attribute `name(N)` and possible `role(R)`, to the CS of x.

Note that the terms added to the CS of various agents as the result of such certification serve as a kind of *seals* that authenticate the role of these agents for their interaction with other agents in the coalition.

(b) Allocation of Budgets, and Moving Currencies: By Rule $\mathcal{R}3$, a director D of enterprise E can send a message `grantBudget(B,E)` to the coalition director D_C, which (by Rule $\mathcal{R}4$) will add the term `budget(B,E)` to the CS of D_C, upon its arrival. This budget for services by enterprise E can be redistributed (by Rules $\mathcal{R}5$ and $\mathcal{R}6$) among the directors of other enterprises.

Preamble:

```
law(name(P_C),ca(publicKey1)).
authority(D_C,publicKey2).
initialCS([]).
protected([myDirector(_,_),budget(_,_),name(_),role(_)]).

certified([issuer([e1Admin,e2Admin]),subject(E3Admin),attributes([join(newEnt)]) ]) :- gets a
```

$\mathcal{R}1$.
```
certified([issuer(D_C),subject(D),attributes([role(directorOf(E))])])
      :- if (D==Self) then do(+role(directorOf(E))) else do(+myDirector(D,E)).
```
The director D *of enterprise* E *needs to be certified by a certificate issued by the* coalition director D_C.

$\mathcal{R}2$.
```
certified([issuer(D),subject(Self),attributes(A)]) :- myDirector(D,E)@CS,name(N)@A,
         do(+name(N)), if role(server)@A then do(+role(server)).
```
Claiming the name and role within one coalition member enterprise E *via certificate issued by the director* D *of that enterprise.*

$\mathcal{R}3$.
```
sent(D,grantBudget(B,E),[D_C,ThisLaw]) :- role(directorOf(E))@CS, do(forward).
```
Only the director of the member enterprise E *can contribute the initial budget* (B,E) *to the coalition director D_C.*

$\mathcal{R}4$.
```
arrived([D,Ld],grantBudget(B,E),D_C) :- conforms(Ld,ThisLaw),
         if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else
         do(+budget(B,E)), do(deliver).
```
The contributed budget B *from enterprise* E *will be recorded as term* budget(B,E) *in the control state of the coalition director.*

$\mathcal{R}5$.
```
sent(D_C,grantBudget(B,E),[D,Ld]) :- conforms(Ld,ThisLaw),
         budget(B1,E)@CS, B1 >= B, do(budget(B1,E)<-budget(B1-B,E)), do(forward).
```
The initial budget assignment from the coalition director to the member enterprise directors.

$\mathcal{R}6$.
```
arrived([D_C,ThisLaw],grantBudget(B,E),D) :- role(directorOf(Ei))@CS,
         if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else
         do(+budget(B,E)), do(deliver).
```
Only the member enterprise's director can receive the initial budget from the coalition director.

$\mathcal{R}7$.
```
sent(X,giveCurrency(B,E),[Y,Ly])
      :- conforms(Ly,ThisLaw), budget(B1,E)@CS, B1>=B, delegate(ThisGoal), if
         (permitS@Ruling) then do(budget(B1,E)<-budget(B1-B,E)), do(forward).
```
An agent can give part of its service currency to others if it is authorized by the local law delta of its enterprise.

$\mathcal{R}8$.
```
arrived([X,Lx],giveCurrency(B,E),Y) :- conforms(Lx,ThisLaw),
         if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else
         do(+budget(B,E)), do(deliver).
```
The service currency given by other agents will be added into the receiver's account.

$\mathcal{R}9$.
```
sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :- conforms(Ly,ThisLaw),
         name(N)@CS, budget(B,E)@CS,B >= P, delegate(ThisGoal),
         if (permitS@Ruling) do(budget(B,E)<-budget(B-P,E)), do(forward).
```
Any service request must contain the name of the sender and the payment, and be authorized by the local law delta of the sender enterprise.

$\mathcal{R}10$.
```
arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y) :- conforms(Lx,ThisLaw),
         if (myDirector(D,E)@CS, role(server)@CS, delegate(ThisGoal),
         permitA@Ruling) then
         (budget(B,E)@CS, do(budget(B,E)<-budget(B+P,E)), do(deliver))
         else (name(M)@CS,
         do(forward(Self,srReject(from(M),service(S),payment(P,E)),[X,Lx]))).
```
Only a certified server can process the service request if it is authorized by its local enterprise law delta. Otherwise, the request will be rejected.

$\mathcal{R}11$.
```
arrived([X,Lx],srReject(from(N),service(S),payment(P,E)),Y) :-
         conforms(Lx,ThisLaw), budget(B,E)@CS, do(budget(B,E)<-budget(B+P,E)),
         do(deliver).
```
The arrival of the service reject message will restore the client's budget by that payment.

$\mathcal{R}12$.
```
rewrite(forward(X,M,[Y,Ly])) :- conforms(Ly,ThisLaw),
         if (M==(grantBudget(_,_)
         |giveCurrency(_,_)|sr(_,_,_)
         |srReject(_,_,_))) then replace([]).
```
Make sure that no subordinate law delta can violate the properties *of this law. without*

Fig. 4. Law $\mathcal{P}_C$

Note that the term `budget(B,E)` in the CS of any agent represents under this law what we have called E-currency, i.e., currency that can be used for paying servers in enterprise E. The law allows such currencies to be *moved* from one agent to another, subject to the various policies, but it provides no means for forging currencies. E-currency can be moved by any agent to another via the `giveCurrency(B,E)` message, regulated by Rules $\mathcal{R}7$ and Rule $\mathcal{R}8$, provided that such transfer is permitted by the enterprise refining delta.

It is instructive to observe how Rule $\mathcal{R}7$ checks for the permission of currency transfer by a refining delta. After the `delegate(ThisGoal)` clause (`ThisGoal` would be bound to the head of the evaluated rule, in this case, the currency `sent` event), a check is performed to see whether an operation `permitS` is in the ruling compiled thus far (`Ruling`); if `permitS` is present, then the currency transfer is performed.

Finally, we note that both the redistribution of the service currency by the enterprise's director, and the subsequent movement of it among the regular agents, are governed by the same Rule $\mathcal{R}7$ and Rule $\mathcal{R}8$.

(c) Regulation over Service Requests: By Rule $\mathcal{R}9$, an agent with official name `N`, certified by its enterprise director, can issue request for service `S` of enterprise `E` via message:

```
sr(from(N),service(S),payment(P,E))
```

if the payment `P` doesn't exceed its current budget and the request is authorized by its refining delta. If it is the case, the corresponding payment will be reduced from the sender's budget before the service request is forwarded.

By Rule $\mathcal{R}10$, only the certified server can receive a service request if the payment type is correct and the request is authorized by the server's local enterprise refining law delta. If this is the case, the two operations will be carried out: (1) the server's budget will increase by that payment, and (2) the request will be delivered to the server.

Otherwise, a service rejection message with the previous payment will be forwarded back to the client and the client will get the refund of that payment as in Rule $\mathcal{R}11$. Both Rule $\mathcal{R}9$ and Rule $\mathcal{R}10$ use the similar process as Rule $\mathcal{R}7$ to check the permission of the request sending and receiving from the refining delta.

(d) Limiting the Power of Refining Deltas: Finally, $\mathcal{P}_\mathcal{C}$ imposes some limitations on the possible effects of refining deltas, to ensure such things as that currencies cannot be forged. First, as has already been pointed out, the `protected` clause in the preamble of this law does not allow certain terms to be changed by deltas.

Second, by Rule $\mathcal{R}12$, this law would reject any recommendation by delta to forward certain messages, such as `grantBudget` and `giveCurrency` messages that might cause, effectively, currency to be forged. This is done because $\mathcal{P}_\mathcal{C}$ cannot depend on refining deltas not to violate these important properties that our law is responsible for.

5.2 Establishing Local Policies of Member-Enterprises

Law $\mathcal{P}_1$ *of Enterprise* E_1*:* This law is formed by refining law $\mathcal{P}_C$ via delta $\overline{\mathcal{P}_1}$, which is shown in Figure 5. As specified in the preamble of this delta, the controllers interpreting this law would be required to be certified by the CA identified by the public key `publicKey3`—as well as by the CA required by the superior law $\mathcal{P}_C$. Furthermore, the preamble specifies the address of agent, which is to be used as an audit-trail server under this law.

$\mathcal{P}$reamble:

law(name($\mathcal{P}_1$),ca(publicKey3)) **refines** $\mathcal{P}_C$.
alias(e1Auditor,"e1Auditor@e1.com").

$\mathcal{R}1$.
```
sent(X,giveCurrency(B,E),[Y,Ly]) :-
        role(directorOf(e1))@CS, do(permitS).
```
In this enterprise, only the enterprise director can give its own service currency to others.

$\mathcal{R}2$.
```
sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :-
        do(permitS), do(deliver(Self,ThisGoal,e1Auditor)).
```
Authorize all the service requests. Furthermore, monitor all the service requests sent by the agents of this enterprise.

$\mathcal{R}3$.
```
arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y) :-
        do(permitA), do(deliver(Self,ThisGoal,e1Auditor)).
```
Authorize all the service requests. Furthermore, monitor all the service requests received by the servers of this enterprise.

Fig. 5. Law delta $\overline{\mathcal{P}_1}$

By Rule $\mathcal{R}1$, law $\mathcal{P}_1$ allows only the director of enterprise E_1 to distribute currencies within the enterprise. No other agent can transfer its currencies to others.

Rules $\mathcal{R}2$ and $\mathcal{R}3$ ensure that copies of all the service requests sent by agents of E_1 and those received by the servers of E_1 will be sent to its own audit-trail server–`e1Auditor`.

Law $\mathcal{P}_2$ *of Enterprise* E_2*:* This law is formed by refining law $\mathcal{P}_C$ via delta $\overline{\mathcal{P}_2}$, which is shown in Figure 6. This law differs from $\mathcal{P}_1$, in the following respects: First, by Rule $\mathcal{R}1$, any agent in enterprise E_2 can transfer part of any service currency it has to others, and any such currency movement will be audited by the enterprise's auditor `e2Auditor`.

Also, by Rules $\mathcal{R}2$ and $\mathcal{R}3$, agents of E_2 are allowed to engaged in coalition activities only between 6:00PM to 8:00AM; where, by "coalition activity" we mean sending or receiving service requests.

6 Conclusions

This paper introduces a new, and fully implemented, regulatory mechanism for coalitions. The main contributions of this mechanism are as follows:

First, this mechanism is based on a very general view of the governance of coalitions, assuming that each coalition C is governed by a global policy P_C, *Preamble:*

```
    law(name(P2),ca(publicKey4)) refines PC.
    alias(e2Auditor,"e2Auditor@e2.com").

R1. sent(X,giveCurrency(B,E),[Y,Ly]) :-
            do(permitS), do(deliver(Self,ThisGoal,e2Auditor)).
```
Any agent in this enterprise can give part of its service currency to others and that currency movement will be monitored by this enterprise.

```
R2. sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :-
            clock(T)@CS, if (T > 6:00PM; T < 8:00AM) then do(permitS).
```
The agents of E_2 are allowed to send the coalition service requests only after the normal working hour,namely, from 6:00PM to 8:00AM.

```
R3. arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y) :-
            clock(T)@CS, if (T > 6:00PM; T < 8:00AM) then do(permitA).
```
The servers of E_2 are allowed to receive and process the coalition service requests only after the normal working hour, namely, from 6:00PM to 8:00AM.

Fig. 6. Law delta $\overline{\mathcal{P}_2}$

and that each coalition member E_i is governed by its own policy P_i, which must conform to P_C.

Second, since it is based on LGI, our mechanism can support a wide range of highly dynamic (stateful) policies, and it enforces these policies in an efficient and decentralized manner.

Finally, our mechanism satisfies the following *flexibility* property:

> Each internal policy P_i can be defined and changed independently of the internal policies of other coalition members, and without any knowledge of them.

Such a flexibility is important for several reasons. First, it provides each enterprise with the *autonomy* to define its own policy at will, subject only to the pre-agreed coalition policy P_C. Second, the mutual independence of the internal policies of member enterprises simplifies their formation and their evolution. Finally, this principle allows an individual enterprise to keep its own policy *confidential*, since policies of other enterprises do not depend on it.

References

1. X. Ao, N. Minsky, T. Nguyen, and V. Ungureanu. Law-governed communities over the internet. In *Proc. of Fourth International Conference on Coordination Models and Languages; Limassol, Cyprus; LNCS 1906*, pages 133–147, September 2000.
2. X. Ao, N. Minsky, and V. Ungureanu. Formal treatment of certificate revocation under communal access control. In *Proc. of the 2001 IEEE Symposium on Security and Privacy, May 2001, Oakland California*, pages 116–127, May 2001.
3. X. Ao, N.H. Minsky, and T.D. Nguyen. A hierarchical policy specification language, and enforcement mechanism, for governing digital enterprises. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 38–49, June 2002.

4. A. Belokosztolszki and K. Moody. Meta-policies for distributed role-based access control systems. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 106–115, June 2002.
5. C. Bidan and V. Issarny. Dealing with multi-policy security in large open distributed systems. In *Proceedings of 5th European Symposium on Research in Computer Security*, pages 51–66, September 1998.
6. P. Bonatti, S. D. Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 164–173, 2000.
7. W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer-Verlag, 1981.
8. V. Gligor, H. Khurana, R. Koleva, V. Bharadwaj, and J. Baras. On the negotiation of access control policies. In *Proc. of the Security Protocols Workshop, Cambridge, UK*, April 2001.
9. L. Gong and X. Qian. Computational issues in secure interoperation. *IEEE Transctions on Software Engineering*, pages 43–52, January 1996.
10. G. Karjoth. The authorization service of tivoli policy director. In *Proc. of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, December 2001. (to appear).
11. H. Khurana, V. Gligor, and J. Linn. Reasoning about joint administration of access policies for coalition resources. In *Proc. of IEEE Int. Conf. On Distr. Computing (ICDCS), Vienna, Austria*, pages 429–440, July 2002.
12. P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 66–80, May 2002.
13. N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.
14. N.H. Minsky and V. Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *7th USENIX Security Symposium*, January 1998.
15. N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *TOSEM, ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.
16. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 50–59, June 2002.
17. B. Schneier. *Applied Cryptography.* John Wiley and Sons, 1996.
18. D. Shands, R. Yee, J. Jacobs, and E. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. In *Proc. of Network and Distributed System Security Symposium, San Diego, California*, Feb 2000.
19. M. Thomson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distrbuted resources. In *Proceedings of 8th USENIX Security Symposium*, August 1999.
20. D. Wijesekera and S. Jajodia. Policy algebras for access control: the propositional case. In *Proceedings of the 8th ACM conference on Computer and communications security*, pages 38–47, 2001.

Initiator-Resilient Universally Composable Key Exchange

Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt

IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth,
Fakultät für Informatik, Universität Karlsruhe, 76 131 Karlsruhe, Germany
{hofheinz,muellerq,steinwan}@ira.uka.de

Abstract. Key exchange protocols in the setting of *universal composability* are investigated. First we show that the ideal functionality $\mathcal{F}_{\mathrm{KE}}$ of [9] cannot be realized in the presence of adaptive adversaries, thereby disproving a claim in [9]. We proceed to propose a modification $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, which is proven to be realizable by two natural protocols for key exchange. Furthermore, sufficient conditions for securely realizing this modified functionality are given. Two notions of key exchange are introduced that allow for security statements even when one party is corrupted. Two natural key exchange protocols are proven to fulfill the "weaker" of these notions, and a construction for deriving protocols that satisfy the "stronger" notion is given.

Keywords: formal cryptography, cryptographic protocols, universal composition, key exchange.

1 Introduction

It is generally agreed upon that providing only non-formal intuitive security statements about cryptographic schemes and protocols is not satisfying. Consequently, models have been developed which try to provide formally satisfying notions of *security* in various settings. The covered topics range from security notions for symmetric and asymmetric encryption schemes, over security concepts for signature schemes to security notions for arbitrary protocols.

We do not try to give a survey of all the work that has been done in this area, but it is worth pointing out that in the cryptographic research community much work can be traced back to a seminal paper of Goldwasser and Micali [16]. As already indicated by the title of the latter, formal approaches in this line of research are often well-suited to model probabilistic aspects of attacks, and attacks which make sophisticated use of the inner structure of messages. Despite some well-known proof methodologies, the typically encountered (reduction) proofs are "hand-made". On the other hand, in the security research community, much focus has been put on the use of term rewriting and formal proof systems. One particularly important model is due to Dolev and Yao [15]. Both the approach of the "crypto camp" and the approach of the "security camp" have clearly led to remarkable results. Unfortunately, at the moment there seems to be a clear gap between these two "camps". In research on protocol security, the situation is quite similar—two different models are used, and both of them have proven to be useful: The model of Canetti [6], e. g., allowed for interesting insights in the limitations of the composability of two-party computations [10], and the approach of Pfitzmann and Waidner [20] led to the development of a *universally composable cryptographic library* [2], for instance. In fact, the latter work can be seen as a very interesting step towards closing the gap between the cryptographic and the security research community. Our contribution is formulated in the model of Canetti and deals with notions for the security of key exchange; it can be seen in the line of the works [4,3,21,8,9]. So far, we have not explored to what extent our results can be adapted to the setting of Pfitzmann and Waidner, where the problem of key exchange has been explored, e. g., by Steiner [22].

In [6], a very strict notion of security is given which guarantees *universal composability* of protocols. More specifically, that means that given any secure protocol π which utilizes an idealized version $\mathcal{F}$ of a protocol task (called an *ideal functionality*), another protocol τ which in turn securely realizes $\mathcal{F}$ can replace a polynomial number of instances of $\mathcal{F}$ in protocol π without compromising the overall security of π. Key exchange protocols in this setting were studied in [9]. However, as we will show in the following, the security notion of [9] cannot be fulfilled when considering *adaptive* adversaries, which may corrupt participants of the protocol at any time during the protocol execution. In this contribution we will therefore provide a slightly modified specification for key exchange realizable in the presence of adaptive adversaries. Furthermore, two natural key exchange protocols are proven secure in that sense. In fact, we investigate general sufficient conditions for key exchange protocols to be secure with respect to our notion.

In view of universal composability one must not restrict attention to the case where the "initiator" and the "responder" of a key exchange are uncorrupted and need to be protected against an adversary monitoring the communication channel "from the outside". To be able to employ a key exchange protocol within a more complicated protocol context it is necessary to specify the behavior of a key exchange protocol also for the case when the initiator or the responder are corrupted. In [9], in face of a corrupted initiator *or* responder, the adversary may freely choose the key which is to be the outcome of the key exchange protocol. Investigating, e. g., a Diffie-Hellman-like key exchange we observe that it is not obvious how the initiator could, if corrupted, let the adversary freely choose the key agreed upon. This leads to the natural question whether or not some known key exchange protocols may in fact realize something strictly stronger than a universally composable key exchange as described in [9] (resp., in Section 3 below). Specifically, it seems that a Diffie-Hellman-like key exchange is "initiator-resilient" in the sense that a corrupted initiator cannot force the outcome of the key exchange to be some specific key, which could then be known to some third party or be some "weak" key of an encryption functionality to be used after the key exchange.

To make this intuition explicit, we first give a very straightforward and intuitive ideal functionality for initiator-resilient key exchange where even in case of a corrupted initiator, the key agreed upon is chosen at random. It turns out that this ideal functionality can be realized securely, although it might be considered "too restrictive", as two natural and "intuitively initiator-resilient" key exchange protocols can be shown not to realize this ideal functionality. Therefore, we also present a slightly more involved ideal functionality making use of a *non-information oracle*, as defined in [9].

In the new ideal functionalities introduced in this contribution, the adversary still has complete control over the outcome of the key exchange when the *responder* gets corrupted. Yet a close inspection of, e. g., a Diffie-Hellman-like key exchange protocol suggests that there exist key exchange protocols for which the influence each individual party has on the key is limited. It is an interesting open question if this additional property of certain key exchange protocols can be captured in an appropriate ideal functionality.

2 Preliminaries

To start, we shortly outline the framework for multi-party protocols defined in [6]. First of all, *parties* (denoted by P_1 through P_n) are modeled as *interactive Turing machines (ITMs)* (cf. [6]) and are supposed to run some (fixed) protocol π. There also is an *adversary* (denoted $\mathcal{A}$ and modeled as an ITM as well) carrying out attacks on protocol π. Therefore, $\mathcal{A}$ may corrupt parties (in which case it learns the party's current state and the contents of all its tapes, and controls its future actions), and intercept or, when assuming unauthenticated message transfer[1], also fake messages sent between parties. If $\mathcal{A}$ corrupts parties only *before* the actual protocol run of π takes place, $\mathcal{A}$ is called *non-adaptive*, otherwise $\mathcal{A}$ is said to be *adaptive*. The respective local inputs for protocol π are supplied by an *environment machine* (modeled as an ITM and denoted $\mathcal{Z}$), which may also read all outputs locally made by the parties and communicate with the adversary. Here we will only deal with environments guaranteeing a polynomial (in the security parameter) number of total steps all participating ITMs run. For more discussion on this issue, cf. [17].

The model we have just described is called the *real* model of computation. In contrast to this, the *ideal* model of computation is defined just like the real model, with the following exceptions: we have an additional ITM called the *ideal functionality* $\mathcal{F}$ and being able to send messages to and receive messages from the parties privately (i. e., without the adversary being able to even intercept these messages). The ideal functionality may not be corrupted by the adversary, yet may send messages to and receive messages from it. Furthermore, the parties $P_1, \ldots, P_n$ are replaced by *dummy parties* $\tilde{P}_1, \ldots, \tilde{P}_n$ which simply forward their respective inputs to $\mathcal{F}$ and take messages received from $\mathcal{F}$ as output. Finally, the adversary in the ideal model is called the *simulator* and denoted $\mathcal{S}$. The only means of attack the simulator has in the ideal model are those of corrupting parties, delaying or even suppressing messages sent from $\mathcal{F}$ to a party, and all actions that are explicitly specified in $\mathcal{F}$. However, $\mathcal{S}$ has no access to the

[1] In [9], the model for message transfer is called *unauthenticated*, even when each ordered pair (P_i, P_j) of parties is allowed to exchange *one* message in an authenticated manner (i. e., the adversary is unable to fake such a message).

contents of the messages sent from $\mathcal{F}$ to the dummy parties (except in the case the receiving party is corrupted) nor are there any messages actually sent between (uncorrupted) parties $\mathcal{S}$ could intercept. Intuitively, the ideal model of computation (or, more precisely, the ideal functionality $\mathcal{F}$ itself) should represent what we ideally expect a protocol to do. In fact, for a number of standard tasks, there are formulations as such ideal functionalities (see, e. g., [6]).

To decide whether or not a given protocol π does what we would ideally expect some ideal functionality $\mathcal{F}$ to do, the framework of [6] uses a *simulatability*-based approach: at a time of its choice, $\mathcal{Z}$ may enter its halt state and leave output on its output tape. The random variable describing the first bit of $\mathcal{Z}$'s output will be denoted by $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run on *security parameter* $k \in \mathbb{N}$ and initial input $z \in \{0,1\}^*$ (which may, in case of a non-uniform $\mathcal{Z}$, depend on k) in the real model of computation, and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run in the ideal model. Now if for any adversary $\mathcal{A}$ in the real model, there exists a simulator $\mathcal{S}$ in the ideal model such that for *any* environment $\mathcal{Z}$ and *any* initial input z, we have that

$$|\mathbf{P}(\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)=1) - \mathbf{P}(\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)=1)| \qquad (1)$$

is a negligible[2] function in k, then protocol π is said to *securely realize* functionality $\mathcal{F}$ [3]. Intuitively, this means that any attack carried out by adversary $\mathcal{A}$ in the real model can also be carried out in the idealized modeling with an ideal functionality by the simulator $\mathcal{S}$ (hence the name), such that no environment is able to tell the difference. Analogously to [6] we restrict to protocols which generate output if all messages are delivered and no party gets corrupted.

Remark 1. In the framework of [6], the above definition of security is equivalent to the seemingly weaker requirement that there is a simulator $\mathcal{S}$ so that (1) is a negligible function in k for any environment $\mathcal{Z}$ and input z, and the special real-model *dummy adversary* $\tilde{\mathcal{A}}$, which follows explicit instructions from $\mathcal{Z}$.

Remark 2. The original modeling of [6] does not involve an explicit message sent to the ideal functionality upon party corruptions. Yet exactly this additional feature proved helpful in later works (e. g., [9,11]) and in particular allows to formulate key exchange functionalities in a convenient way. This change does not affect the validity of the crucial *composition theorem* proven in [6].

Remark 3. In [6], the environment machine is modeled as a *non-uniform* ITM (i. e., as an ITM having input $z = z(k)$ dependent on the security parameter k). However, as the *composition theorem* of [6] remains valid when restricting to *uniform* environment machines (i. e., those with input not dependent on k, cf. [17]), it makes sense to alternatively consider only uniform environments where appropriate. In particular, all proofs given below hold for both uniform and non-uniform environments; alone the respective assumptions (i. e., the decisional Diffie-Hellman assumption) have to be considered with respect to the uniformity class in question.

[2] A function $f : \mathbb{N} \to \mathbb{R}$ is called *negligible*, if for any $c \in \mathbb{N}$, there is a $k_0 \in \mathbb{N}$ such that $|f(k)| < k^{-c}$ for all $k > k_0$.

[3] The formulation in [6] is slightly different, but equivalent to the one chosen here which allows to simplify our presentation.

Functionality $\mathcal{F}_{\mathrm{KE}}$

$\mathcal{F}_{\mathrm{KE}}$ proceeds as follows, running on security parameter k, with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value `(Establish-session,`sid`,`P_i`,`P_j`,`$role$`)` from some party P_i, record the tuple $(sid, P_i, P_j, role)$ and send this tuple to the adversary. In addition, if there already is a recorded tuple $(sid, P_j, P_i, role')$ (either with $role' \neq role$ or $role' = role$) then proceed as follows:
   (a) If P_i and P_j are uncorrupted then choose $\kappa \overset{R}{\leftarrow} \{0,1\}^k$, send `(key,`$sid$`,`$\kappa$`)` to P_i and P_j, send `(key,`sid`,`P_i`,`P_j`)` to the adversary, and halt.
   (b) If either P_i or P_j is corrupted, then send a message `(Choose-value,`sid`,` P_i`,`P_j`)` to the adversary; receive a value κ from the adversary, send `(key,`sid`,`κ`)` to P_i and P_j, and halt.
2. Upon corruption of either P_i or P_j, proceed as follows. If the session key is not yet sent (i. e., it was not yet written on the outgoing communication tape), then provide $\mathcal{S}$ with the session key. Otherwise provide no information to $\mathcal{S}$.

Fig. 1. The ideal functionality $\mathcal{F}_{\mathrm{KE}}$ from [9]

3 Key Exchange

Now we are ready to show the ideal functionality $\mathcal{F}_{\mathrm{KE}}$ from [9] (see also Figure 1) to be non-realizable if adversaries are allowed to corrupt adaptively. The key observation in our argument is that in the formulation of [9], the functionality $\mathcal{F}_{\mathrm{KE}}$ determines the common key later handed to both participants right after the respective initialization messages arrived. As the ideal-model adversary cannot delay or block these initialization messages, this happens right at the start of the protocol. Furthermore, if at this point in time, neither participant is corrupted, the session key is chosen uniformly by $\mathcal{F}_{\mathrm{KE}}$. Yet again, in the real model, at least one of the two participants should be able to influence the session key! (Although this is intuitively clear, showing it in our situation makes up the main part of the proof below.)

More specifically, we show that corrupting a party right after it received input and then running the protocol completely through this corrupted party causes the output of the other party to allow for distinguishing real from ideal. In the ideal model, this output is chosen uniformly by $\mathcal{F}_{\mathrm{KE}}$, whereas in the real model, it results from a factually executed key exchange[4]. The proof of Theorem 14 in [9] simply does not take into account such malicious behavior of a party that was corrupted *after* it received its initialization message. This applies in particular to the reduction in the proof of [9, Theorem 14] of an environment $\mathcal{Z}$ (together with an adversary $\mathcal{A}$) to an attacker $\mathcal{A}'$ in the model of SK-security. Generalizing this kind of argument[5] seems possible, thereby excluding the realizability of, e. g.,

[4] Recently we learned that our argument is very similar to the one of [13] presented against the bit commitment functionality $\mathcal{F}_{\mathrm{COM}}$ from [6,7].

[5] i. e., giving input to an uncorrupted party P_i and then, after that party forwarded its input to the ideal functionality in the ideal model, corrupting P_i and forcing it to take part in a real protocol run with *different* input.

the functionality $\mathcal{F}_{\text{SFE}}$ (as formulated in [6]) for secure function evaluation even in the $\mathcal{F}_{\text{CRS}}$-hybrid model[6]. Here, $\mathcal{F}_{\text{CRS}}$ denotes the *common reference string functionality* as used in [11].

To overcome such problems, we introduce a modified key exchange functionality and prove two common protocols to be secure realizations hereof. In fact, these protocols are very similar to the ones considered in [3] for key exchange. However, our key exchange functionality differs from $\mathcal{F}_{\text{KE}}$ in several aspects: first, the common key may be chosen by the ideal-model adversary if at the *end* of a simulated protocol run, anyone of the participants is corrupted. Moreover, to exclude complications conditional on the order and roles in which parties are asked to perform a key exchange, we define a family $\{\mathcal{F}_{\text{KE}}^{(i,j)}\}_{P_i,P_j}$ of ideal functionalities indexed by the parties involved and thus implicitly fixing the respective roles they take in the key exchange. We remark that there is also a subtlety regarding the distribution from which the common keys are picked. As with $\mathcal{F}_{\text{KE}}$ from [9], we demand random k-bit strings (where k is the security parameter) for keys. On the other hand, the "raw" output resulting from a, say, Diffie-Hellman-like key exchange may be computationally distinguishable from random k-bit strings, even under the decisional Diffie-Hellman assumption. In the case of Diffie-Hellman-like key exchange protocols, we therefore follow the approach in [21, Section 5.2.2] and use a family of pair-wise independent hash functions to pass from random group elements to random bitstrings.

Proposition 1. *Presuming authenticated links and no further set-up assumptions, $\mathcal{F}_{\text{KE}}$ from [9] cannot be securely realized by any two-party protocol π terminating in strict polynomial time if adversarial corruption is adaptive.*

Proof. Assume that π securely realizes $\mathcal{F}_{\text{KE}}$. Let $m(k)$ be a polynomial bounding the total number of messages sent between parties while performing π. Furthermore, let's fix two distinct parties P_i and P_j. To cover ideal-model adversaries $\mathcal{S}$ which do not guarantee timely delivery of the common key, we introduce the following environment $\mathcal{Z}_1$ (expecting to be run with the *dummy adversary* $\tilde{\mathcal{A}}$ in the real model):

1. Activate P_i with `(Establish-session,`sid`,`P_i`,`P_j`,initiator)`.
2. Activate P_j with `(Establish-session,`sid`,`P_j`,`P_i`,responder)`.
3. Advise the adversary to deliver all messages between P_i and P_j, but at most $m(k)$ messages in total.
4. If P_i or P_j outputs a key, call it α, resp. β; if both P_i and P_j output keys, output 1, else output 0.

Since π is terminating, in the real model $\mathcal{Z}_1$ always outputs 1. Moreover, as π securely realizes $\mathcal{F}_{\text{KE}}$, $\mathcal{Z}_1$ must also output 1 in the ideal model in all but a negligible fraction of runs. That means we may assume that in a "normal"

[6] After acceptance of this paper, a revised version [12] of [11] was published in the IACR ePrint archive; this revision addresses attacks like this by changing the ideal model. In the changed model, the ideal-model adversary is in charge of delivery of input messages forwarded by the dummy parties to the ideal functionality.

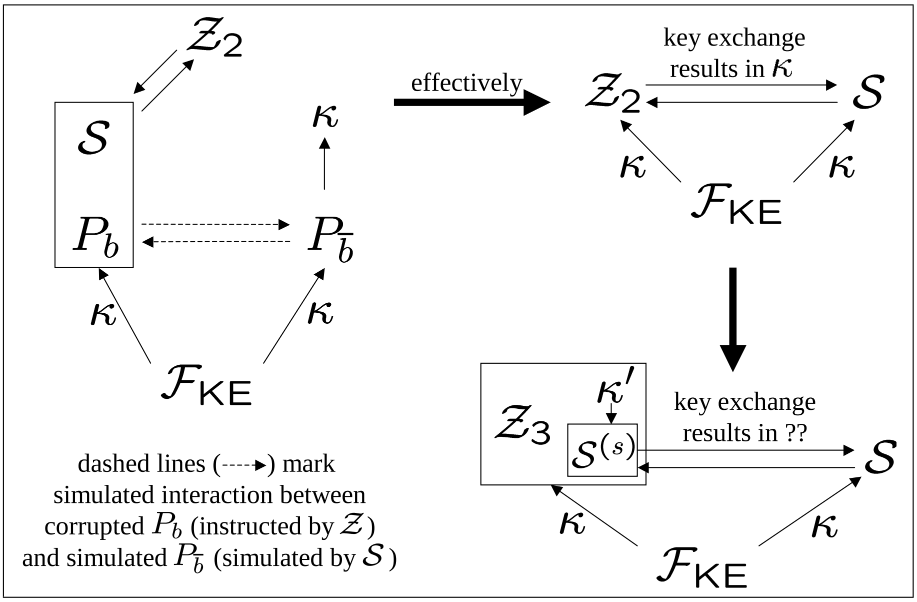
protocol run of π, the ideal-model adversary eventually delivers output to the parties (except in a negligible fraction of runs). A similar argument shows that π must guarantee matching keys (i. e., $\alpha = \beta$) in all but a negligible fraction of runs. To see this, we only need to modify $\mathcal{Z}_1$ in its fourth step, so that it outputs 1 exactly if $\alpha = \beta$. Now we are ready to formalize the attack discussed at the beginning of this section. Consider the following environment $\mathcal{Z}_2$, which also expects to communicate with the *dummy adversary* $\tilde{\mathcal{A}}$ in the real model:

1. Pick randomly $(b, \bar{b}) \in \{(i, j), (j, i)\}$.
2. Activate P_i with `(Establish-session,`sid`,`P_i`,`P_j`,initiator)`.
3. Activate P_j with `(Establish-session,`sid`,`P_j`,`P_i`,responder)`.
4. Instruct the adversary to corrupt P_b and to discard all messages possibly waiting to be delivered from P_b to $P_{\bar{b}}$.
5. Perform protocol π in the role of P_b, therefore send and receive messages through the corrupted "relay" P_b; let the adversary deliver all messages between P_b and $P_{\bar{b}}$.
6. Compare the output value of $P_{\bar{b}}$ with the local result of the key exchange protocol performed with $P_{\bar{b}}$ over P_b; if both match, output 1; otherwise output 0.

Now in the real model, the adversary $\tilde{\mathcal{A}}$ will follow precisely $\mathcal{Z}_2$'s instructions; consequently, a "normal" run of protocol π will take place between $P_{\bar{b}}$ (which expects to talk to P_b) and $\mathcal{Z}_2$. As $\alpha = \beta$ with overwhelming probability, the probability for $\mathcal{Z}_2$ to output 1 in the real model will be at most negligibly away from 1.

On the other hand, in the ideal model, the session key which will be output by the uncorrupted initiator $P_{\bar{b}}$ at the end of the simulated run of π (we'll call this key κ here) is fixed by $\mathcal{F}_{\mathrm{KE}}$ right after step 3, so at a time when neither initiator nor responder is corrupted. Consequently, κ is picked uniformly out of $\{0, 1\}^k$ by $\mathcal{F}_{\mathrm{KE}}$. (Of course, in step 4 the simulator is allowed to corrupt P_j and thereby may get to know κ, but it is not able to *influence* κ.) For mimicking the real model, $\mathcal{S}$ must now be able to convince $\mathcal{Z}_2$ that the session key explicitly negotiated in step 5 is exactly κ. In other words, either $\mathcal{Z}_2$ succeeds in distinguishing the real from the ideal model, or π offers the initiator as well as the responder the possibility of "provoking" any output value κ. In case $\mathcal{Z}_2$ is *not* a successful distinguisher, we will construct from $\mathcal{Z}_2$ an environment $\mathcal{Z}_3$ which *must* be successful in distinguishing real from ideal. The reasoning will be as follows: if $\mathcal{S}$ can provoke keys "at wish" both in the roles of initiator and responder, then there must be a contradiction when $\mathcal{S}$ performs a key exchange *with itself* (more precisely, with a simulation of itself). This situation is illustrated in Figure 2.

Specifically, consider an environment $\mathcal{Z}_3$, which is a modification of $\mathcal{Z}_2$. Namely, we modify $\mathcal{Z}_2$ only from the fifth step on, in which $\mathcal{Z}_2$ performs protocol π in the role of P_b with $P_{\bar{b}}$. Instead of playing the role of an "honest" P_b with uniformly selected random tape, $\mathcal{Z}_3$ internally keeps a simulation of a *complete* ideal model, including simulated dummy parties $P_1^{(s)}, \ldots, P_n^{(s)}$, a simulated ideal functionality $\mathcal{F}_{\mathrm{KE}}^{(s)}$, and a simulation $\mathcal{S}^{(s)}$ of the simulator $\mathcal{S}$ itself. However, the



$\mathcal{Z}_2$
$\mathcal{S}$
P_b
$P_{\bar{b}}$
$\mathcal{K}$
$\mathcal{F}_{\mathsf{KE}}$
effectively
key exchange results in $\mathcal{K}$
$\mathcal{Z}_3$
$\mathcal{K}'$
$\mathcal{S}^{(s)}$
key exchange results in ??
dashed lines (----▸) mark simulated interaction between corrupted P_b (instructed by $\mathcal{Z}$) and simulated $P_{\bar{b}}$ (simulated by $\mathcal{S}$)

Fig. 2. The environments $\mathcal{Z}_2$ and $\mathcal{Z}_3$ from the proof of Proposition 1 in the ideal model

role of the environment in $\mathcal{Z}_3$'s simulation is taken by a simulation $\mathcal{Z}_2^{(s)}$ of $\mathcal{Z}_2$ which in its first step selects b to be the $\bar{b}$ of $\mathcal{Z}_3$ and vice versa. (To avoid confusion, with b and $\bar{b}$, we mean in the following $\mathcal{Z}_3$'s choices of these variables.) The idea of this is to let $\mathcal{Z}_2^{(s)}$ corrupt $P_{\bar{b}}$ in the simulation and to let $\mathcal{S}^{(s)}$ perform a simulated run of π in the role of P_b with the non-simulated $P_{\bar{b}}$ (whose role is taken by $\mathcal{S}$ if we are in the ideal model). Therefore, all messages sent from $P_{\bar{b}}$ are forwarded to $P_b^{(s)}$ and vice versa. Finally, $\mathcal{Z}_3$ outputs 1 exactly if the local output of $P_{\bar{b}}$ matches that of $P_b^{(s)}$. (Again, if either of them does not generate output after $m(k)$ delivered messages, $\mathcal{Z}_3$ halts with output 0.)

In the real model, since we assumed $\mathcal{Z}_2$ not to be successful in distinguishing the real from the ideal model, $\mathcal{S}^{(s)}$ must be "successful" in performing a key exchange with a non-corrupted party $P_{\bar{b}}$ which yields as output exactly the key generated by the (simulated) ideal functionality $\mathcal{F}_{\mathrm{KE}}^{(s)}$. As in $\mathcal{Z}_3$'s simulation, the latter output is eventually delivered to $P_b^{(s)}$, $\mathcal{Z}_3$ will output 1 with overwhelming probability in the real model.

In the ideal model, either the protocol fails (i. e., either $\mathcal{S}$ or $\mathcal{S}^{(s)}$ does not deliver an output message from the ideal functionality to an uncorrupted party), or the local outputs of $P_b^{(s)}$ and $P_{\bar{b}}$ are *distinct* with overwhelming probability. (Note that $\mathcal{F}_{\mathrm{KE}}$ and the simulated $\mathcal{F}_{\mathrm{KE}}^{(s)}$ have independent random tapes from which they pick their respective output values.) In any case, $\mathcal{Z}_3$ outputs 0 in all but a negligible fraction of runs in the ideal model, thereby distinguishing the real from the ideal model. □

Functionality $\mathcal{F}_{\text{KE}}^{(i,j)}$

$\mathcal{F}_{\text{KE}}^{(i,j)}$ proceeds as follows, running on security parameter k, with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive values `(ready,`*sid*`)` from the parties P_i and P_j and from the adversary $\mathcal{S}$. When receiving `(ready,`*sid*`)` from either P_i or P_j, forward this message (including the sender identity) to $\mathcal{S}$.
2. After having received values `(ready,`*sid*`)` from P_i, P_j, and $\mathcal{S}$ (in any order), proceed as follows:
   (a) If both P_i and P_j are uncorrupted, choose $\kappa \overset{R}{\leftarrow} \{0,1\}^k$.
   (b) If at least one of the parties P_i and P_j is corrupted, send a message `(choose-key,`*sid*`)` to the adversary $\mathcal{S}$. Upon receiving an answer `(key,`*sid*`,`κ`)` from $\mathcal{S}$, extract the value of κ from it.

   Once κ is set, send `(key,`*sid*`,`κ`)` to P_i and P_j, and send `(key,`*sid*`)` to the adversary $\mathcal{S}$. Then halt.

Fig. 3. The modified key exchange functionality $\mathcal{F}_{\text{KE}}^{(i,j)}$

Now we present a family $\{\mathcal{F}_{\text{KE}}^{(i,j)}\}_{P_i,P_j}$ of functionalities intended to capture the requirements for key exchange[7]. More specifically, the functionality $\mathcal{F}_{\text{KE}}^{(i,j)}$ (presented in Figure 3) is aimed at modeling a key exchange between the parties P_i and P_j. This functionality is derived from the functionality $\mathcal{F}_{\text{KE}}$ from [9], yet differs from it in several important aspects, see the discussion above. In the case of authenticated communication, we will show two common protocols to be securely realizing our key exchange functionality. (For unauthenticated communication in the sense of [9], one can use, e. g., an existentially unforgeable signature scheme to implement authenticated links.) Therefore, we start with

Definition 1. *A protocol $\pi^{(i,j)}$, parametrized by indices of two parties P_i and P_j, will be called a* universally composable key exchange protocol*, provided that*

– *it has the same interface as $\mathcal{F}_{\text{KE}}^{(i,j)}$ (with respect to communication between $\mathcal{Z}$ and the parties),*
– *it involves communication only between P_i and P_j,*
– *once a party generates output, it immediately erases all internal information,*
– *when all messages between P_i and P_j are delivered, and neither P_i nor P_j gets corrupted, $\pi^{(i,j)}$ guarantees common output (i. e., matching keys) computationally indistinguishable from random k-bit strings, even when all the communication between P_i and P_j is made public,*
– *at the time the first party (either P_i or P_j) generates output, the other party has erased all protocol information other than the output unless it is corrupted; furthermore, at this point, the protocol involves only one more fixed "acknowledgment" message sent from the party which generated output to the one which did not yet do so.*

[7] Here and from now on, we assume pairs of parties over which families of functionalities or protocols are indexed *not* to be of the form (i,i), i. e., we assume the participating parties to be distinct.

Protocol $\text{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$

These are instructions for two parties P_i and P_j to carry out a key exchange. Prior to acting upon these instructions, each of the parties waits for an initial `(ready,`sid`)` input.

1. Dependent on the security parameter k, party P_i chooses a group $\langle g\rangle \in \mathcal{G}$ along with a generator g. Then P_i chooses $x \overset{R}{\leftarrow} \{1,\dots,|\langle g\rangle|-1\}$, calculates $\alpha = g^x$ and sends $(sid, \mathrm{D}(g), \alpha)$ to P_j, where $\mathrm{D}(g)$ is a description of $\langle g\rangle$ which also specifies the generator g.
2. Upon receiving from P_i a message $(sid, \mathrm{D}(g), \alpha)$ with $\mathrm{D}(g)$ being acceptable for the current security parameter, P_j chooses $y \overset{R}{\leftarrow} \{1,\dots,|\langle g\rangle|-1\}$ and a random index ν into the family $\mathcal{H}_{\langle g\rangle}$. Then P_j calculates $\beta = g^y$, $\gamma = \alpha^y$, and $\kappa = H_{\langle g\rangle,\nu}(\gamma)$, sends (sid, β, ν) to P_i, and erases all local information but κ.
3. Upon receipt of (sid, β, ν) from P_j, P_i calculates $\gamma = \beta^x$ and $\kappa = H_{\langle g\rangle,\nu}(\gamma)$, then erases all local information but κ and sends `(`sid`,done)` to P_j. Party P_i then outputs `(key,`sid`,`κ`)`, erases κ and halts.
4. Upon receipt of `(`sid`,done)` from P_i, party P_j outputs `(key,`sid`,`κ`)`, erases κ and halts.

Fig. 4. Protocol $\text{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$

It should be remarked that the last of the requirements in Definition 1 can be interpreted as a special case of the `ack` property defined in [9], whereas the requirement for keys indistinguishable from random k-bit strings can be seen as a variant of SK-security (see [9]).

Example 1. Protocol $\text{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$ (presented in Figure 4) is a variant of the common Diffie-Hellman key exchange protocol, derived from the protocol SIG-DH of [9]. At this $\mathcal{G} = \{G_\mu\}_\mu$ is a family of cyclic groups of prime order (explicitly given by a generator g) with hard decisional Diffie-Hellman (DDH) problem (cf., e. g., [5, Section 2]). For each group $\langle g\rangle \in \mathcal{G}$ we denote by $\mathcal{H}_{\langle g\rangle} = \{H_{\langle g\rangle,\nu}\}_\nu$ a family of pair-wise independent hash functions that is used to pass from group elements g^{xy} to bitstrings $H_{\langle g\rangle,\nu}(g^{xy})$ where $x, y \in \{1,\dots,|\langle g\rangle|-1\}$: as the ideal functionality $\mathcal{F}^{(i,j)}_{\mathrm{KE}}$ chooses the key as a random *bitstring* κ, we follow the approach in [21, Section 5.3.2] (see also [22]) and assume the parameters to be chosen such that the decisional Diffie-Hellman problem and the entropy smoothing theorem (cf., e. g., [19, Chapter 8]) imply the computational indistinguishability of the distributions $\{(g, g^x, g^y, \nu, H_{\langle g\rangle,\nu}(g^{xy}))\}_k$ and $\{(g, g^x, g^y, \nu, \kappa)\}_k$—with κ a random bitstring of length equal to the output length of $\mathcal{H}_{\langle g\rangle}$ and k the security parameter. We assume that for a group $G \in \mathcal{G}$ associated with security parameter k, the output length of $\mathcal{H}_G$ is exactly k.

While the protocol SIG-DH in [9] assumes that a suitable group description along with a group generator is provided to the protocol participants as 'initial information', in the protocol $\text{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$ in Figure 4 a description $\mathrm{D}(g)$ of a suitable group, including the specification of a generator g, is explicitly transmitted within the protocol. To avoid incorrect choices by a (corrupted) initiator of the key exchange, we assume that for given security parameter and description $\mathrm{D}(g)$

Protocol $\text{PKKE}_{\text{PK}}^{(i,j)}$

These are instructions for two parties P_i and P_j to carry out a key exchange. Prior to acting upon these instructions, each of the parties waits for an initial `(ready,`sid`)` input.

1. Party P_i generates a key pair (d, e) via $(d, e) \leftarrow \mathtt{K}(k)$ and sends the public key e in form of the message `(`sid`,`e`)` to P_j while locally storing the corresponding private key d.
2. Upon receiving a message `(`sid`,`e`)` from P_i, party P_j first chooses a random k-bit string κ, then computes κ's encryption with respect to the public key e via $c \leftarrow \mathtt{E}(e, \kappa)$, sends `(`$sid$`,`$c$`)` to P_i, and erases all local information except the key κ.
3. Upon receiving `(`sid`,`c`)` from P_j, party P_i computes the decryption κ of c via $\kappa \leftarrow \mathtt{D}(d, c)$, then erases all local information but κ, sends `(`sid`,done)` to party P_j, outputs `(key,`sid`,`κ`)`, erases κ and halts.
4. Upon receiving `(`sid`,done)` from P_i, party P_j outputs `(key,`sid`,`κ`)`, erases κ and halts.

Fig. 5. Protocol $\text{PKKE}_{\text{PK}}^{(i,j)}$

one can verify in strict polynomial time whether $\langle g \rangle \in \mathcal{G}$ holds and $\langle g \rangle$ is acceptable for the security parameter, thus implying difficulty of the DDH problem in $\langle g \rangle$[8]. Having in mind practical proposals like IKEv2 [18] or JFKi, JFKr [1] where the agreement on the specific group is a relevant issue, this slightly more complicated formulation seems acceptable. From the formal point of view, this modeling also avoids the set-up assumption that implicitly is made when using "globally available" parameters (which depend on the security parameter).

From the construction it is clear that protocol $\text{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ fulfills the requirements for a universally composable key exchange protocol—note here that the use of signed messages is not necessary, as we assume authenticated communication.

Example 2. As another example, take a look at protocol $\text{PKKE}_{\text{PK}}^{(i,j)}$ in Figure 5, where $\text{PK} = (\mathtt{K}, \mathtt{E}, \mathtt{D})$ is a semantically secure public-key encryption scheme (see [16]). By the semantic security of PK, an eavesdropped encryption of the secret key κ reveals no information about κ to a polynomially bounded adversary, and thus protocol $\text{PKKE}_{\text{PK}}^{(i,j)}$ satisfies all the requirements of Definition 1 and can be called a universally composable key exchange protocol.

Proposition 2. *Suppose we are in a model with authenticated links and trusted erasures. Assume further that for two fixed different parties P_i and P_j, protocol $\pi^{(i,j)}$ is a* universally composable key exchange protocol *as defined above. Then $\pi^{(i,j)}$ securely realizes functionality $\mathcal{F}_{\text{KE}}^{(i,j)}$ with respect to adaptive adversaries.*

Proof. The proof can be found in Appendix B. □

[8] As an example in which "wrong" parameter choices can be detected easily, $\langle g \rangle$ could be computed canonically from k, so there would be only *one* $\langle g \rangle \in \mathcal{G}$ associated with each k. Of course, hardness of the DDH problem then requires specific assumptions.

Functionality $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$

$\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ proceeds as follows, running on security parameter k, with parties $P_1, \dots, P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive values (`ready`,*sid*) from the parties P_i and P_j and from the adversary $\mathcal{S}$. When receiving (`ready`,*sid*) from either P_i or P_j, forward this message (including the sender identity) to $\mathcal{S}$.
2. After having received values (`ready`,*sid*) from P_i, P_j, and $\mathcal{S}$ (in any order), proceed as follows:
   (a) If P_j is *not* corrupted, choose κ uniformly from $\{0,1\}^k$.
   (b) If P_j *is* corrupted, send a message (`choose-key`,*sid*) to the adversary $\mathcal{S}$. Upon receiving an answer (`key`,*sid*,κ) from $\mathcal{S}$, extract the value of κ from it.

   Once κ is set, send (`key`,*sid*,κ) to P_i and P_j, and send (`key`,*sid*) to the adversary. Then halt.

Fig. 6. The key exchange functionality $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$

4 A Stronger Notion of Key Exchange

The description of $\mathcal{F}_{\mathrm{KE}}$ as well as the one of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ allows the adversary to freely *choose* the session key if at least one participating party is corrupted. This also holds for the 'relaxed' key exchange functionality $\mathcal{F}_{\mathrm{RKE}}^{\mathcal{N}}$ from [9], and one may ask whether this "worst case modeling" of corrupted parties is indeed justified. E. g., in the protocol $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$ the consequences of corrupting P_i and of corrupting P_j are intuitively quite different: a malicious party P_j has complete control over the resulting key κ, and it can, e. g., choose a value for κ that has been chosen earlier by some "outsider" P_a. If κ is later used to encrypt messages sent by P_i, then the "outsider" P_a will be able to read all these messages without any communication between P_j and P_a taking place during or after the key exchange of P_i and P_j. For doing so, P_a does not even have to eavesdrop the communication between P_i and P_j during the key exchange. On the other hand, a corrupted P_i is not able to influence an honest choice of κ performed by P_j.

In the Diffie-Hellman protocol $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ a similar "asymmetry" exists, but this property is not reflected in the definitions of the mentioned key exchange functionalities, either. As in some situations an additional security guarantee as provided by $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ and $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$ may be desirable, in the sequel we want to put this observation on firmer grounds. The functionality $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$, which we introduce for this purpose, is certainly not completely satisfactory from a conceptual point of view. Nevertheless, we think it gives ample evidence for the possibility to provide non-trivial (formal) security guarantees even if a participant in a key exchange is corrupted: A natural modification of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ might be the one presented in Figure 6. The functionality $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ guarantees random keys even when P_i gets corrupted. However, as soon as P_j is corrupted, the adversary may freely determine the common key κ as with $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$.

Protocol $\textsc{pad}^{(i,j)}$

These are instructions for two parties P_i and P_j to carry out a key exchange. Prior to acting upon these instructions, each of the parties waits for an initial `(ready,`sid`)` input. Furthermore, the parties expect to be run in the $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$-hybrid model, i. e., with access to a polynomial number of instances of the ideal functionality $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$.

1. Immediately after having received the initial `(ready,`sid`)` message, P_i as well as P_j sends the message `(ready,0)` to the $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$-instance with session ID 0.
2. Then P_j, after having received the key $\bar{\kappa}$ from this instance of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, uniformly chooses $\psi \in \{0,1\}^k$ and calculates $\kappa = \psi \oplus \bar{\kappa}$. It then erases all local information but κ and sends `(`sid`,`ψ`)` to P_i.
3. Upon receiving a message `(`sid`,`ψ`)` and after having received a key $\bar{\kappa}$ from the $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$-instance with session ID 0, P_i first calculates $\kappa = \psi \oplus \bar{\kappa}$ and erases all local information but κ. Then P_i sends `(`sid`,done)` to P_j, outputs `(key,`sid`,`κ`)`, erases κ and halts.
4. Upon receipt of `(`sid`,done)` from P_i, party P_j outputs `(key,`sid`,`κ`)`, erases κ and halts.

Fig. 7. Protocol $\textsc{pad}^{(i,j)}$

Remark 4. Unfortunately, neither protocol $\textsc{dh}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ nor protocol $\textsc{pkke}_{\mathrm{PK}}^{(i,j)}$ securely realizes $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$. This holds also for the "side-reversed" versions $\textsc{dh}_{\mathcal{G},\mathcal{H}}^{(j,i)}$ and $\textsc{pkke}_{\mathrm{PK}}^{(j,i)}$. To see this, consider the following environment $\mathcal{Z}$, expecting to be run with the dummy adversary in the real model: $\mathcal{Z}$ first corrupts P_i and does everything P_i would do to carry out a key exchange with P_j, thereby communicating over the corrupted "relay party" P_i with P_j. When P_j finally outputs a key, $\mathcal{Z}$ checks if it is the same $\mathcal{Z}$ itself generated in its key exchange with P_j. If and only if this is the case, $\mathcal{Z}$ outputs 1. Furthermore, when P_j outputs no key at a time it should do in the real model or P_j sends messages of the wrong format or no messages at all, $\mathcal{Z}$ outputs 0.

For a very brief analysis, first note that by construction of the protocols in question, $\mathcal{Z}$ always outputs 1 in the real model. On the other hand, in the ideal model with functionality $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$, regardless of the simulator $\mathcal{S}$ and the messages simulated between P_i and P_j, P_j's output is chosen uniformly from $\{0,1\}^k$ since only P_i, but not P_j is corrupted. So P_j outputs the key $\mathcal{Z}$ locally generated in its key exchange only in a negligible fraction of runs and thus, $\mathcal{Z}$ outputs 0 in the ideal model with overwhelming probability. Hence $\mathcal{Z}$ serves as a distinguisher between any of the abovementioned protocols and $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$, as stated in Figure 6. In the next section a relaxation of $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ is given, which yields a "stronger" security notion than $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ but still is securely realized by $\textsc{dh}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ and $\textsc{pkke}_{\mathrm{PK}}^{(i,j)}$.

Consider protocol $\textsc{pad}^{(i,j)}$ given in Figure 7. As it makes use of exactly one instance of the ideal functionality $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, it can be seen as an extension to any protocol intended to realize $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$. In the next proposition, we will show $\textsc{pad}^{(i,j)}$

to be securely realizing $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$. By the composition theorem of [6], this means that for any protocol π which securely realizes $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, the extension $\mathrm{PAD}_{\pi}^{(i,j)}$ (which is essentially protocol $\mathrm{PAD}^{(i,j)}$, but canonically uses instances of π as subprotocols instead of talking to instances of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$) is guaranteed to still realize $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ securely. So we have the interesting situation that neither $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ nor $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$ securely realizes $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ "by itself", but already simple refinements of these protocols do so. Namely, since both of them securely realize $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, their extensions $\mathrm{PAD}_{\pi}^{(i,j)}$, with π taken as one of them, securely realize $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$.

Proposition 3. *Protocol* $\mathrm{PAD}^{(i,j)}$ *securely realizes* $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ *in the* $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$*-hybrid model with respect to adaptive adversaries.*

Proof. The proof can be found in Appendix B. □

In principle, the protocol $\mathrm{PAD}^{(i,j)}$ is sufficient for securely realizing $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ on the basis of a Diffie-Hellman-like key exchange like $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$. However, the additional communication introduced by protocol $\mathrm{PAD}^{(i,j)}$ might seem superfluous when dealing with a protocol like $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$, which intuitively provides the desired additional security guarantee "by itself". In Appendix A, we show how to catch this intuition by means of a suitable ideal functionality.

5 Conclusions

The above discussion shows that a universally composable notion of key exchange, as expressed through the functionality $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, can be realized through quite natural key exchange protocols. However, additional security guarantees which are provided, e. g., by the described Diffie-Hellman based realization of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ are not reflected by functionalities like $\mathcal{F}_{\mathrm{KE}}$, $\mathcal{F}_{\mathrm{RKE}}^{\mathcal{N}}$, or $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, and we have shown that at least a part of these additional qualities can be captured by an appropriately "strengthened" functionality that makes use of a non-information oracle.

Nevertheless, for all notions of key exchange discussed above, the adversary has complete control over the result of the key exchange, if the "wrong" party is corrupted, and it seems that a Diffie-Hellman-like key exchange protocol can allow for a stronger guarantee. E. g., for appropriate families $\mathcal{H}_{\langle g\rangle}$ the following variation of $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ seems to limit the possibilities of a corrupted P_j somewhat more: let P_j choose and send the index ν into the family $\mathcal{H}_{\langle g\rangle}$ directly after P_i has fixed the group description $\mathrm{D}(g)$ (and before $\alpha = g^x$ is received). In this protocol it is not obvious how P_j could force a specific key—even if the index ν is not chosen at random.

It remains an interesting open question if such additional guarantees of certain key exchange protocols can be captured through an appropriate ideal functionality in the framework of universal composition.

Acknowledgments

We thank Ran Canetti, Dieter Gollmann, María Isabel González Vasco and the anonymous referees for helpful discussions and comments.

References

1. William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D. Keromytis, and Omer Reingold. Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 48–58. ACM Press, 2002.
2. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A Universally Composable Cryptographic Library. Cryptology ePrint Archive, Report 2003/015, January 2003. `http://eprint.iacr.org/2003/015/`.
3. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 419–428. ACM Press, 1998. Full version at `http://eprint.iacr.org/1998/009`.
4. Mihir Bellare and Phillip Rogaway. Provably Secure Session Key Distribution: the Three Party Case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 57–66. ACM Press, 1995.
5. Dan Boneh. The Decision Diffie-Hellman Problem. In Joe P. Buhler, editor, *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.
6. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version at `http://eprint.iacr.org/2000/067`.
7. Ran Canetti and Marc Fischlin. Universally Composable Commitments. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001. Full version at `http://eprint.iacr.org/2001/055`.
8. Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Proceedings*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001. Full version at `http://eprint.iacr.org/2002/047`.
9. Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*. Springer, 2002. All citations refer to the full version at `http://eprint.iacr.org/2002/059`.
10. Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-up Assumptions. In *Proceedings of EUROCRYPT 2003*, Lecture Notes in Computer Science. Springer, 2003. To appear.

11. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally Composable Two-Party and Multi-party Secure Computation. In *Proceedings on Thirty-Fourth Annual ACM Symposium on Theory of Computing, STOC 2002*, pages 494–503. ACM Press, 2002.
12. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally Composable Two-Party and Multi-party Secure Computation, July 2003. Full (and revised) version of [11], available at `http://eprint.iacr.org/2002/140`.
13. Ivan B. Damgård. Presentation of [14] at CRYPTO 2002, 2002.
14. Ivan B. Damgård and Jesper B. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002.
Full version at `http://eprint.iacr.org/2001/091`.
15. Danny Dolev and Andrew C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.
16. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Science*, 28, 1984.
17. Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. Cryptology ePrint Archive, Report 2003/024, February 2003. `http://eprint.iacr.org/2003/024`.
18. Internet Key Exchange (IKEv2) Protocol. Charlie Kaufman, editor. IPSEC Working Group INTERNET-DRAFT draft-ietf-ipsec-ikev2-06.txt, March 2003. Available at
`http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-06.txt`.
19. Michael Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes. Princeton University Press, 1996.
20. Birgit Pfitzmann and Michael Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Research in Security and Privacy*, pages 184–200. IEEE Computer Society Press, 2001. Full version at `http://eprint.iacr.org/2000/066`.
21. Victor Shoup. On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012, 1999. `http://eprint.iacr.org/1999/012`.
22. Michael Steiner. *Secure Group Key Agreement*. PhD thesis, Universität des Saarlandes, 2002. Online available at
`http://www.semper.org/sirene/publ/Stei_02.thesis-final.pdf`.

A An Additional Security Guarantee of $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ and $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$

As explained in Remark 4, we cannot hope that $\mathrm{PKKE}_{\mathrm{PK}}^{(i,j)}$ or $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ provide a secure realization of $\mathcal{F}_{\mathrm{KE}+}^{(i,j)}$. To show that these protocols do in fact guarantee strictly more than needed for realizing $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$, we utilize so-called *non-information oracles*, a tool that has been introduced in [9]:

Definition 2. *Let $\mathcal{N}$ be an ITM with strict polynomial running time. Then $\mathcal{N}$ is a* non-information oracle *if no ITM $\mathcal{M}$, having interacted with $\mathcal{N}$ on secu-*

rity parameter k, can distinguish with non-negligible probability between the local output of $\mathcal{N}$ and a value drawn uniformly from $\{0,1\}^k$.

Example 3. In the proof of the next proposition, we will make use of the following ITM $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ (with $\mathcal{G},\mathcal{H}$ as in Example 1): when activated for the first time, $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ randomly chooses $\langle\bar{g}\rangle \in \mathcal{G}$ (in dependence of the security parameter), two values $\bar{x},\bar{y} \in \{1,\ldots,|\langle\bar{g}\rangle|-1\}$, and a random index $\bar{\nu}$ into the family $\mathcal{H}_{\langle\bar{g}\rangle}$. Then $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ sends a description $\mathrm{D}(\bar{g})$ (as in Example 1) as well as $\bar{\alpha} = \bar{g}^{\bar{x}}$, $\bar{\beta} = \bar{g}^{\bar{y}}$, and $\bar{\nu}$ to the ITM it interacts with. After this, when receiving a message `accept`, it locally outputs $H_{\langle\bar{g}\rangle,\bar{\nu}}(\bar{g}^{\bar{x}\bar{y}})$ and halts. On the other hand, upon receiving a value `reject`, $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ sends $\bar{x}$ and $\bar{y}$ to the ITM it interacts with and waits to receive a pair $(\mathrm{D}(g'),\alpha)$ with $\mathrm{D}(g')$ describing a $\langle g'\rangle \in \mathcal{G}$ that is acceptable for the current security parameter and $\alpha \in \langle g'\rangle \setminus \{1\}$. Then $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ uniformly selects $r \in \{1,\ldots,|\langle g'\rangle|-1\}$ and an index ν' into the family $\mathcal{H}_{\langle g'\rangle}$, locally outputs $H_{\langle g'\rangle,\nu'}(\alpha^r)$, and halts.

We claim that under the decisional Diffie-Hellman assumption, $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ is a non-information oracle. To show this, assume that there is an ITM $\mathcal{M}$ that, after running with $\mathcal{N}_{\mathcal{G},\mathcal{H}}$, successfully distinguishes (i. e., differs in its output distribution) the local output of $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ from a random k-bit string. When we modify $\mathcal{M}$ to never issue a `reject` message (possibly followed by some $(\mathrm{D}(g'),\alpha)$), but instead to send an `accept` message and to halt with random output without even looking at the challenge, this cannot downgrade $\mathcal{M}$'s advantage in distinguishing. This is so since in case of a `reject` message, followed by some $(\mathrm{D}(g'),\alpha)$ with $\alpha \in \langle g'\rangle \setminus \{1\}$, $\mathcal{N}_{\mathcal{G},\mathcal{H}}$ outputs the hash value of a uniformly selected element from $\langle g'\rangle \setminus \{1\}$ (for $\langle g'\rangle$ is of prime order and therefore $\langle g'\rangle = \langle\alpha\rangle$), this group element about which $\mathcal{M}$ has no information whatsoever.

Thus $\mathcal{M}$ is able to distinguish random k-bit strings from the hash values of group elements $\bar{g}^{\bar{x}\bar{y}} \in \langle\bar{g}\rangle \setminus \{1\}$. By the universal hash property of $\mathcal{H}_{\langle\bar{g}\rangle}$, this means that $\mathcal{M}$ can also distinguish triples $(\bar{g}^{\bar{x}},\bar{g}^{\bar{y}},\bar{g}^{\bar{x}\bar{y}})$ from triples $(\bar{g}^{\bar{x}},\bar{g}^{\bar{y}},\bar{g}^{r})$, hence contradicting the decisional Diffie-Hellman assumption[9].

Now we are ready to give the definition of a key exchange functionality which, on the one hand, guarantees "essentially" random keys which are not predictable or influencable by the adversary even when one party is corrupted. Yet, on the other hand, this functionality is securely realized by both $\mathrm{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$ and $\mathrm{PKKE}^{(i,j)}_{\mathrm{PK}}$. (As with $\mathcal{F}^{(i,j)}_{\mathrm{KE+}}$, the party which "may" safely be corrupted without losing the feature of random keys needs to be fixed in advance.) More specifically, consider the family $\{\mathcal{F}^{\mathcal{N},(i,j)}_{\mathrm{KE+}}\}_{\mathcal{N},P_i,P_j}$, parametrized by a non-information oracle $\mathcal{N}$ and the indices of two parties P_i and P_j, specified in Figure 8.

Proposition 4. *Presuming authenticated links and trusted erasures, protocol $\mathrm{DH}^{(i,j)}_{\mathcal{G},\mathcal{H}}$ securely realizes functionality $\mathcal{F}^{\mathcal{N},(i,j)}_{\mathrm{KE+}}$ with respect to adaptive adversaries for an appropriate non-information oracle $\mathcal{N} = \mathcal{N}_{\mathcal{G},\mathcal{H}}$ under the decisional Diffie-Hellman assumption.*

[9] Formally, $\mathcal{M}$ only distinguishes triples $(\bar{g}^{\bar{x}},\bar{g}^{\bar{y}},\bar{g}^{\bar{x}\bar{y}})$ from triples $(\bar{g}^{\bar{x}},\bar{g}^{\bar{y}},\bar{g}^{r})$ both subject to the condition $\bar{x},\bar{y},r \neq 0$. Yet when choosing $\bar{x}$, $\bar{y}$, and r at random, this happens only in a negligible number of cases, and thus can be neglected.

Functionality $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$

$\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$ proceeds as follows, running on security parameter k, with parties $P_1, \dots, P_n$ and an adversary $\mathcal{S}$.

1. Invoke a new copy of $\mathcal{N}$ with fresh random input. Allow $\mathcal{S}$ to interact with $\mathcal{N}$, i. e., to receive messages from and send messages to $\mathcal{N}$. If P_i or P_j gets corrupted at a time $\mathcal{N}$ already produced output, or $\mathcal{N}$ produces output at a time at least one of these parties is corrupted, send the complete state of $\mathcal{N}$ (including randomness and in particular the local output) to the adversary. (Of course, after step 3, $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$ has halted and this is no longer possible.)
2. Wait to receive values `(ready,`sid`)` from the parties P_i and P_j and from the adversary $\mathcal{S}$. When receiving `(ready,`sid`)` from either P_i or P_j, forward this message (including the sender identity) to $\mathcal{S}$.
3. After having received values `(ready,`sid`)` from P_i, P_j, and $\mathcal{S}$ (in any order), and after $\mathcal{N}$ produced local output, proceed as follows:
   (a) If neither P_i nor P_j is corrupted, choose κ uniformly from $\{0,1\}^k$.
   (b) If only P_i is corrupted, but *not* P_j, set κ to the local output of $\mathcal{N}$.
   (c) If P_j *is* corrupted, send a message `(choose-key,`sid`)` to the adversary $\mathcal{S}$. Upon receiving an answer `(key,`sid`,`κ`)` from $\mathcal{S}$, extract κ from it.

   Once κ is set, send `(key,`sid`,`κ`)` to P_i and P_j, and send `(key,`sid`)` to the adversary. Then halt.

Fig. 8. The key exchange functionality $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$

Proof. As already mentioned above, $\mathcal{N} = \mathcal{N}_{\mathcal{G},\mathcal{H}}$ is the non-information oracle from Example 3. We now present a simulator $\mathcal{S}$ mimicking attacks carried out by the dummy adversary on protocol $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$. The idea behind the simulation is as follows: $\mathcal{S}$ is provided with a transcript of a Diffie-Hellman key exchange by $\mathcal{N}$. It then simulates messages from this transcript between P_i and P_j. When one of the parties is corrupted, $\mathcal{N}$ (resp., $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$) provides $\mathcal{S}$ with corresponding secret information consistent with the protocol transcript. Furthermore, if P_i is corrupted, $\mathcal{S}$ has the chance to perform the "second half" of a key exchange with $\mathcal{N}$ to provide $\mathcal{Z}$ with a consistent view of a party actually taking part in the key exchange. If P_j is corrupted, $\mathcal{S}$ may even pick the common key freely, reflecting that the message sent from P_j to P_i in the Diffie-Hellman key exchange fully determines this common key.

So $\mathcal{S}$ behaves exactly like the simulator $\mathcal{S}_{\pi}^{(i,j)}$ from the proof of Proposition 2 (with protocol $\mathrm{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ taken as π), with the following exceptions:

– When being supplied by $\mathcal{N}$ with $\mathrm{D}(\bar{g})$, $\bar{\alpha}$, $\bar{\beta}$, and $\bar{\nu}$, $\mathcal{S}$ stores these for future use. (Note that this happens at the first activation of the ideal functionality, so before the actual protocol simulation takes place.)

– When the simulated $P_i^{(s)}$ wishes to send a message $(sid, \mathrm{D}(g), \alpha)$ to $P_j^{(s)}$, and P_i is not corrupted, $\mathcal{S}$ simulates a message $(sid, \mathrm{D}(\bar{g}), \bar{\alpha})$ from P_i to P_j instead.

- When $P_j^{(s)}$ is delivered a message $(\mathit{sid}, \mathrm{D}(g), \alpha)$ with $\alpha \in \langle g \rangle \setminus \{1\}$, and P_j is uncorrupted, then $\mathcal{S}$ proceeds as follows: if P_i is uncorrupted (then we have $\alpha = \bar{\alpha}$ and $\mathrm{D}(g) = \mathrm{D}(\bar{g})$), $\mathcal{S}$ sends `accept` to $\mathcal{N}$ and simulates a message $(\mathit{sid}, \bar{\beta}, \bar{\nu})$ from P_j to P_i; if, on the other hand, P_i is corrupted, $\mathcal{S}$ sends $(\mathrm{D}(g), \alpha)$ to $\mathcal{N}$ and, when receiving $\mathcal{N}$'s complete state (which by definition happens immediately afterwards), simulates a message $(\mathit{sid}, g^r, \nu')$ from P_j to P_i, where r and ν' are extracted from $\mathcal{N}$'s state.
- When $\mathcal{S}$ is requested to corrupt P_i, then $P_i^{(s)}$'s internal state first has to be modified so as to match the simulated protocol execution. If P_i already erased internal data, only the common key may have to be acquired from either $\mathcal{N}$'s state (if so far, no uncorrupted party generated output) or from $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$ itself by corrupting the dummy party P_i and delivering $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$'s output message to P_i. If, on the other hand, $P_i^{(s)}$ already chose, but did not yet erase its secret exponent x, $\mathcal{S}$ has to obtain the corresponding secret exponent $\bar{x}$ from $\mathcal{N}$: if $P_j^{(s)}$ has not yet received its first message from $P_i^{(s)}$, $\mathcal{S}$ does this by sending `reject` to $\mathcal{N}$ (by definition, $\mathcal{N}$ immediately replies with $\bar{x}$ and $\bar{y}$); else, $\mathcal{S}$ has already sent `accept` to $\mathcal{N}$ and therefore gets to know $\mathcal{N}$'s complete state immediately. Now $\mathcal{S}$ modifies the internal state, which is sent to $\mathcal{Z}$ as that of P_i, to be consistent with the exponent $\bar{x}$ and the group description $\mathrm{D}(\bar{g})$.
- When $\mathcal{S}$ is requested to corrupt P_j at a time $\mathcal{S}$ has already simulated a message back from P_j to P_i, $\mathcal{S}$ may have to provide $\mathcal{Z}$ with a key consistent with the preceding protocol transcript. (Note that P_j erases all other secret information in the same activation it generates it, so $\mathcal{S}$ needs never provide such "temporary secrets".) If no uncorrupted party generated output so far, $\mathcal{S}$ can obtain this common key by extracting $\mathcal{N}$'s output from $\mathcal{N}$'s state, with which $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$ provides $\mathcal{S}$ upon corruption of P_j. However, if some party already generated output, the ideal functionality already halted and therefore, the common key has to be acquired by corrupting the dummy party P_j and delivering the output message from $\mathcal{F}_{\mathrm{KE+}}^{\mathcal{N},(i,j)}$ to P_j.

The analysis of $\mathcal{S}$ is very similar to that of $\mathcal{S}_\pi^{(i,j)}$ in the proof of Proposition 2; we therefore only treat the differences caused by the above modifications. First, note that the states of corrupted parties with which $\mathcal{S}$ supplies the environment machine are by construction always consistent with the messages sent by the respective party prior to its corruption. In particular, this holds although the messages simulated by $\mathcal{S}$ between P_i and P_j are in general not those sent between $P_i^{(s)}$ and $P_j^{(s)}$. Moreover, by construction, the common key agreed upon in the ideal model is consistent with the messages between P_i and P_j as long as at least one of them is corrupted: when P_i is corrupted *before* its first message to P_j is delivered, the common key is consistent with the sent messages since $\mathcal{N}$ then fixes the key according to the message actually sent to an uncorrupted P_j. Later corruptions of P_i do not influence the key. When P_j is corrupted, $\mathcal{S}$ may freely choose the key and can thereby guarantee consistency of the common key by itself exactly as $\mathcal{S}_\pi^{(i,j)}$. If, on the other hand, neither P_i nor P_j gets corrupted, then the common key is indistinguishable from a k-bit random string by the universal hash property of $\mathcal{H}_{\langle g \rangle}$ in combination with the decisional Diffie-Hellman assumption. Putting all this together, we can now apply the argumentation of the proof of Proposition 2. □

Proposition 5. *When supposing authenticated links and trusted erasures, protocol* $\text{PKKE}_{\text{PK}}^{(i,j)}$ *securely realizes functionality* $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(i,j)}$ *with respect to adaptive adversaries for an appropriate non-information oracle* $\mathcal{N} = \mathcal{N}_{\text{PK}}$ *once* PK *is a semantically secure public-key cryptosystem.*

Proof. The proof is very similar to the proof of Proposition 4, yet much easier, as by construction of $\text{PKKE}_{\text{PK}}^{(i,j)}$, P_j may choose the common key by itself. The non-information oracle $\mathcal{N} = \mathcal{N}_{\text{PK}}$ generates a key pair $(\bar{d}, \bar{e})$ via the key generation algorithm $\mathtt{K}$ (internally invoked on input k) and stores it. Then $\mathcal{N}$ chooses a random k-bit string $\bar{\kappa}$, encrypts it via $\bar{c} \leftarrow \mathtt{E}(\bar{e}, \bar{\kappa})$ and sends the public key $\bar{e}$ and the ciphertext $\bar{c}$ to the ITM it interacts with. It then locally outputs $\bar{\kappa}$ and halts. By the semantic security of PK, $\mathcal{N}$ has the non-information property.

We shortly describe the simulator $\mathcal{S}$ mimicking attacks carried out by the dummy adversary on $\text{PKKE}_{\text{PK}}^{(i,j)}$. In particular, $\mathcal{S}$ differs from the simulator $\mathcal{S}_{\pi}^{(i,j)}$ (with protocol $\text{PKKE}_{\text{PK}}^{(i,j)}$ taken as π) only as follows: when P_i is uncorrupted at that time, $\mathcal{S}$ replaces an initial message (sid, e) sent from $P_i^{(s)}$ to $P_j^{(s)}$ with a message $(\mathit{sid}, \bar{e})$, where the public key $\bar{e}$ together with a ciphertext $\bar{c}$ is obtained from $\mathcal{N}$ at the beginning of the protocol run. Consequently, a message (sid, c) sent back from $P_j^{(s)}$ (with uncorrupted P_j) to $P_i^{(s)}$ is replaced by a simulation of the message $(\mathit{sid}, \bar{c})$. Upon corruption requests of P_i (resp., P_j), $\mathcal{S}$ first modifies the internal data of $P_i^{(s)}$ (resp., $P_j^{(s)}$) to be consistent with $\bar{e}$, $\bar{d}$, and $\bar{\kappa}$ (resp., $\bar{e}$, $\bar{c}$, and $\bar{\kappa}$), where upon such a corruption, $\bar{d}$ and $\bar{\kappa}$ are extracted from $\mathcal{N}$'s state with which $\mathcal{S}$ is supplied. □

Remark 5. The requirement for trusted erasures might seem a bit hard. If one is willing to give up perfect forward secrecy, one can modify $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(i,j)}$ so as to be realizable by the non-erasing counterparts of the protocols $\text{DH}_{\mathcal{G},\mathcal{H}}^{(i,j)}$ and $\text{PKKE}_{\text{PK}}^{(i,j)}$. Namely, $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(i,j)}$ has to be modified to ask the non-information oracle $\mathcal{N}$ for a key even when neither P_i nor P_j is corrupted at the time the key is fixed. Furthermore, $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(i,j)}$ must not halt after having sent the key to P_i and P_j; instead, even upon *later* corruptions of P_i or P_j, it has to send the complete state of the (terminated) non-information oracle $\mathcal{N}$ to supply the adversary with the (in the real model non-erased) secret information of the simulated parties.

B Proof of Proposition 2

Proof. Consider the simulator $\mathcal{S}_{\pi}^{(i,j)}$ presented in Figure 9 mimicking attacks carried out by the dummy adversary $\tilde{\mathcal{A}}$ on protocol $\pi^{(i,j)}$.

Fix an environment $\mathcal{Z}$ trying to distinguish between execution of protocol $\pi^{(i,j)}$ together with the dummy adversary $\tilde{\mathcal{A}}$ and running with the ideal functionality $\mathcal{F}_{\text{KE}}^{(i,j)}$ and the simulator $\mathcal{S}_{\pi}^{(i,j)}$. As parties different from P_i and P_j are not involved at all in protocol $\pi^{(i,j)}$, without losing generality we may assume $\mathcal{Z}$ not to request corruptions of parties P_l with $l \notin \{i, j\}$.

In a first step, we consider a modified $\mathcal{Z}$ which we will call $\mathcal{Z}_1$ and which instead of corrupting P_i or P_j *before* any of them generated output, halts with a random bit as output. We claim that $\mathcal{Z}_1$ has exactly the same success probability in distinguishing real and ideal model as $\mathcal{Z}$ has. For proving this, note that $\mathcal{Z}$ has completely identical views in the real and the ideal model before either P_i or P_j generates output. But if either P_i or P_j gets corrupted *before* any of them generated output, by construction of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ and $\mathcal{S}_{\pi}^{(i,j)}$, the simulator may choose the output value for the respective other party and thereby simulate a protocol run exactly as in the real model. So as $\mathcal{Z}$ has completely identical views of the real and the ideal model in this case, it cannot do better than to toss a coin (thereby doing exactly what $\mathcal{Z}_1$ does). Hence, $\mathcal{Z}_1$ distinguishes real from ideal exactly as good as $\mathcal{Z}$ does. Moreover, for on the one hand, we assumed authenticated links (so the adversary is only allowed to block, but not to forge messages sent by uncorrupted parties), and on the other hand $\mathcal{Z}_1$ does not corrupt P_i or P_j while their protocol output is being fixed (i. e., before one of them actually outputs its key), $\mathcal{Z}_1$ is guaranteed matching outputs of P_i and P_j both in the real and the ideal model. (In the real model this is implied by assumption about $\pi^{(i,j)}$; in the ideal model it is clear by construction of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$.)

Consider an environment $\mathcal{Z}_2$ which internally simulates environment $\mathcal{Z}_1$ and outputs whatever $\mathcal{Z}_1$ outputs. All communication of $\mathcal{Z}_1$ with the parties and the adversary is relayed, with the following exception: if the simulated $\mathcal{Z}_1$ wishes to ask the adversary to corrupt a party P_l (where $l \in \{i, j\}$) *after* the first party generated output κ, then $\mathcal{Z}_2$ answers $\mathcal{Z}_1$'s request on its own. It presents $\mathcal{Z}_1$ with a state of P_l that contains as local information *only* the key which was output by the first party, *except* when P_l did already generate output. By assumption about protocol $\pi^{(i,j)}$ and the simulator $\mathcal{S}_{\pi}^{(i,j)}$, this is exactly what $\mathcal{Z}_1$ would have got both in the real and in the ideal model. From this point on, $\mathcal{Z}_2$ ignores all output possibly generated by the (actually uncorrupted) party P_l. Messages the adversary is asked to deliver in the name of P_l are ignored unless for acknowledgment messages sent to the respective other party; in this case, $\mathcal{Z}_2$ pretends to $\mathcal{Z}_1$ that the receiving party generated output κ. As by assumption about protocol $\pi^{(i,j)}$, both P_i and P_j have already fixed their output (in our case to κ), $\mathcal{Z}_1$ will have identical views in the simulation inside $\mathcal{Z}_2$ and running "live" with parties and an adversary. Then $\mathcal{Z}_2$ has still exactly the same advantage in distinguishing the real model from the ideal one as $\mathcal{Z}_1$ and therefore $\mathcal{Z}$ has, even though $\mathcal{Z}_2$ corrupts neither P_i nor P_j at any point in time.

We have just shown that we may restrict to environments not corrupting *any* party. For such an environment, $\mathcal{S}_{\pi}^{(i,j)}$ simulates the communication of a complete protocol run of $\pi^{(i,j)}$ in the ideal model exactly as it would happen in the real model. Hence the *only* difference between real and ideal model is the value κ output as a common key by both P_i and P_j. In the ideal model, this value is a random k-bit string which in general "does not fit" to the simulated protocol run of $\pi^{(i,j)}$; however as $\pi^{(i,j)}$ is a *universally composable key exchange protocol* and therefore has output computationally indistinguishable from random k-bit

The simulator $\mathcal{S}_\pi^{(i,j)}$

$\mathcal{S}_\pi^{(i,j)}$ internally keeps a simulation of two parties $P_i^{(s)}$ and $P_j^{(s)}$ running protocol $\pi^{(i,j)}$.

– **Communication with $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$:**
  - Upon receiving from $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ a forwarded message (`ready`,*sid*) with initial sender P_l (where $l \in \{i,j\}$), forward this message to $P_l^{(s)}$.
  - As soon as the first simulated party $P_l^{(s)}$, whose dummy counterpart P_l is *not* corrupted, produces output (which we will call κ here), store κ and send (`ready`,*sid*) to $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$.
  - Upon receiving (`choose-key`,*sid*) from $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ (which by construction of the ideal functionality can only happen when $\mathcal{S}_\pi^{(i,j)}$ signalized through its ready signal that it has output κ available), send (`key`,*sid*,κ) back to $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$.
  - Deliver output messages sent from the ideal functionality to either P_i or P_j exactly when the respective simulated party $P_l^{(s)}$ generated output in the simulation.

– **Communication with $\mathcal{Z}$:**
  - When being requested by $\mathcal{Z}$ to check for messages sent by parties, reply that no messages were sent, except in the following case: when the dummy party P_i is uncorrupted and the simulated party $P_i^{(s)}$ wants to send a message m to $P_j^{(s)}$, then, upon being asked by the environment $\mathcal{Z}$, claim that P_i wants to send m to P_j but do not yet deliver m in the simulation (similarly for messages being sent from $P_j^{(s)}$ to $P_i^{(s)}$ when P_j is uncorrupted).
  - When being asked by $\mathcal{Z}$ to deliver a message m to some uncorrupted party P_l, store this request for future use and additionally, if $l \in \{i,j\}$, deliver m to $P_l^{(s)}$ in the simulation.
  - When being told by $\mathcal{Z}$ to corrupt some party P_l, first corrupt the dummy party P_l; then prepare state information for P_l taking into account all messages $\mathcal{S}_\pi^{(i,j)}$ was asked to deliver to P_l. Additionally, if $l \in \{i,j\}$ and P_l has not done so before, let P_l send a ready signal to the ideal functionality as soon as the corresponding session ID *sid* becomes available to $\mathcal{S}$. Moreover, if $l \in \{i,j\}$, proceed as follows:
    (a) If $\mathcal{S}_\pi^{(i,j)}$ has not yet sent its ready signal to the ideal functionality (i. e., if no simulated party $P_t^{(s)}$, whose counterpart P_t is uncorrupted, produced output), then deliver $P_l^{(s)}$'s internal state as P_l's internal state.
    (b) If $\mathcal{S}_\pi^{(i,j)}$ sent its ready signal and therefore, the ideal functionality sent output κ to both P_i and P_j, then P_l's state must be prepared to include κ *if and only if* it was not output by P_l. Then, κ can be obtained by $\mathcal{S}_\pi^{(i,j)}$ through delivering κ from the ideal functionality to P_l. So in any case, prepare state information for P_l consistent with κ. (This is possible by assumption about protocol $\pi^{(i,j)}$.)

Fig. 9. The simulator $\mathcal{S}_\pi^{(i,j)}$

strings, there can be no environment $\mathcal{Z}$ distinguishing the real from the ideal model. $\square$

Proof (of Proposition 3). For any hybrid-model adversary $\mathcal{H}$, we describe a simulator $\mathcal{S} = \mathcal{S}_{\mathcal{H}}$ mimicking attacks carried out by $\mathcal{H}$ in the hybrid model. $\mathcal{S}$ internally runs a simulation of a complete run of $\textsc{pad}^{(i,j)}$ in the hybrid model, including parties $P_1^{(s)}, \ldots, P_n^{(s)}$, the adversary $\mathcal{H}$, and (as needed) instances of the ideal functionality $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$. Yet all communication of the simulated $\mathcal{H}$ with the environment is forwarded to the (non-simulated) environment $\mathcal{Z}$ with which $\mathcal{S}$ is to interact. That means, incoming messages from $\mathcal{Z}$ are forwarded to $\mathcal{H}$ and vice versa.

The idea is to give $\mathcal{Z}$ a complete view of a hybrid-model run with adversary $\mathcal{H}$. By construction, the described simulation already does this job well, with two exceptions: by definition, $\mathcal{Z}$ is informed about corruptions as they take place. The solution to this issue is easy: every time $\mathcal{H}$ corrupts a party $P_l^{(s)}$ in the simulation, $\mathcal{S}$ first corrupts the corresponding party P_l in the ideal model. The state of P_l with which $\mathcal{Z}$ possibly expects to be supplied is then delivered by the simulated $\mathcal{H}$. Furthermore, if P_l has not yet sent its ready signal to the ideal functionality, $\mathcal{S}$ lets P_l do that as soon as the corresponding session ID *sid* becomes available to $\mathcal{S}$. (This is to allow the ideal functionality to generate output even in face of corrupted parties which did not yet send a ready signal.)

The other thing which has to be taken care of is the communication of $\mathcal{Z}$ with the parties P_l in the ideal model. The messages sent (as input) to and received (as output) from these parties by $\mathcal{Z}$ have to be consistent with the view of the hybrid-model execution by $\mathcal{Z}$. This is a little more involved and we describe our solution in detail.

In protocol $\textsc{pad}^{(i,j)}$, only the respective first message of the form $(sid,\mathtt{ready})$ to P_i or P_j has some effect, all other messages are ignored. Yet by construction of $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$, $\mathcal{S}$ is informed about exactly these inputs as they arrive. So when $\mathcal{S}$ is informed about a message $(sid,\mathtt{ready})$ which P_l ($l \in \{i,j\}$) got as input, it immediately forwards this as input to the simulated party $P_l^{(s)}$.

It remains to take care of the *output* behavior of the parties. We describe the necessary modifications:

– When the simulated party $P_j^{(s)}$ sends a message (sid,ψ) to $P_i^{(s)}$ at a time $P_i^{(s)}$, but *not* $P_j^{(s)}$ is corrupted, then $\mathcal{S}$ temporarily stops the simulation, sends $(sid,\mathtt{ready})$ to $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ and delivers to P_i (i. e., to itself, since P_i is corrupted) the key κ which is sent from $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ to P_i. It then sets $\bar{\psi} = \kappa \oplus \bar{\kappa}$ (where $\bar{\kappa}$ is the key $P_j^{(s)}$ received from $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$) and modifies the internal state of $P_j^{(s)}$ so to hold only the secret key κ from $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ instead of $\bar{\kappa} \oplus \psi$.
Consequently, the corresponding message (sid,ψ) is then altered to read $(sid,\bar{\psi})$ and the simulation is continued. Note that $\bar{\psi}$ as a random variable has uniform distribution over $\{0,1\}^k$ because κ has and is, as a random variable, independent of $\bar{\kappa}$ (even when $\mathcal{H}$ chooses $\bar{\kappa}$); that means that the pad $\bar{\psi}$ "looks" exactly as if generated by $P_j^{(s)}$ itself and thus this does not disturb the authenticity of the hybrid-model simulation.

- When an uncorrupted simulated party (say, $P_l^{(s)}$, where $l \in \{i, j\}$) generates output, then $\mathcal{S}$ has to deliver the corresponding output message containing the common key from $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ to P_l. If the key has not yet been determined, $\mathcal{S}$ first sends $(sid,\texttt{ready})$ to $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$. (When P_j is corrupted at that time, $\mathcal{S}$ is additionally asked for the common key. It then replies with the key $P_l^{(s)}$ produced.)
- The one case in which the output of the simulated $\textsc{pad}^{(i,j)}$ still differs from that in the ideal model is the case in which at the time the message (sid,ψ) is delivered from $P_j^{(s)}$ to $P_i^{(s)}$, neither of them is corrupted.
  This is no problem when nobody gets corrupted later, since then $\mathcal{H}$ has no information about the key agreed upon in the simulation. On the other hand, we have to take that into consideration upon later corruptions of $P_i^{(s)}$ or $P_j^{(s)}$. Namely, if in the situation in question a later corruption of $P_i^{(s)}$ is requested by $\mathcal{H}$, and the message just mentioned is *not* yet delivered, the key $\bar{\kappa}$ delivered from an instance of $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$ has to be replaced by $\kappa \oplus \psi$ in $P_i^{(s)}$'s memory; here, κ denotes the key generated by $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$ in the ideal model. Upon a later corruption of $P_i^{(s)}$ or one of $P_j^{(s)}$, we only have to replace the key they locally hold as output by κ. (Note that κ is accessible to $\mathcal{S}$ upon corruption of the corresponding dummy party P_l if P_l did not yet generate output.)
  Again, if the key κ generated in the ideal model is not yet determined, $\mathcal{S}$ first corrupts P_l (where $P_l^{(s)}$ is the respective party to be corrupted in the simulation), then sends $(sid,\texttt{ready})$ to $\mathcal{F}_{\mathrm{KE+}}^{(i,j)}$, possibly chooses the common key (in that case $\mathcal{S}$ chooses it at random), and delivers to P_l (i. e., to itself) the common key just generated. Furthermore, since no information about the key $\bar{\kappa}$ is revealed to $\mathcal{H}$ when neither $P_i^{(s)}$ nor $P_j^{(s)}$ is corrupted, the above modifications still yield a "valid" view of a protocol execution in the hybrid model to $\mathcal{H}$ and therefore to $\mathcal{Z}$. (Note that the value $\kappa \oplus \psi$ replaced for $\mathcal{F}_{\mathrm{KE}}^{(i,j)}$'s output in the hybrid model has uniform output distribution, as in our case, ψ is picked uniformly and in particular independent of κ, for P_j is by assumption not corrupted when picking ψ.)

By the above discussion, in any case, $\mathcal{S}$ provides $\mathcal{Z}$ with an "authentic-looking", complete view of the hybrid model. □

Multi-round Secure Light-Weight Broadcast Exclusion Protocol with Pre-processing

Yuji Watanabe and Masayuki Numao

IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi,
Kanagawa, 242-8502, Japan
{muew,numao}@jp.ibm.com

Abstract. A broadcast exclusion protocol allows a broadcaster to transmit a encrypted message to a set of n users over a broadcast channel so that all but some specified small group of k excluded users can decrypt the message, even if these excluded users collude with each other in an arbitrary manner. Recently, Matsuzaki et al. pointed out a potential problem in the earlier works regarding the number of modular exponentiation, and proposed an extended scheme in which decryption requires only two modular exponentiations regardless of n and k. However, our analysis shows this scheme has a limitation of the number of rounds.
The contribution of this paper is to present a new broadcast exclusion protocol maintaining security within a virtually unlimited number of rounds without spoiling the efficiency. First, we demonstrate a limitation of the rounds of the previous work by showing how a user can derive the system secret parameters after more than a certain number of rounds. Then, we present a new protocol for which we can provide rigorous security proof under the Computational Diffie-Hellman (CDH) assumption.
We note that even if we point out some limitation of the previous work, we still consider it nevertheless significant. In particular, we derived our new protocol by modifying some of their fundamental techniques.

Keywords: broadcast encryption, broadcast exclusion problem, pre-processing

1 Introduction

1.1 Background

Consider a situation where a large amount of data is distributed to all authorized users over a broadcast channel. Typically, a broadcaster provides each authorized user with a private decryption key, then the data is broadcasted in an encrypted form. Finally, the authorized users are able to decrypt the data and obtain the service they intended to get. This scenario can come up in the context of Cable, pay-TV, Internet multicasts, satellite, and group telecommunications.

In the above situation, it is required for the broadcaster to share a key (which is called the *broadcast key* in this paper) with all the non-excluded users quickly and securely. A broadcast encryption, introduced by Fiat and Naor [1], allows a broadcaster to distribute the broadcast key simultaneously and securely over a broadcast channel to selected subsets of users. A number of derivative works have been done mainly toward minimizing the communication overhead and the storage requirements for each user.

The problem of transmitting a message only to some small specified subset of the users has been considered in [1]. On the other hand, the broadcast exclusion problem, which is a kind of broadcast encryption, is the problem of transmitting a message over a broadcast channel shared by a number n of users so that all but some specified small coalition of k excluded users can decrypt the message, even if these excluded users collude with each other in an arbitrary manner. This paper addresses the broadcast exclusion protocol, in which a broadcast key is transmitted to each user in an encrypted form (we call this ciphertext "*header*") over a broadcast channel so that all but some specified subset of users can get the broadcast key[1].

If the private key of a user is exposed, for example, the user must be excluded as quickly as possible in order to prevent an eavesdropper from accessing the broadcasted secret information. The broadcast exclusion protocol allows the broadcaster to exclude the user from the recipients. Trivially, the broadcast exclusion protocol can be realized by a broadcaster sending a new broadcast key to each user except for the excluded one. This is optimal regarding the user's storage size, while the communication overhead is $O(n)$, and thus it takes a long time to send the broadcast key if n is large.

Several results on the broadcast exclusion problem have been proposed so for. Kumar, Rajagopalan and Sahai [4] proposed a broadcast exclusion protocol based on an error correcting code without any computational assumption. In their scheme, the communication overhead is $O(k^2)$ regardless of n, while the size of the private keys stored by each user is $O(k \log n)$, still depending on n.

Another method proposed by Anzai, Matsuzaki and Matsumoto [5] employs threshold cryptosystems [6] in order to avoid any dependence on n for the size of the private keys. A similar result was discovered independently by Naor and Pinkas [7] (In addition, their work includes the tracing and self enforcement extensions). The underlying idea of their schemes is as follows. A broadcaster divides a secret s into $n+k$ shares with a technique for a $(k+1, n+k)$-threshold scheme, and distributes a shares to each of the n users in a secure way. When the broadcaster excludes d users, k shares including d shares of the excluded users are broadcasted to all of the users. A non-excluded user can recover the broadcast key using $k+1$ shares, i.e., his share and k broadcasted shares, while the excluded users cannot do so, since the broadcasted shares contain his own share, and thus he can only obtain k shares. In their schemes, the communication overhead is $O(k)$ regardless of n, and the size of the private key that each user stores is $O(1)$. Their schemes also work in a public-key setting in the sense that there is no requirement for a fixed-privileged broadcaster. Moreover, the schemes are used

[1] Our protocol assumes the user is *stateful* in the sense that the users update their state from round to round. The stateless cases have been studied in [2], [3] etc.

multiple rounds, i.e., a single initialization enables multiple rounds of execution of broadcast exclusion among a set of users. Their schemes, however, require each user to compute the broadcast key with $O(k)$ modular exponentiations. The parameter k should be enough large to withstand the collusion, while setting a large value for k results in computing a large number of modular exponentiations, which takes a long time especially when the user's computing device has low computational power, such as a pervasive device.

The recent result in [8] (by the same authors as [5]) proposed an innovative technique to extend the scheme of [5] in such a way that each user computes the broadcast key with just two modular exponentiations, regardless of n and k. The transmission overhead is $O(k)$ and the size of the private key each user stores is $O(1)$, thus there are no changes in those orders from [5]. As an additional assumption, this scheme requires a fixed-privileged broadcaster which knows all the secret parameters and determines the excluded users. This is considered to be reasonable in some actual applications.

However, our analysis shows this scheme [8] has a limitation of the number of rounds. Actually, a user who observes the communications from the broadcaster for more than a certain number of rounds can calculate the secret parameters and defeat the subsequent exclusion. It is desirable to avoid such limitation of the number of rounds without spoiling the efficiency of computing the broadcast key.

1.2 Our Contribution

Computing modular exponentiation comprises the major portion of the computation for updating the broadcast key. For example, modular exponentiation requires t squarings and $t/2$ multiplications by using the general square-and-multiply algorithm, where the exponent can be expressed by t bits. Thus, reducing the number of modular exponentiations makes a substantial contribution to lowering the workload of broadcast key computations, which allows for the quick initiation of a new round.

It is required in [5] to compute the product of $k+1$ exponentials with distinct bases and distinct exponents. The authors of [7] mentioned that in this situation, a simultaneous multiple exponentiations algorithm can be used to reduce the computation overhead, where it takes k squarings and $(2^{k+1} - 2) + t - 1$ multiplications with $O(2^k)$ pre-computed values. However, this is effective only for small k. If k is large, e.g., $k = 1000$, the pre-computed values and run-time multiplications cannot be processed in a practical timeframe.

The computational overhead to derive the broadcast key is divided into two parts, the pre-processing and the real-time processing, which means a part of the computation before and after receiving the header, respectively.. The pre-processing can be performed during the previous round, while the real-time processing should be done within a short period of time just after receiving the header. Thus, the real-time processing would be the most significant part for rapid initiation of the new round. Meanwhile, we assume that it is not significant if pre-processing takes a long time to compute. This is reasonable for many

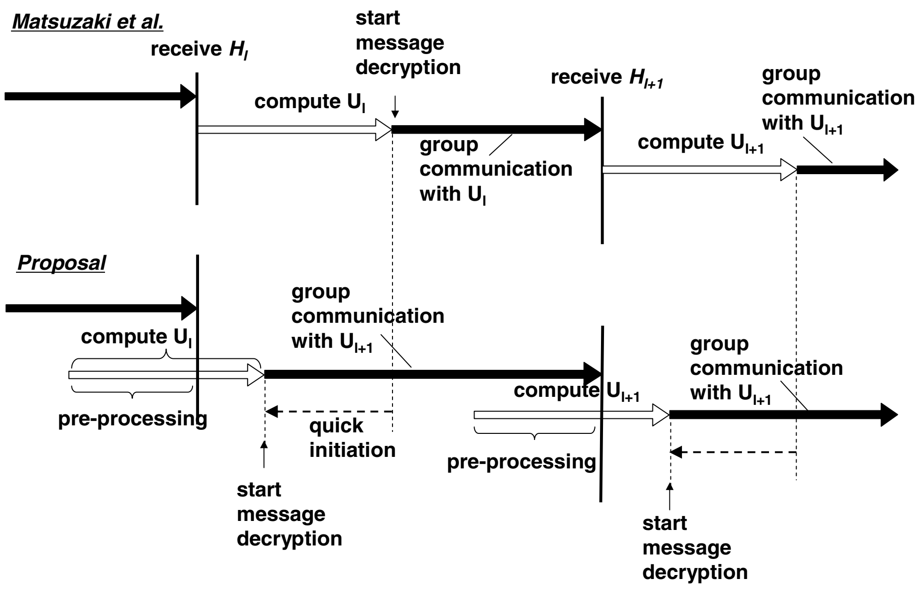


Fig. 1. Rapid decryption of new broadcast key using pre-processing

applications unless changes of the rounds happen frequently. As shown in Figure 1, we use the fact that shifting a part of the computation from the real-time processing into the pre-processing makes a significant improvement to accelerate the initiation of the rounds even if the overall computational overhead remains unchanged.

Our contribution provides a new efficient broadcast exclusion protocol maintaining the security for a virtually unlimited number of rounds while restraining the number of modular exponentiations in the real-time processing $O(1)$. First, we demonstrate a limitation of the rounds in [8] by showing how a user who observes the communications from the broadcaster for more than a certain number of rounds can derive the system secret parameters, which implies a total break of the scheme. Then, we present a new efficient broadcast exclusion protocol for which we provide a rigorous security proof by exploiting a strategy similar to that presented in [9][10]. Therefore, our protocol is proven secure against the collusion among up to k excluded users under the Computational Diffie-Hellman (CDH) assumption.

Table 1 shows a comparison of [5], [8], and ours. Apparently, [5] requires $O(k)$ modular exponentiations during the real-time processing for computing the new broadcast key, and there exists no trivial way to shift the computation to the pre-processing. The approach of [8] and our proposal require only two modular exponentiations during the real-time processing, but only ours has no limitation on the rounds. Thus, as far as we know, this result is the first one which satisfies the above properties. Another difference between [5] and our protocol

Table 1. Comparison

Description	[5]	[8]	proposal
num. of exp. (real-time processing)	$O(k)$	$O(1)$	$O(1)$
num. of exp. (pre-processing)	0	0	$O(k)$
size of header	$O(k)$	$O(k)$	$O(k)$
size of private key	$O(1)$	$O(1)$	$O(1)$
message distributor	arbitrary	$\mathcal{B}$ only	$\mathcal{B}$ only
num. of rounds	unlimited	limited	unlimited

is to require a fixed-privileged broadcaster. As with [8], this is considered to be reasonable in many applications.

We note that even if we point out some limitations of [8], we consider their work nevertheless significant. In particular, they found a potential problem in the earlier work and introduced important steps to reduce the amount of computation. We use some of their fundamental techniques as basic building blocks.

1.3 Organization

Our paper is structured as follows: In Section 2, we describe the model of our broadcast exclusion protocol and the definitions used in this paper. Section 3 shows some limitations on the multi-round security of the preceding works. We present our new protocol in Section 4, and show the security proof of the protocol in Section 5. Finally, we shall conclude in Section 6 with a summary of our results and issues that still remain to be solved.

2 Preliminaries

2.1 Definitions

We define the following notations used throughout the paper. Suppose a broadcaster, denoted as $\mathcal{B}$, wants to set up a broadcast communication channel only to a set of non-excluded users so that the set of excluded users chosen by $\mathcal{B}$ cannot get any information from the channel within a certain period of time, which is referred to as a round. $\mathcal{B}$ wants to be able to flexibly choose such a set of excluded users for every round. Let $\Phi = \{1, \ldots, n\}$ be the set of all users in the whole system, and $n = |\Phi|$ be the number of users.

At the time of system setup, $\mathcal{B}$ determines the system parameters and distributes a private key key_v to each user v in Φ. For the l-th round, the users in Φ are classified into a set of excluded users and a set of authorized (or non-excluded) users, denoted as Λ_l and Ω_l, respectively. We say a user v is authorized in the l-th round if v can compute a key U_l shared among $\mathcal{B}$ and Φ (which is called the broadcast key in this paper) from the header H_l using its private key key_v. We also say a user v' is excluded in the l-th round if v' cannot compute U_l from H_l. For simplicity, $\Omega_0 = \Phi$ and $\Lambda_0 = \{\emptyset\}$. The model of broadcast exclusion protocol we focus on in this paper is described as follows.

1. First, $\mathcal{B}$ sends to a user v ($\in \Phi$) the initial broadcast key U_0 and v's private key key_v over a secure private channel. $\mathcal{B}$'s public key is broadcasted to Φ.
2. The following procedures are iterated for $l \geq 1$ (we call the l-th iteration the "l-th round".)
   (a) $\mathcal{B}$ determines Λ_l. (i.e., $\Omega_l = \Phi \setminus \Lambda_l$).
   (b) $\mathcal{B}$ computes the header, denoted as H_l, and broadcasts it to Φ.
   (c) $v \in \Omega_l$ can compute U_l with H_l and key_v, while $v' \notin \Omega_l$ cannot do so.
   (d) U_l is used for $\mathcal{B}$ to securely send any message to Ω_l over a broadcast channel.

Remarks: The rule of exclusion is different between [8] and our protocol. In the scheme in [8], the user that is excluded once can never perform subsequent decryption of updated broadcast keys due to security reason, i.e., $\Lambda_l \supset \Lambda_{l-1}$. This means users excluded once have their permission permanently revoked unless their private keys are retransmitted[2]. On the other hand, our protocol allows a set of excluded users to be chosen independently for every round, i.e., the excluded user can continue to decrypt headers for the subsequent rounds without resending the new private key to him. This property will be nice when a subscriber does not pay a fee for a daily charged service, and thus the service must be stopped until the fee is paid. If he paid the fee, he can continue the decryption without any private communication with the broadcaster.

In another cases, however, it might be desirable that an excluded user stays excluded and can restart decryption only if the broadcaster resends him another set of keys. Our protocol can be modified in such a way that the excluded user cannot decrypt the headers for the further rounds only by encrypting the header information with the previous round broadcast key. In practice, the broadcast key might be distributed in the above two ways simultaneously to handle fine-grained control of the exclusion. For example, the former will be used for temporary exclusion at short interval, e.g., once a day, while the latter will be used for permanent exclusion at longer interval, e.g., once a week.

The parameters used in this paper are defined here. Let p be a prime power, q be a prime such that $q|p-1$, and g be a q-th root of unity over $GF(p)$ and $\langle g \rangle$ be a subgroup in $GF(p)$ generated by g. All the participants agree on p, q, g and $\langle g \rangle$. All arithmetic operations in this article are done in $GF(p)$ hereafter unless otherwise noted.

We assume that no polynomial-time algorithm solves $\log_g h$ in Z_q only with negligible probability in the size of q when h is selected uniformly from $\langle g \rangle$. We also make stronger assumptions: the intractability of the Computational Diffie-Hellman problem (CDH) in $\langle g \rangle$. We say that CDH in the group $\langle g \rangle$ is intractable if no probabilistic polynomial-time algorithm can find $g^{uv} \in \langle g \rangle$ from g^u and g^v.

Let $E_K(m)$ be a ciphertext given by symmetric key encryption of m with the key K. The discussion in this paper assumes that the underlying symmetric encryption $E()$ is sufficiently secure. Let k be the maximum number of excluded

[2] To support this feature, the header H_l is an encrypted message with the previous broadcast key U_{l-1}. See Section 3.

users for a single round. In our model of broadcast exclusion, we assumes $k \ll n$, e.g., k=1,000 and n=1,000,000.

2.2 Requirements

This paper presents a solution to meet the following requirements for the security and the efficiency constraints.

1. The user $v \in \Omega_l$ which is authorized in the l-th round can compute the broadcast key U_l (within a deterministic polynomial time in the size of $\langle g \rangle$).
2. An adversary who controls at most k users excluded in the l-th round cannot compute the broadcast key U_l (within a probabilistic polynomial time in the size of $\langle g \rangle$).W
3. The length of the header H_l and the size of the private key key_v do not depend on the number of users n.
4. The number of modular exponentiations in the *real-time processing* does not depend on either n or k, where "real-time processing" means the part of the computation which must be performed in order to derive U_l after receiving H_l.
5. The protocol does not have any limitation of the number of rounds.

In Requirement 2, we regard the adversary as an algorithm that can access all the information that can be accessed by up to k excluded users before the attack. More precisely, the adversary $\mathcal{A}_R$ who controls k excluded users in Λ_R can access the initial broadcast key U_0, the private keys of the k excluded users, the header $H_1, \ldots, H_R$, and the public information, where we say the protocol meets Requirement 2 if $\mathcal{A}_R$ cannot compute U_R within a probabilistic polynomial time in the size of $\langle g \rangle$. A more rigorous definition appears in Section 5.

2.3 Related Works

We briefly overview earlier work that focuses on the issues described above. The schemes in [4] and [2] presented solutions to Requirements 1 and 2 without making any computational assumptions, while the length of the header and the size of the private keys depend on the number n, and therefore Requirement 3 still remains to be solved. The schemes in [5] and [8] meet Requirement 3, since the length of the header is $O(k)$ and the size of the private key is $O(1)$. The scheme in [5] requires $O(k)$ modular exponentiations for each user to compute the updated broadcast key, while the scheme in [8] requires only two modular exponentiations, which implies [8] also meets Requirement 4.

However our analysis described in the following section shows that [8] has a limitation of the number of rounds. More precisely, more than a certain number of updates of the broadcast key results in a total break of the system. An objective of this paper is to modify the protocol of [8] to make it a multi-round secure one without spoiling the important feature of eliminating the $O(k)$ modular exponentiations. Our protocol is closely based on their original work in [5] and [8]. Our protocol actually uses some of the techniques suggested there, resulting in a protocol which can be proven secure under the CDH-assumption.

3 Multi-round Security of the Light-Weight Broadcast Exclusion Protocol

Before the detailed discussion of the security in [8], we present an outline of that protocol below.

1. (Setup) $\mathcal{B}$ determines the maximum number of excluded users, denoted as k, and randomly chooses two univariate polynomials of degree k over Z_q, denoted as $F(x) = \sum_{j=0}^{k} a_j x^j$, $G(x) = \sum_{j=0}^{k} b_j x^j$, where $F(0) = S$ and $G(0) = T \pmod q$ are secret values only $\mathcal{B}$ knows. Then, $\mathcal{B}$ chooses the initial broadcast key U_0 from $\langle g \rangle$ at random, and sends a pair consisting of U_0 and the private key $key_i = (s_i, f_i) = (F(i), g^{G(i)/F(i)})$ to each user i in Φ over a secure private channel.
2. (Encryption of the broadcast key) Transmitting the l-th round broadcast key U_l is done as follows. First, $\mathcal{B}$ randomly chooses r_l from Z_q, and computes $X_l = g^{r_l}$. Then it determines a set Δ_l of d users to be excluded from Ω_l. In the scheme in [8], any user that is excluded once can never perform subsequent decryption of updated broadcast keys. Thus, excluding users once means their permissions are permanently revoked unless their private keys are retransmitted (formally, $\Lambda_l = \Lambda_{l-1} \cup \Delta_l$ and $\Omega_l = \Omega_{l-1} \setminus \Delta_l$). $\mathcal{B}$ selects a set Θ_l of $k - d$ integers from $Z_q \setminus (\Omega_l \cup (\bigcup_{j=1}^{l-1} \Theta_j))$, and then computes M_{lj} ($j \in \Delta_l \cup \Theta_l$) as
$$M_{lj} = r_l F(j) + G(j) \bmod q \quad .$$
Finally, $\mathcal{B}$ computes the header H_l as
$$H_l = E_{U_{l-1}}(B_l) = E_{U_{l-1}}(X_l \| \{(j, M_{lj}) \mid j \in \Delta_l \cup \Theta_l\})$$
and broadcasts it to every user, where U_{l-1} is the $(l-1)$-th broadcast key, and $\mathcal{B}$ can compute a new broadcast key U_l shared among all of the authorized users in the l-th round as $U_l = g^{r_l S + T}$.
3. (Decryption of the broadcast key) The user $v \in \Omega_l$ authorized in the l-th round has already obtained U_{l-1} before the $(l-1)$-th round because if $v \in \Omega_l$, then $v \in \Omega_{l-1}$. Then v computes B_l by decrypting the received ciphertext $E_{U_{l-1}}(B_l)$ with U_{l-1}, and obtains the new broadcast key U_l as
$$U_l = (X_l f_v)^{W_{l1}} g^{W_{l2}}$$
where W_{l1} and W_{l2} are represented by
$$W_{l1} = s_v L(v) \bmod q$$
$$W_{l2} = \sum_{j \in \Delta_l \cup \Theta_l} (M_{lj} L(j)) \bmod q$$
where $L(j)$ is a Lagrange interpolation coefficient which can be derived from
$$L(j) = \prod_{t \in \Delta_l \cup \Theta_l \cup \{v\} \setminus \{j\}} t/(t-j) \bmod q \ .$$

In the above schemes, the number of modular exponentiations required to compute U_l is only two regardless of n and k. The scheme, however, has a limitation of the number of rounds. Consider any non-excluded user in the R-th round, denoted as $v \in \Omega_R$, who obtains (j, M_{lj}) $(j \in \Delta_l \cup \Theta_l)$ in the l-th round for $l = 1, \ldots, R$, where the following equations are satisfied.

$$M_{lj} = r_l \sum_{t=0}^{k} a_t j^t + \sum_{t=0}^{k} b_t j^t \bmod q$$

$$(l = 1, \ldots, R,\ j \in \Delta_l \cup \Theta_l,\ |\Delta_l \cup \Theta_l| = k)$$

The value of j is known, while the values of $a_0, \ldots, a_k$, $b_1, \ldots, b_k$, and r_l are unknown. Thus, these equations gives a system of kR equations w.r.t. $2k+2+R$ variables, $a_0, \ldots,\ a_k$, $b_0, \ldots, b_k$, $r_1, \ldots, r_R$. According to this reduction, if there is an algorithm to solve the system of equations, v can compute all of those $2k + 2 + R$ variables even if the discrete logarithm problem is hard to compute. We remark that $S(= a_0)$ and $T(= b_0)$ are the secret information of $\mathcal{B}$, so the exposure of these secrets implies a total break of the system.

There is a chance to solve systems of equations if the number of equations exceeds the number of variables. This happens if $kR \geq 2k+2+R$, i.e., $R \geq 3$ for any k (≥ 5). However, in general, solving large systems of quadratic multivariate polynomial equations is NP-hard over any field. When the number of equations γ_m is the same as the number of unknowns γ_n, the best known algorithms are exhaustive search for small fields, and a Gröbner base algorithm for large fields[11][12]. Gröbner base algorithms have large exponential complexity and cannot solve large systems in practice.

The number of equations becomes larger than the number of equations after a certain number of rounds. Recently, efficient algorithms for solving overdefined systems of multivariate polynomial equations have been proposed in [13]. The asymptotic complexity of their algorithm is expected to be polynomial with an exponent of $O(1/\sqrt{\epsilon})$ if the number of equations γ_m and the number of variables γ_n are related by $\gamma_m \geq \epsilon {\gamma_n}^2$ for any constant $0 < \epsilon \leq 1/2$ and the maximum degree D approximates $\lceil 1/\sqrt{\epsilon} \rceil$.

As the rounds proceed, the number of equations and the number of variables are increased simultaneously, whereas it is easy to see that solving the above system of kR equations of k degree with $2k + 2 + R$ variables can be reduced into solving a system of $(k - 1)R$ quadratic equations with $2k + 2$ variables (fixed regardless of R), as follows: First, the equations in the l-th round can be represented by

$$\begin{pmatrix} M_{l1} \\ M_{l2} \\ \vdots \\ M_{lk} \end{pmatrix} = \begin{pmatrix} r_l a_0 + b_0 \\ r_l a_0 + b_0 \\ \vdots \\ r_l a_0 + b_0 \end{pmatrix} + B \begin{pmatrix} r_l a_1 + b_1 \\ r_l a_2 + b_2 \\ \vdots \\ r_l a_k + b_k \end{pmatrix} \quad ,$$

where B is a $k \times k$ matrix.

$$B = \begin{pmatrix} j_{l1} \ {j_{l1}}^2 \cdots {j_{l1}}^k \\ j_{l2} \ {j_{l2}}^2 \cdots {j_{l2}}^k \\ \vdots \quad \vdots \qquad \vdots \\ j_{lk} \ {j_{lk}}^2 \cdots {j_{lk}}^k \end{pmatrix}$$

B is nonsingular because it is a Vandermonde matrix. Therefore,

$$\begin{pmatrix} r_l a_1 + b_1 \\ r_l a_2 + b_2 \\ \vdots \\ r_l a_k + b_k \end{pmatrix} = B^{-1} \begin{pmatrix} M_{l1} - r_l a_0 - b_0 \\ M_{l2} - r_l a_0 - b_0 \\ \vdots \\ M_{lk} - r_l a_0 - b_0 \end{pmatrix} \quad ,$$

Each line consists of a linear equation with respect to r, and thus k equations can be reduced to $k-1$ equations that have only $2k+2$ variables, $a_0, \ldots, a_k$ and $b_0, \ldots, b_k$. As a result, the kR equations with $2k+2+R$ variables can be reduced to $(k-1)R$ equations with $2k+2$ variables. The solving algorithm in [13] works efficiently if

$$(k-1)R \geq \epsilon(2k+2)^2 \ \Leftrightarrow \ R \geq \frac{\epsilon(2k+2)^2}{k-1} \quad \approx 4\epsilon k$$

for any constant $0 < \epsilon \leq 1/2$. This implies that the scheme in [8] has a limitation regarding the number of rounds.

4 Protocol

The goal of this paper is to provide the two desirable properties: (1) rapid computation of broadcast keys, and (2) no limitation of the number of rounds. The main idea is to reduce the number of exponentiations in the *real-time processing* by employing a technique for pre-processing. The underlying mechanism is based on the schemes in [5] and [8]. Our protocol is described as follows:

Setup. $\mathcal{B}$ determines the maximum number of excluded users, denoted as k, and randomly chooses two univariate polynomials of degree k over Z_q, denoted as $F(x) = S + \sum_{j=1}^{k} a_j x^j$, $G_1(x) = T_1 + \sum_{j=1}^{k} b_{1j} x^j$, where $F(0) = S$ and $G(0) = T_1 \pmod q$ are secret values only $\mathcal{B}$ knows. Then $\mathcal{B}$ chooses the initial broadcast key U_0 from $\langle g \rangle$ at random, and sends a pair consisting of U_0 and the private key of the first round, denoted as $key_i = (s_i, f_{1i}) = (F(i), g^{G_1(i)/F(i)})$, to each user i in Φ over a secure private channel.

Encryption of the broadcast key for the *l*-th round. Transmitting the l-th round broadcast key U_l is done as follows. First, $\mathcal{B}$ randomly chooses r_l from Z_q, and computes $X_l = g^{r_l}$. Then it determines a set Λ_l of d users to be excluded. In our protocol, a set of excluded users are independently chosen for each round, which is a different feature from the protocol in [8]. Thus, excluding users in a round means they are revoked only until the broadcast key is refreshed, i.e., $\Lambda_l \subset \Phi$ and $\Omega_l = \Phi \setminus \Lambda_l$. $\mathcal{B}$ selects a set Θ_l of $k-d$ integers from $Z_q\Phi$, and then computes M_{lj} $(j \in \Lambda_l \cup \Theta_l)$ as

$$M_{lj} = r_l F(j) + G_l(j) \bmod q \ ,$$

and constructs B_l, the information for a user in Ω_l to compute U_l, as

$$B_l = \langle X_l \parallel \{(j, M_{lj}) | j \in \Lambda_l \cup \Theta_l\} \rangle \ ,$$

where $\mathcal{B}$ can compute U_l by $U_l = g^{r_l S + T_l}$. Then, $\mathcal{B}$ produces the information C_l for pre-processing a part of the computation of the next round broadcast key U_{l+1}. $\mathcal{B}$ chooses $b_{l+1,j}$ $(j = 0, \ldots, k)$ from Z_q at random, and computes $(u_{lj} = g^{b_{l+1,j}}$, where C_l can be represented by

$$C_l = (u_{l0} \| \ldots \| u_{lk}) \quad .$$

Finally, $\mathcal{B}$ computes the header H_l, which can be derived from

$$H_l = (B_l \| C_l) \quad ,$$

and broadcasts it to every user.

Decryption of the broadcast key for the l-th round. The user v receives H_l, which contains B_l and C_l, and checks if v is authorized in this round. If so, v can compute U_l using

$$U = (X_l f_{lv})^{W_{l1}} g^{W_{l2}}$$

$$W_{l1} = s_v L(v) \bmod q$$

$$W_{l2} = \sum_{j \in \Lambda \cup \Theta} M_{lj} L(j) \bmod q \ ,$$

where

$$L(j) = \prod_{t \in \Lambda \cup \Theta \cup \{v\} \setminus \{j\}} t/(t-j) \bmod q \ .$$

Pre-processing for computing U_l. Before receiving H_{l+1}, each user v performs pre-processing with

$$f_{l+1,v} := \left(\prod_{j=0}^{k} {u_{lj}}^{v^j} \right)^{1/s_v}$$

For computing H_{l+1}, $\mathcal{B}$ updates T_{l+1} and $G_{l+1}(x)$ as

$$T_{l+1} := b_{l+1,0} \bmod q$$

$$G_{l+1}(x) := \sum_{j=0}^{k} b_{l+1,j} x^j \bmod q.$$

5 Security

Suppose an adversary $\mathcal{A}_R$ who controls k users excluded in the R-th round, denoted as Λ_R, tries to find the R-th broadcast key U_R. As mentioned in Requirement 2, $\mathcal{A}_R$ can be formulated as a probabilistic algorithm $\mathcal{M}_1$ which given all the information observed by Λ_R outputs U_l.

Definition 1 (Adversary). *$\mathcal{M}_1$ is a probabilistic polynomial time algorithm in the size of $\langle g \rangle$, which takes the following values and outputs U_R with a negligible probability.*

- *public information : $g.p, q, k$*
- *private keys of k excluded users in Λ_R : $\{(s_j, f_{1j}) \mid j \in \Lambda_R\}$*
- *the initial broadcast key : U_0*
- *the header : $H_1, \ldots, H_R$*

Theorem 1. *$\mathcal{M}_1$ is as hard as CDH in the group $\langle g \rangle$.*

Proof. (Sketch)
Let $\mathcal{M}_2$ be the probabilistic polynomial time algorithm to solve CDH.
$(\mathcal{M}_2 \to \mathcal{M}_1)$ It is clear that the existence of $\mathcal{M}_2$ implies the existence of $\mathcal{M}_1$.
$(\mathcal{M}_1 \to \mathcal{M}_2)$ Suppose there exists $\mathcal{M}_1$. We show $\mathcal{M}_2$ by using $\mathcal{M}_1$ as a subroutine. Let the input to $\mathcal{M}_2$ be (α, β). The goal is to compute $\gamma = \beta^{\log_g \alpha}$ by using $\mathcal{M}_1$ as a subroutine. Intuitively, the input to $\mathcal{M}_1$ is constructed by using the following strategy. In the setup phase, $\mathcal{M}_2$ determines $F(x)$ such that $F(0) = S = \log_g \alpha$, and then computes all the values given to $\mathcal{M}_1$ with T_R and X_R such that T_R is chosen at random from Z_q and $r_R = \log_g \beta$. After the setup, $\mathcal{M}_2$ invokes $\mathcal{M}_1$ and obtains its output $U_R = g^{r_R S + T_R}$. Finally, $\mathcal{M}_2$ computes $g^{r_R S} = U_R / g^{T_R}$, which implies $\mathcal{M}_2$ can derive $\gamma = \beta^{\log_g \alpha}$. The procedure using M_2 proceeds as follows. Let a set of k integers in $\Lambda_l \cap \Theta_l$ be $\Gamma_l = \{\psi_{l1}, \ldots, \psi_{lk}\}$.

1. R set of k users $\Gamma_1, \ldots, \Gamma_R$ are uniformly chosen from Φ, where $\Lambda_R = \Gamma_R$.
2. The following parameters are uniformly chosen from Z_q.
   - $s_j \ (j \in \Lambda_R)$
   - $r_l \ (l = 1, \ldots, R-1)$
   - $T_l \ (l = 1, \ldots, R)$
   - $M_{lj} \ (l = 1, \ldots, R, \ j \in \Gamma_l)$
   - $b_{R+1,t} \ (t = 0, \ldots, k)$
3. U_0 is uniformly chosen from $\langle g \rangle$.
4. S and r_R are defined as $S = \log_g \alpha$ and $r_R = \log_g \beta$ (but these values are not explicitly known).
5. Let us consider a polynomial $F(x)$ of k degree such that $F(0) = S, F(j) = s_j (j \in \Lambda_R)$. Such a polynomial $F(x)$ is uniquely determined by

$$F(x) = S\lambda_0(x) + \sum_{j \in \Lambda_R} s_j \lambda_j(x) \bmod q$$

where $\lambda_j(x)$ is a function derived by Lagrange interpolation, and defined as

$$\lambda_j(x) = \prod_{j' \in \Lambda_R,\ j' \neq j} \frac{x - j'}{j' - j} \bmod q.$$

From the above observation, it is easy to see that $g^{F(x)}$ and $g^{G_l(x)}$ can be computed using

$$g^{F(x)} = \alpha^{\lambda_0(x)} \times \prod_{j \in \Lambda_R} g^{s_j \lambda_j(x)}$$

$$g^{G_l(\psi_{lj})} = \begin{cases} g^{M_{lj}} / (g^{F(\psi_{lj})})^{r_l} & (l = 1, \ldots, R-1, j \in \Gamma_l) \\ g^{M_{lj}} / (g^{F(\psi_{lj})})^{r_R} = g^{M_{lj}} / \beta^{s_j} & (l = R,\ j \in \Lambda_R) \end{cases}$$

6. From the definition,

$$B_l \begin{pmatrix} b_{l0} \\ b_{l1} \\ b_{l2} \\ \vdots \\ b_{lk} \end{pmatrix} = \begin{pmatrix} T_l \\ G_l(\psi_{l1}) \\ G_l(\psi_{l2}) \\ \vdots \\ G_l(\psi_{lk}) \end{pmatrix} \quad ,$$

for $l = 1, \ldots, R$, where B_l is a $(k+1) \times (k+1)$ matrix defined by

$$B_l = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & \psi_{l1} & {\psi_{l1}}^2 & \cdots & {\psi_{l1}}^k \\ 1 & \psi_{l2} & {\psi_{l2}}^2 & \cdots & {\psi_{l2}}^k \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \psi_{lk} & {\psi_{lk}}^2 & \cdots & {\psi_{lk}}^k \end{pmatrix}$$

B_l is nonsingular because it is a Vander monde matrix, so its inverse matrix ${B_l}^{-1} = \{\mu_{ij}\}_{0 \le i,j \le k}$ can be defined. Thus, $b_{lj} = \mu_{lj} T_l + \sum_{j'=1}^{k} \mu_{jj'} G_l(\psi_{lj'})$ $(j = 0, \ldots, k)$. From this observation, $u_{l-1,j} (j = 0, \ldots, k,\ l = 1, \ldots, R)$ is derived from

$$u_{l-1,j} = g^{b_{lj}} = g^{\mu_{lj} T_l} \times \prod_{j'=1}^{k} \left(g^{G_l(\psi_{lj'})}\right)^{\mu_{jj'}}$$

for $j = 0, \ldots, k,\ l = 1, \ldots, R$. Note that $g^{G(x)}$ can be computed for $x \in \Gamma_l, l = 1, \ldots, R$ as mentioned before.

7. f_{1j} can be computed for $j \in \Lambda_R$ using

$$f_{1j} = g^{G^{(1)}(j)/F(j)} = \left(\prod_{j'=0}^{k} (g^{b_{1j'}})^{j^{j'}} \right)^{1/F(j)} = \left(\prod_{j'=0}^{k} {u_{0j'}}^{j^{j'}} \right)^{1/s_j}$$

8. $X_l = g^{r_l}$ $(l = 1, \ldots, R)$ and $u_{Rj} = g^{b_{R+1,j}}$ $(j = 0, \ldots, k)$
9. After the above setup, H_l $(l = 1, \ldots, R)$ can be represented as

$$B_l = \langle X_l \parallel \{(j, M_{lj}) | j \in \Gamma_l\} \rangle$$

$$C_l = (u_{l0} \| \ldots \| u_{lk})$$

$$H_l = (B_l \parallel C_l)$$

10. $\mathcal{M}_2$ calls $\mathcal{M}_1$ with the inputs: $(g, p, q, k, U_0, \{(s_j, f_{1j}) | j \in \Lambda_R\}, H_1, \ldots, H_R)$. If $\mathcal{M}_1$ outputs U_R, $\mathcal{M}_2$ can compute γ by $\gamma = U_R / g^{T_R}$. Thus, we can conclude $\mathcal{M}_1$ implies $\mathcal{M}_2$.

Forward Secrecy. Our protocol provides forward secrecy in a sense that an adversary $\mathcal{A}_R$ cannot compute U_R even if $\mathcal{A}_R$ can access any broadcasted information given in the future (which implies $\mathcal{A}_R$ might access future broadcast keys). Therefore, haing a key only gives information about the current round, and that there is not link between past, present and future. To prove this, we can slightly extend the above definition of M_1 to M_1^*.

Definition 2. *$\mathcal{M}_1^*$ is a probabilistic polynomial time algorithm in the size of $\langle g \rangle$, which takes $\mathcal{M}_1$'s inputs and the future headers $H_{R+1}, \ldots, H_{R'}$ for any $R' > R$, and outputs U_R with a negligible probability.*

Theorem 2. *$\mathcal{M}_1^*$ is as hard as CDH in the group $\langle g \rangle$.*

Proof. (Sketch)
The almost same proof as that to Theorem 1 can be applied. The only differnece is $\mathcal{M}_1^*$'s additional input which can be computed at step 8 and 9 by choosing r_l, M_{lj} $(j \in \Gamma_l), b_{l+1j}$ $(j = 0, \ldots, k)$ uniformly from Z_q for $l = R+1, \ldots, R'$.

6 Conclusion

This paper presented a provably secure solution to overcome a limitation of the prior broadcast exclusion protocol by resharing a blinding polynomial in a per-round pre-processing step. Our method improves the performance of updating rounds because major part of computation can be performed within the pre-processing period.

As mentioned in Section 1, we assume that the pre-processing can be completed before starting to decrypt the header for the next round. In some applications such as PPV, however, broadcast keys have to be changed frequently. Our protocol might not handle the frequent round update because it requires $O(k)$ pre-processing for each round. Reducing the pre-processing will be an issue to be solved.

Acknowledgements

The authors would like to thank Junji Shikata for his many helpful comments, and the anonymous referees for useful suggestions. We also appreciate Natsume Matsuzaki, Jun Anzai and Tsutomu Matsumoto because our result is closely built on their prior works.

References

1. Amos Fiat and Moni Naor. Broadcast encryption. In *Advances in Cryptology - CRYPTO '93*, LNCS 773, pp 480–491. Springer-Verlag, 1994.
2. Dalit Naor, Moni Naor, and Jeffrey B. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO 2001*, LNCS 2139, pp 41–62. Springer-Verlag, 2001.
3. Tomoyuki Asano. A revocation scheme with minimal storage at receivers. In *Advances in Cryptology - ASIACRYPT 2002*, LNCS 2501, pp 433–450. Springer-Verlag, 2002.
4. Ravi Kumar, Sridhar Rajagopalan, and Amit Sahai. Coding constructions for blacklisting problems without computational assumptions. In *Advances in Cryptology - CRYPTO '99*, LNCS 1666, pp 609–623. Springer-Verlag, 1999.
5. Jun Anzai, Natsume Matsuzaki, and Tsutomu Matsumoto. A quick group key distribution scheme with "entity revocation". In *Advances in Cryptology - ASIACRYPT '99*, LNCS 1716, pp 333–347. Springer-Verlag, 1999.
6. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO '89*, LNCS 435, pp 307–315. Springer-Verlag, 1990.
7. Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *Financial Cryptography 2000*, LNCS 1962, pp 1–20. Springer-Verlag, 2001.
8. Natsume Matsuzaki, Jun Anzai, and Tsutomu Matsumoto. Light weight broadcast exclusion using secret sharing. In *5th Australasian Conference Information Security and Privacy (ACISP'2000)*, LNCS 1841, pp 313–327. Springer-Verlag, 2000.
9. Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *Advances in Cryptology - EUROCRYPT '98*, LNCS 1403, pp 145–157. Springer-Verlag, 1998.
10. Kaoru Kurosawa and Takuya Yoshida. Linear code implies public-key traitor tracing. In *5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002*, LNCS 2274, pp 172–187. Springer-Verlag, 2002.
11. E. R. Berlekamp. Factoring polynomials over large finite fields. In *Math. Comp.*, pp 713–735, 1970.
12. D. E. Knuth. Seminumerical algorithms - the art of computer programming. In *Addison-Wesley.*
13. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Proc. of EUROCRYPT'2000*, LNCS 1807, pp 392–407. Springer-Verlag, 2000.

Precisely Answering Multi-dimensional Range Queries without Privacy Breaches[⋆]

Lingyu Wang, Yingjiu Li[⋆⋆], Duminda Wijesekera, and Sushil Jajodia

Center for Secure Information Systems, George Mason University
Fairfax, VA 22030-4444, USA
{lwang3,yli2,dwijesek,jajodia}@gmu.edu

Abstract. This paper studies the privacy breaches caused by multi-dimensional range (MDR) sum queries in online analytical processing (OLAP) systems. We show that existing inference control methods are generally infeasible for controlling MDR queries. We then consider restricting users to even MDR queries (that is, the MDR queries involving even numbers of data values). We show that the collection of such even MDR queries is safe if and only if a special set of sum-two queries (that is, queries involving exactly two values) is safe. On the basis of this result, we give an efficient method to decide the safety of even MDR queries. Besides safe even MDR queries we show that any odd MDR query is unsafe. Moreover, any such odd MDR query is different from the union of some even MDR queries by only one tuple. We also extend those results to the safe subsets of unsafe even MDR queries.

1 Introduction

Multi-dimensional range (MDR) query is an important class of decision support query in online analytical processing (OLAP) systems [19]. One of the most popular data models of OLAP systems, data cube [18], can be viewed as a special collection of MDR queries. MDR queries are intended to assist users in exploring trends and patterns in large amounts of data stored in data warehouses. Contrary to this initial objective, MDR queries can be used to obtain protected sensitive data, which results in the breach of an individual's privacy. Access control alone is insufficient in controlling information disclosure, because information not released directly may be inferred indirectly from the answers to legitimate queries, which is known as the *inference problem* in databases. Providing precise answers to MDR queries without privacy breaches is the subject of this paper.

The inference problem has been investigated since the 1970's with many inference control methods proposed, especially for statistical databases. Those methods usually have run times proportional to the size of the queries or the data sets, and they are invoked only after queries have arrived. On the other hand, OLAP applications demand instant

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

[⋆⋆] The author's current address is School of Information Systems, Singapore Management University, Singapore 259756

Table 1. An Example of a Two-dimensional Data Core.

The Data Core $year\_emp\_adj$

year / emp / adj	Alice	Bob	Mary	Jim
2002	1000.00	500.00	-2000.00	
2003		1500.00	-500.00	1000.00

responses to queries, although the queries usually aggregate a large amount of data. Consequently, the delay in query answering renders most existing methods impractical for OLAP systems. In this paper we propose efficient inference control methods by exploiting the unique structures of MDR queries.

The first contribution of this paper is that it will call more attention to the privacy issue of OLAP systems, which is unfortunately ignored in most of today's commercial products. We study several existing inference methods and the results show that they are infeasible for MDR queries. We also show that finding maximal safe subsets of unsafe MDR queries is NP-hard. Second, we reduce the inference control of MDR queries to that of sum-two queries with a necessary and sufficient condition on their compromisability. By treating sum-two queries as edges of simple undirected graphs, this reduction relates the inference control of MDR queries with existing results in inference control in statistical databases and graph theory. Finally, we give efficient methods (the complexity is bounded by $O(mn)$, where m, n are the number of queries and tuples, respectively) to determine safe MDR queries, safe arbitrary queries, and large subsets of unsafe MDR queries.

The rest of the paper is organized as follows. Section 1.1 gives a motivating example, and Section 1.2 describes our assumptions. Section 2 reviews existing inference control methods proposed in traditional statistical databases and modern decision support systems. Section 3 presents basic definitions and formalizes MDR queries and the compromisability. Section 4 gives negative results of applying existing inference control methods to MDR queries. Section 5 investigates the problem of determining safe MDR queries. Section 6 extends the results to subsets of unsafe MDR queries. Section 7 discusses the implementation. Section 8 concludes the paper. Due to space limitations, we have omitted the proofs of all theorems, lemmata, corollaries, and propositions, which can be found in [31].

1.1 Motivating Example

Suppose that part of a data set owned by a fictitious organization, *Company A*, is shown in Table 1. It contains salary adjustments for four employees in years 2002 and 2003. Let the three attributes be $year$, emp (employee), and adj (adjustment), respectively. The empty cells in Table 1 indicate that the employee did not work for *Company A* in that year.

Company A invites an analyst *Mallory* to analyze the data set. For this purpose, *Mallory* is allowed to ask sum queries about the attribute adj in Table 1. In addition, she has access to the non-sensitive attributes $year$, emp as well as the locations of empty cells in Table 1. On the other hand, *Company A* worries that *Mallory* may inappropriately use the information she learns about each employee. Hence, *Mallory* is prohibited from directly

Table 2. An Example of Even MDR Queries.

Ranges	Answer
$[(Alice, 2002), (Jim, 2003)]$	1500
$[(Alice, 2002), (Bob, 2002)]$	1500
$[(Bob, 2002), (Mary, 2002)]$	-1500
$[(Bob, 2002), (Bob, 2003)]$	2000
$[(Mary, 2003), (Jim, 2003)]$	500

asking the individual values (of attribute adj) in Table 1. Now we ask the following Questions: *Can Mallory learn any of the individual values through sum queries?* If so, *how can we safeguard these values?* Suppose *Mallory* asks the following query:

```
SELECT emp, SUM(adj)
FROM year_emp_adj
GROUP BY emp;
```

The answer to the above query contains four records $(Alice, 1000)$, $(Bob, 2000)$, $(Mary, -2500)$ and $(Jim, 1000)$. Each record corresponds to a one-dimensional MDR sum query, such as $(Alice, 1000)$, which sums the values in the first column of the table. Intuitively, by viewing each MDR query as a *box*, we can represent it using its longest *diagonal*. For example, use $[(Alice, 2002), (Alice, 2003)]$ for the first column of the table and $[(Alice, 2002), (Bob, 2003)]$ for the first two columns. For simplicity purpose, we shall use this notation instead of SQL for MDR query henceforth.

Mallory is able to learn from the MDR query $[(Alice, 2002), (Alice, 2003)]$ that the adjustment for Alice in 2002 is 1000.00, because the query sums a single value. This threat can be thwarted by answering only the MDR queries that sum two or more values. However, *Mallory* can easily get around this restriction by subtracting (the answers to) $[(Bob, 2002), (Mary, 2002)]$ from $[(Alice, 2002), (Mary, 2002)]$.

This second inference occurs because the queries $[(Bob, 2002), (Mary, 2002)]$ and $[(Alice, 2002), (Mary, 2002)]$ sum even (two) and odd (three) number of values, respectively. Is it helpful for protecting the individual values to restrict *Mallory* to only *even MDR queries* (MDR queries involving even number of values) or only *odd MDR queries* (MDR queries involving odd number of values)? The restriction to odd MDR queries is ineffective because the difference of two odd numbers yields an even number. For example, the first two and three columns of Table 1 are both odd, but their difference gives the third column, which is even. Conversely, to obtain odd MDR queries from even ones is not always straightforward. Because an individual value can also be viewed as an odd MDR query, restricting users to even MDR queries makes inferences substantially more difficult.

Nonetheless, inference is still possible with only even MDR queries. A series of five even MDR queries asked by *Mallory* and their answers are given in Table 2. The first query sums all six values and the remaining four queries each sum two values. *Mallory* then adds the answers to the last four queries (2500) and subtracts from the result the answer to the first queries (1500). Dividing the result of the subtraction (1000) by two gives Bob's adjustment in 2002 (500).

In the rest of this paper we address the following questions naturally motivated by the above example: *1. How can we efficiently determine whether even MDR queries are* *safe? 2. What is the impact on users if only even MDR queries are allowed? 3. Besides the even MDR queries, what else can be answered safely? 4. If even MDR queries are unsafe, can we find a large safe subset?*

1.2 Assumptions

We only consider *stateless* inference control methods. That is, the methods that grant or deny incoming queries independent of the queries previously asked by the user. For example, restrictions on the size or parity of queries are stateless. On the other hand, the *stateful* methods base authorization decisions on the history of queries asked by a specific user, for example, controlling the size of overlaps between queries. Stateful restrictions are usually infeasible in practice, because users can subvert them by using aliases for login or colluding.

We assume users do not possess the *external knowledge*[1] about the boundaries of protected individual values. Consequently, we consider the protected values as unbounded reals. Under that assumption, it is relevant for inference control to know which values users know and which they do not, but the specific values are irrelevant. For example, all the inferences we discuss in Section 1.1 are possible regardless of the explicit values we put in Table 1. Inference of approximate values caused by external knowledge about boundaries or data types has been studied in [22, 24]. Their inference control methods can be incorporated into our methods as post-processing, because the inferences we study require less external knowledge and should be checked first.

On the other hand, we assume users may know some of the protected values from external knowledge. For example, in Table 1 users know *Alice* does not have a valid salary adjustment in year 2003 because she has left *Company A* by the end of 2002. Regardless of the specific sources of external knowledge, we shall treat all known values as empty cells. We do not consider the known values of which the inference control mechanism is not aware (undetected external knowledge). Under this assumption, the summation of any two real unbounded values is considered safe. We address the issue of undetected external knowledge in Section 7.

2 Related Work

Inference control has been extensively studied in statistical databases [12, 1, 14] and the proposed methods are usually classified into two categories: *restriction-based* techniques and *perturbation-based* techniques. Restriction-based techniques include restricting the size of *query sets* (i.e., the tuples that satisfy a single query) [17], restricting the size of overlaps [15] between query sets, detecting inferences by auditing all queries asked by a specific user [10, 8, 20, 5], suppressing sensitive data in released statistical tables [11], and grouping tuples and treating each group as a single tuple [9, 25]. Perturbation-based techniques add noise to source data or outputs [28, 4, 27]. Other aspects of the inference problem include the inference caused by arithmetic constraints [6], inferring approximate values instead of exact values [24] and inferring intervals enclosing exact values [22, 21, 23]. The inference control methods proposed for statistical databases do

[1] The knowledge obtained from sources other than queries [12].

not consider the unique structure of MDR queries. This renders them ineffective and inefficient for MDR queries. We show some examples in Section 4.

Recently a variation of the inference control problem, namely, *privacy preserving data mining*, has drawn considerable attention as seen in [2, 26]. They all attempt to perturb sensitive values while preserving the classifications or association rules that can be learned from the data set. In doing so, they assume that a user's objective of data analysis is predictable. However, in OLAP systems this assumption may not hold, because we do not know in advance what users may want to discover. Our work does not have this limitation, because what we give users is not the results (e.g., classifications or association rules), but the means (the precise answers to their queries) to obtain the results they desire.

Controlling inferences of a special class of MDR queries, namely, *data cubes*, is studied in [29]. They give sufficient conditions for safe data cubes based on the cardinality of the data core. A data core is safe if it is full or dense (the number of known values is either zero or under the given bound). Note that this condition does not apply to those MDR queries that are not included in the data cube. This paper strengthens this result by giving necessary and sufficient conditions for all MDR queries.

The inference problem of one-dimensional range queries is studied in [8], and the MDR case is considered difficult. The *usability* (i.e., the highest possible ratio of the number of safe queries to that of all queries) of MDR queries in the full core is studied in [5]. They mention but do not fully explore the restriction of even MDR queries. However, the general case with known values (referred to as *holes* in [5]) is thought to be challenging. In [7, 10] Chin gives necessary and sufficient condition for the compromisability of sum-two queries. He also proves that finding the maximal safe subsets of unsafe sum-two queries is NP-hard. However, sum-two queries are rare in practice. In this paper we use his results by reducing the compromisability of even MDR queries to that of sum-two queries.

3 Basic Definitions

This section defines the basic concepts and notations. We use $\mathbb{I}, \mathbb{R}, \mathbb{I}^k, \mathbb{R}^k, \mathbb{R}^{m\times n}$ to denote the set of integers, reals, k-dimensional integer vectors, k-dimensional real vectors and m by n real matrices, respectively. For any $u, v, t \in \mathbb{R}^k$, we write $u \leq v$ and $t \in [u, v]$ to mean that $u[i] \leq v[i]$ and $min\{u[i], v[i]\} \leq t[i] \leq max\{u[i], v[i]\}$ for all $1 \leq i \leq k$, respectively. We use t for the singleton set $\{t\}$ whenever clear from the context.

Definition 1 (Core).
For any $d \in \mathbb{I}^k$, use $\mathcal{F}(d)$ to denote the Cartesian product $\Pi_{i=1}^k [1, d[i]]$. We say $F = \mathcal{F}(d)$ is the full core. Any $C \subseteq F$ is a core. Any $t \in F$ is a tuple. Any $t \in F \setminus C$ is a tuple missing from C.

Definition 1 formalizes the concepts of *full core*, *core*, and *tuple*. The full core is formed by the Cartesian product of closed integer intervals. A core is any subset of the full core. A tuple is any vector in the full core and a tuple missing from the core is any vector in the complement of the core with respect to the full core.

Definition 2 (MDR Query, Sum-two Query and Arbitrary Query). *Given any full core F and core $C \subseteq F$,*

1. *Define functions*
   (a) $q^\star(.) : F \times F \to 2^C$ *as* $q^\star(u,v) = \{t : t \in C, t \in [u,v]\}$.
   (b) $q^2(.) : C \times C \to 2^C$ *as* $q^2(u,v) = \{u,v\}$ *if* $u \neq v$, *and* ϕ *otherwise.*
2. *Use* $\mathcal{Q}_d(C)$ *and* $\mathcal{Q}_t(C)$ *(or simply* $\mathcal{Q}_d$ *and* $\mathcal{Q}_t$ *when* C *is clear from context) for* $\{q^\star(u,v) : q^\star(u,v) \neq \phi\}$ *and* $\{q^2(u,v) : q^2(u,v) \neq \phi\}$, *respectively.*
3. *We call any non-empty* $q \subseteq C$ *an arbitrary query, any* $q^\star(u,v) \in \mathcal{Q}_d$ *an MDR query (or simply query), and any* $q^2(u,v) \in \mathcal{Q}_t$ *a sum-two query.*

In Definition 2 we formalize the concepts of *arbitrary query*, *MDR query*, and *sum-two query*. An arbitrary query is any non-empty subset of the given core. An MDR query $q^\star(u,v)$ is a non-empty subset of the core that includes all and only those tuples *bounded* by two given tuples. Intuitively, an MDR query can be viewed as a multi-dimensional axis-parallel box. A sum-two query is any set of exactly two tuples. We use $\mathcal{Q}_d$ and $\mathcal{Q}_t$ for the set of all MDR queries and all sum-two queries, respectively.

Definition 3 (Compromisability). *Given any full core F, core $C \subseteq F$, and any set of arbitrary queries $\mathcal{S}$, use $\mathcal{M}(\mathcal{S})$ to denote the incidence matrix[2] of the set system formed by C and $\mathcal{S}$, we say that*

1. $\mathcal{S}_1$ *is derivable from* $\mathcal{S}_2$, *denoted as* $\mathcal{S}_1 \preceq_d \mathcal{S}_2$, *if there exists* $M \in \mathbb{R}^{|\mathcal{S}_1| \times |\mathcal{S}_2|}$ *such that* $\mathcal{M}(\mathcal{S}_1) = M \cdot \mathcal{M}(\mathcal{S}_2)$ *holds, where* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are sets of arbitrary queries.*
2. $\mathcal{S}_1$ *compromises* $t \in C$ *if* $t \preceq_d \mathcal{S}_1$ *(we write* t *for* $\{\{t\}\}$*), and* $\mathcal{S}_1$ *is safe if it compromises no* $t \in C$.
3. $\mathcal{S}_1$ *is equivalent to* $\mathcal{S}_2$, *denoted as* $\mathcal{S}_1 \equiv_d \mathcal{S}_2$, *if* $\mathcal{S}_1 \preceq_d \mathcal{S}_2$ *and* $\mathcal{S}_2 \preceq_d \mathcal{S}_1$.

Definition 3 formalizes the concept of compromisability and related concepts. Because an arbitrary query is a set of tuples, any given set of arbitrary queries can be characterized by the incidence matrix of the set system formed by the core and the set of arbitrary queries. Given two sets of arbitrary queries $\mathcal{S}_1, \mathcal{S}_2$, and the incidence matrices $\mathcal{M}(\mathcal{S}_1)$, $\mathcal{M}(\mathcal{S}_2)$, we say $\mathcal{S}_1$ is derivable from $\mathcal{S}_2$ if the row vectors of $\mathcal{M}(\mathcal{S}_1)$ can be represented as the linear combination of those of $\mathcal{M}(\mathcal{S}_2)$. Intuitively, this implies that the information disclosed through $\mathcal{S}_1$ can be computed from that through $\mathcal{S}_2$. We say $\mathcal{S}_1$ compromises a tuple t in the core if the set of queries $\{\{t\}\}$ (notice $\{t\}$ is an MDR query) is derivable from $\mathcal{S}_1$, and $\mathcal{S}_1$ is safe if it compromises no tuple in the core. We say any two sets of arbitrary queries are equivalent if they are mutually derivable.

Example 1. Table 3 gives an example of the core, MDR queries, and compromisability. As shown in the left upper table in Table 3, the core C contains six tuples. The subscripts of the tuples give their order. The right upper table shows a set of five MDR queries S. The lower equation shows that S compromises $(1,2)$. The left side of the equation is

[2] $\mathcal{M}(\mathcal{S})[i,j] = 1$ if the i^{th} arbitrary query in $\mathcal{S}$ contains the j^{th} tuple in C, and $\mathcal{M}(\mathcal{S})[i,j] = 0$ otherwise.

Table 3. An Example of Core, MDR Queries, and Compromisability.

The Core C

	1	2	3	4
1	(1,1)$_1$	(1,2)$_2$	(1,3)$_3$	
2		(2,2)$_4$	(2,3)$_5$	(2,4)$_6$

A Set of MDR Queries:$\boldsymbol{S}$

$q^\star((1,1),(2,4))$	$\{(1,1),(1,2),(1,3),(2,2),(2,3),(2,4)\}$
$q^\star((1,1),(1,2))$	$\{(1,1),(1,2)\}$
$q^\star((1,2),(1,3))$	$\{(1,2),(1,3)\}$
$q^\star((1,2),(2,2))$	$\{(1,2),(2,2)\}$
$q^\star((2,3),(2,4))$	$\{(2,3),(2,4)\}$

$\mathbf{(1,2)} \preceq_d \boldsymbol{S}$ **because**

$$[\mathbf{0,1,0,0,0,0}] = [-\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2}]\cdot \begin{pmatrix} 1\,1\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1 \end{pmatrix}$$

the incidence matrix of $(1,2)$, and the right side gives a linear combination of the row vectors in the incidence matrix of S. Table 3 characterizes exactly the same example given in Tables 1 and 2, except that it uses the concepts and notations we defined in this section.

The relation $\equiv_d$ of Definition 3 is an equivalence relation on the family of all sets of arbitrary queries, because it is reflexive, symmetric, and transitive. Hence if any two sets of arbitrary queries are equivalent, then one is safe iff the other is. In Section 5 we shall reduce the compromisability of even MDR queries to that of a special set of sum-two queries based on this fact.

4 Ineffective or Infeasible Restrictions

In this section we apply several existing restriction-based inference control methods to MDR queries. Our results show that they are ineffective or infeasible for MDR queries. We first investigate three methods, *Query set size control*, *overlap size control*, and *Audit Expert* in Section 4.1. Then we consider the problem of finding maximal safe subsets of unsafe MDR queries in Section 4.2.

4.1 Query Set Size Control, Overlap Size Control, and Audit Expert

Query Set Size Control. This method prohibits users from asking *small* queries whose cardinalities are smaller than some pre-determined threshold n_t [17]. For arbitrary queries, query set size control can be easily subverted by asking two legitimate queries whose intersection yields a prohibited one, a mechanism known as the *tracker* in statistical databases [13]. It is shown that finding a tracker for arbitrary queries is possible even when n_t is about half of the cardinality of the core. At first glance, trackers may seem to be more difficult to find when users are restricted to MDR queries. However, in [30] we show that when n_t is not big enough ($n_t \leq \frac{n}{3^k}$) a tracker can always be found to derive *any* given small MDR query, and the tracker consists of only MDR queries.

Overlap Size Control. This method prevents users from asking queries with large intersections [15]. Any answerable query must have a cardinality of at least n, and the intersection of any two queries is required to be no larger than r. In order to compromise any tuple t, one must first ask one query $q \ni t$ and subsequently $(n-1)/r$ or more queries to form the complement of t with respect to q. Consequently, no inference is possible if less than $(n-1)/r+1$ queries are answered. This bound is not improved (increased) by restricting users to MDR queries, because for almost any MDR query the complement of a tuple can always be formed. Overlap size control is infeasible because it is a stateful method. Moreover, it depends on the restriction of small queries, which is ineffective as described above.

Audit Expert. Chin gives a necessary and sufficient condition for determining safe arbitrary queries in Audit Expert [10]. By treating tuples and queries as a set system, the queries are safe iff the incidence matrix of the set system contains one or more unit row vectors in its reduced row echelon form (RREF). The elementary row transformation used to obtain the RREF of an m by n matrix has the complexity $O(m^2n)$. Using this condition *online* (after queries arrive) may incur unacceptable delay in answering queries because m and n can be very large in OLAP systems. Moreover, it is a stateful method because it requires the entire history of queries. A better way to use the condition is to determine the compromisability of queries off-line [5]. However, although this condition certainly applies to MDR queries, it is not efficient because it does not take into consideration the inherent redundancy among MDR queries. In Section 5 we further investigate this issue in detail.

4.2 Finding Maximal Safe Subsets of Unsafe MDR Queries

When a set of queries is not safe, it is desired to find its maximal safe subset. In [10] it has been shown that finding the maximal safe subset of unsafe arbitrary queries (the MQ problem) or sum-two queries (the RMQ problem) is NP-hard. A natural question is whether restricting users to MDR queries makes the problem easier. Unfortunately, this is not the case. We show that this problem remains NP-hard even when restricted to MDR queries (the MDQ problem). The result is based on the intuition that given any core C_0 and any set of sum-two queries $S_0 \subseteq \mathcal{Q}_t(C_0)$, we can find another core C_1 and a set of MDR queries $S_1 \subseteq \mathcal{Q}_d(C_1)$, such that the maximal safe subset of S_1 gives the maximal safe subset of S_0 in polynomial time. Consequently, MDQ problem is also NP-hard.

Theorem 1. *The MDQ problem is NP-hard.*

Restricted MDQ Problem. Knowing that the MDQ problem is NP-hard, is it possible to reduce the complexity with further restrictions? We consider *data cubes*, a special class of MDR queries originally defined in [18]. We illustrate some of the concepts of data cubes in Example 2. The following Corollary 1 shows that the MDQ problem remains NP-hard even when it is restricted to those special MDR queries.

Example 2. In Table 3, the two *1-star cuboids* are $\{q^\star((1,1),(1,4)), q^\star((2,1),(2,4))\}$ and $\{q^\star((1,1),(2,1)), q^\star((1,2),(2,2)), q^\star((1,3),(2,3)), q^\star((1,4),(2,4))\}$. The only

Table 4. An Example Showing $\mathcal{Q}_e$ Not Equivalent to $\mathcal{Q}_e \cap \mathcal{Q}_t$ using the same C in Table 3.

$\mathcal{Q}_e$	$q^\star((1,1),(1,2)), q^\star((1,2),(1,3)), q^\star((2,2),(2,3)), q^\star((2,3),(2,4))$ $q^\star((1,2),(2,2)), q^\star((1,3),(2,3)), q^\star((1,2),(2,3)), q^\star((1,1),(2,4))$
$\mathcal{Q}_e \cap \mathcal{Q}_t$	$\mathcal{Q}_e \setminus \{q^\star((1,2),(2,3))\} \cup \{q^\star((1,1),(2,4))\}$

$$\boldsymbol{q^\star((1,1),(2,4)) \not\preceq_d \mathcal{Q}_e \cap \mathcal{Q}_t}$$

2-star cuboid is a singleton set $\{q^\star((1,1),(2,4))\}$. The *data cube* is the union of the three cuboids, which also includes all *skeleton queries*.

Corollary 1. *The problem MDQ remains NP-hard under the restriction that the given set of MDR queries must be: 1. a set of skeleton queries; 2. a union of some cuboids; or 3. a data cube.*

5 Compromisability of Even MDR Queries

This section investigates the compromisability of *even MDR queries* (that is, MDR queries involving even number of tuples). First, in Section 5.1 we show that the set of even MDR queries is equivalent to a subset of sum-two queries (that is, sets of two tuples). Based on this equivalence, the compromisability of even MDR queries can be efficiently determined. In Section 5.2 we show that answering any odd MDR query in addition to even MDR queries leads to compromises, and any odd MDR query is different from the union of a few even MDR queries by only one tuple. We also show that the compromisability of arbitrary queries can be efficiently determined given that the even MDR queries are safe.

5.1 Equivalence between MDR Queries and Sum-Two Queries

Denote the set of all even MDR queries as $\mathcal{Q}_e$. To efficiently determine the compromisability of the even MDR queries $\mathcal{Q}_e$, we show that there exists a subset $\mathcal{Q}_{dt}$ of sum-two queries $\mathcal{Q}_t$, such that $\mathcal{Q}_{dt} \equiv_d \mathcal{Q}_e$. Then we can determine whether $\mathcal{Q}_e$ is safe by checking if $\mathcal{Q}_{dt}$ is safe. Intuitively, determining the compromisability of $\mathcal{Q}_{dt}$ is easier because by reducing $\mathcal{Q}_e$ to $\mathcal{Q}_{dt}$ we have removed most redundant queries.

Two natural but untrue conjectures are $\mathcal{Q}_e \equiv_d \mathcal{Q}_t$ and $\mathcal{Q}_e \equiv_d \mathcal{Q}_e \cap \mathcal{Q}_t$. To see why $\mathcal{Q}_e \equiv_d \mathcal{Q}_t$ is untrue, consider the counter-example with the one-dimensional core $C = \{1,2,3\}$. We have that $q^2(1,3) \in \mathcal{Q}_t$ is not derivable from $\mathcal{Q}_e = \{q^\star(1,2), q^\star(2,3)\}$. Example 3 gives a counter-example to $\mathcal{Q}_e \equiv_d \mathcal{Q}_e \cap \mathcal{Q}_t$.

Example 3. Table 4 shows $\mathcal{Q}_e \not\preceq_d \mathcal{Q}_e \cap \mathcal{Q}_t$ because $q^\star((1,1),(2,4)) \in \mathcal{Q}_e$ is not derivable from $\mathcal{Q}_e \cap \mathcal{Q}_t$.

From Example 3 we see that $\mathcal{Q}_e \not\preceq_d \mathcal{Q}_e \cap \mathcal{Q}_t$ because of even queries such as $q^\star((1,1),(2,4))$. Such an even query is the union of *odd queries* like $q^\star((1,1),(1,3))$ and $q^\star((2,2),(2,4))$. Intuitively, suppose that from $\mathcal{Q}_e \cap \mathcal{Q}_t$ we can derive each odd query up to the *last tuple*. Then we *pair* the adjacent last tuples of all the odd queries by adding other sum-two queries to $\mathcal{Q}_e \cap \mathcal{Q}_t$. Hence, we can derive the even query with these additional sum-two queries. Conversely, these additional sum-two queries can be derived from $\mathcal{Q}_e$ by reversing this process. We demonstrate this in Example 4 and generalize the result in Theorem 2.

Example 4. In Example 3, we can let $\mathcal{Q}_{dt} = \mathcal{Q}_e \cap \mathcal{Q}_t \cup \{q^2((1,3),(2,4))\}$. Consequently, we derive $q^\star((1,1),(2,4))$ as the union of $q^2((1,1),(1,2))$, $q^2((2,2),(2,3))$ and $q^2((1,3),(2,4))$. Conversely, $q^2((1,3),(2,4)$ can be derived as $q^\star((1,1),(2,4)) \setminus (q^2((1,1),(1,2)) \cup q^2((2,2),(2,3)))$. Hence, now we have $\mathcal{Q}_e \equiv_d \mathcal{Q}_{dt}$.

Theorem 2. *For any core C, there exists $\mathcal{Q}_{dt} \subseteq \mathcal{Q}_t$ such that $\mathcal{Q}_e \equiv_d \mathcal{Q}_{dt}$ holds.*

The proof of Theorem 2 [30] includes a procedure that constructs $\mathcal{Q}_{dt}$ by calling a subroutine *Sub_QDT* for each even MDR query $q^\star(u_0, v_0)$. *Sub_QDT* adopts a divide-and-conquer approach in pairing the tuples in $q^\star(u_0, v_0)$. Intuitively, we view each MDR query as an axis-parallel box. At the first stage, *Sub_QDT* recursively divides the current j-dimensional box into $(j-1)$-dimensional boxes, until single tuples are returned as zero-dimensional boxes. Then at the second stage, suppose the current box $q^\star(u,v)$ is j-dimensional; *Sub_QDT* pairs every two tuples returned by the $(j-1)$-dimensional boxes (that $q^\star(u,v)$ has been divided into). If $q^\star(u,v)$ contains even number of tuples, then all of them can be properly paired and *null* is returned to the $(j+1)$-dimensional box. Otherwise, the returned tuple t from the last $(j-1)$-dimensional box cannot be paired and is returned by $q^\star(u,v)$.

Graph Representation and Complexity Analysis. The time complexity of building $\mathcal{Q}_{dt}$ using *Sub_QDT* is $O(mn)$, where $m =\mid \mathcal{Q}_e \mid$ and $n =\mid C \mid$. Because $\mid \mathcal{Q}_{dt} \mid \leq \mid \mathcal{Q}_t \mid \leq \binom{|C|}{2}$ and $m = O(\binom{|C|}{2})$, we have $\mid \mathcal{Q}_{dt} \mid = O(m)$. Hence, no more storage is required by $\mathcal{Q}_{dt}$ than by $\mathcal{Q}_e$.

For any $S \subseteq \mathcal{Q}_{dt}$, we use $G(C,S)$ for the undirected simple graph having C as the vertex set, S as the edge set, and each edge $q^2(t_1,t_2)$ incident the vertices t_1 and t_2. We call $G(C,\mathcal{Q}_{dt})$ the *QDT Graph*. It has been shown in [7] that a set of sum-two queries is safe iff the corresponding graph is a bipartite graph (that is, a graph with no cycle containing an odd number of edges). This can easily be decided with a breadth-first search (BFS) on $G(C,\mathcal{Q}_{dt})$, taking time $O(n+ \mid \mathcal{Q}_{dt} \mid) = O(m+n)$. Hence, the complexity of determining the compromisability of $\mathcal{Q}_e$ is dominated by the construction of $\mathcal{Q}_{dt}$, which is $O(mn)$. Notice that from Section 4 we know that directly applying the condition of Audit Expert [10] has the complexity of $O(m^2 n)$. Therefore, our solution is more efficient than Audit Expert with respect to MDR queries.

Example 5. Example 3 has the cycle composed of $q^2((1,3),(2,3))$, $q^2((2,3),(2,4))$, and $q^2((1,3),(2,4))$ in G_{dt}. Hence, G_{dt} is not a bipartite graph and $\mathcal{Q}_{dt}$ (and hence $\mathcal{Q}_e$) is not safe.

5.2 Beyond Even MDR Queries

Characterizing the QDT Graph. We give some properties of the QDT graph in Lemma 1 that are useful for the rest of this section. The first property shown in Lemma 1 is straightforward. The second property is based on the intuition that if any two tuples t_1,

t_2 in the core are not *close enough* (i.e., $q^\star(t_1,t_2) \notin \mathcal{Q}_{dt}$), then we can find another tuple $t_3 \in q^\star(t_1,t_2)$, such that $q^\star(t_1,t_2) \in \mathcal{Q}_{dt}$ and t_3 is closer to t_1 than t_2. If $q^\star(t_1,t_3) \notin \mathcal{Q}_{dt}$, we repeat this process. This process can be repeated less than $\mid q^\star(t_1,t_2) \mid$ times, and upon termination we have a tuple that is close enough to t_1. The third claim is a natural extension of the first two.

Lemma 1. *1.* $\mathcal{Q}_e \cap \mathcal{Q}_t \subseteq \mathcal{Q}_{dt}$.
2. For any $t_1, t_2 \in C$ *satisfying that* $\mid q^\star(t_1,t_2) \mid > 2$, *there exists* $t_3 \in q^\star(t_1,t_2)$ *such that* $q^\star(t_1,t_3) \in \mathcal{Q}_{dt}$.
3. $G(C, \mathcal{Q}_{dt})$ *is connected.*

Properties of $\boldsymbol{\mathcal{Q}_{dt}}$. Although we have shown that $\mathcal{Q}_{dt} \equiv_d \mathcal{Q}_e$, $\mathcal{Q}_{dt}$ may not be the smallest or the largest subset of $\mathcal{Q}_t$ that is equivalent to $\mathcal{Q}_e$. The smallest subset can be obtained by removing all of the cycles containing even number of edges from $G(C, \mathcal{Q}_{dt})$. If $\mathcal{Q}_e$ is safe, we then have a spanning tree of $G(C, \mathcal{Q}_{dt})$, which corresponds to a set of linearly independent row vectors in the incidence matrix. On the other hand, we are more interested in the maximal subset of $\mathcal{Q}_t$ that is equivalent to $\mathcal{Q}_e$. According to Lemma 2, a safe $\mathcal{Q}_e$ essentially allows users to sum any two tuples from different color classes of $G(C, \mathcal{Q}_{dt})$, and to subtract any two tuples of the same color. The maximal subset of $\mathcal{Q}_t$ equivalent to $\mathcal{Q}_e$ is hence the complete bipartite graph with the same bipartition of $G(C, \mathcal{Q}_{dt})$.

Lemma 2. *Given that* $\mathcal{Q}_e$ *is safe, let* (C_1, C_2) *be the bipartition of* $G(C, \mathcal{Q}_{dt})$ *and* $\mathcal{Q}^\star_{dt} = \{q^2(u,v) : u \in C_1, v \in C_2\}$. *We have that*

1. $\mathcal{Q}^\star_{dt} \equiv_d \mathcal{Q}_{dt}$.
2. *For any* $S \subseteq \mathcal{Q}_t$, *if* $S \equiv_d \mathcal{Q}_{dt}$ *then* $S \subseteq \mathcal{Q}^\star_{dt}$.
3. *For any* $t_1, t_2 \in C_1$ *(or* $t_1, t_2 \in C_2$ *), there exists* $r \in \mathbb{R}^{|\mathcal{Q}_{dt}|}$ *such that* $\mathcal{M}(t_1) - \mathcal{M}(t_2) = r \cdot \mathcal{M}(\mathcal{Q}_{dt})$.

Odd MDR Queries. Now that we can determine the compromisability of $\mathcal{Q}_e$, we would like to know if anything else can be answered safely. First, we consider odd MDR queries that form the complement of $\mathcal{Q}_e$ with respect to all MDR queries $\mathcal{Q}_d$. Intuitively, feeding any odd MDR query $q^\star(u_0, v_0)$ into *Sub_QDT* as the input gives us a single tuple t. Suppose $q^\star(u_0, v_0)$ is a j-dimensional box. It can be divided into two j-dimensional boxes excluding t, together with a $(j-1)$-dimensional box containing t. We can recursively divide the $(j-1)$-dimensional box in the same way. Hence, $q^\star(u_0, v_0)$ is the union of a few disjointed even MDR queries together with a singleton set $\{t\}$. This is formally stated in Corollary 2.

Corollary 2. *Given* $d \in \mathbb{R}^k$, $F = \mathcal{F}(d)$, $C \subseteq F$, *and any* $q^\star(u,v) \in \mathcal{Q}_d \setminus \mathcal{Q}_e$ *satisfying* $\mid \{i : u[i] \neq v[i]\} \mid = j$, *there exists* $q^\star(u_i, v_i) \in \mathcal{Q}_e$ *for all* $1 \leq i \leq 2j-1$, *such that* $\mid q^\star(u,v) \setminus \bigcup_{i=1}^{2j-1} q^\star(u_i,v_i) \mid = 1$ *and* $q^\star(u_i,v_i) \cap q^\star(u_l,v_l) = \phi$ *for all* $1 \leq i < l \leq 2j-1$.

Example 6. In Table 4, using $q^\star((1,1),(2,3))$ as the input of *Sub_QDT* gives the output $(1,3)$. $q^\star((1,1),(2,3))$ can be divided into $q^\star((1,1),(1,3))$ and $q^\star((2,2),(2,3))$. $q^\star((1,1),(1,3))$ can be further divided into $q^\star((1,1),(1,2))$ and $\{(1,3)\}$. Hence, we have $q^\star((1,1),(2,3)) = q^\star((1,1),(1,2)) \cup q^\star((2,2),(2,3)) \cup \{(1,3)\}$

Corollary 2 has two immediate consequences. First, no odd MDR query is safe in addition to $\mathcal{Q}_e$. Equivalently, any subset of $\mathcal{Q}_d$ with $\mathcal{Q}_e$ as its proper subset is unsafe. Second, any odd MDR query is different from the union of a few number of even MDR queries by only one tuple. This difference is negligible because most users of MDR queries are interested in patterns and trends instead of individual values.

Arbitrary Queries. We know the implication of $\mathcal{Q}_e$ in terms of sum-two queries from Lemma 2. Hence, we can easily decide which arbitrary queries can be answered in addition to a safe $\mathcal{Q}_e$. Corollary 3 shows that any arbitrary query can be answered iff it contains the same number of tuples from the two color classes of $G(C, \mathcal{Q}_{dt})$. This can be decided in linear time in the size of the query by counting the tuples it contains. The compromisability of odd MDR queries hence becomes a special case of Corollary 3, because no odd MDR query can satisfy this condition.

Corollary 3. *Given that $\mathcal{Q}_e$ is safe, for any $q \subseteq C$, $q \preceq_d \mathcal{Q}_e$ iff $\mid q \cap C_1 \mid = \mid q \cap C_2 \mid$, where (C_1, C_2) is the bipartition of $G(C, \mathcal{Q}_{dt})$.*

6 Unsafe Even MDR Queries

In this section we consider the situations where even MDR queries are unsafe. We show the equivalence between subsets of even MDR queries and sum-two queries, and give a sufficient condition for the safe subsets.

We have seen in Section 4.2 that finding maximal safe subsets of queries is infeasible even for queries of restricted form, such as sum-two queries and data cubes. Hence, we turn to large but not necessarily maximal safe subsets that can be found efficiently. Recall that in Section 5 we were able to efficiently determine the compromisability of $\mathcal{Q}_e$ because of $\mathcal{Q}_e \equiv_d \mathcal{Q}_{dt}$. If we could establish the equivalence between their subsets, we would be able to extend the results in Section 5 to those subsets. However, equivalence does not hold for arbitrary subsets of $\mathcal{Q}_e$ or $\mathcal{Q}_{dt}$, as shown in Example 7.

Example 7. Consider $\mathcal{Q}_{dt}$ of Example 4. Let $S_{dt} = \mathcal{Q}_{dt} \setminus \{q^2((1,1),(1,2))\}$. Suppose $S_{dt} \equiv_d S_e$ for some $S_e \subseteq \mathcal{Q}_e$. Because $q^\star((1,3),(2,4)) \preceq_d S_e$, S_e must contain $q^\star((1,1),(1,2))$, but then $q^\star((1,1),(1,2)) \not\preceq_d S_{dt}$, a contradiction. Hence, S_{dt} is not equivalent to any subset of $\mathcal{Q}_e$. Similarly, $\mathcal{Q}_e \setminus \{q^\star((1,1),(1,2))\}$ is not equivalent to any subset of $\mathcal{Q}_{dt}$.

Intuitively, any MDR query can be viewed as a *sub-core*. The equivalence given in Theorem 2 must also hold for this sub-core as the following. The even MDR queries defined in the sub-core are equivalent to the sum-two queries added to $\mathcal{Q}_{dt}$ by *Sub_QDT* with those even MDR queries as its inputs. This result can be extended to any subset of the core, as long as the subset can be represented as the union of some sub-cores. Given any $S \subseteq \mathcal{Q}_e$, if we delete each $q^\star(u,v) \in \mathcal{Q}_e \setminus S$ from the core then the result must be the union of some sub-cores. Similarly, given any $S \subseteq \mathcal{Q}_{dt}$, for each $q^2(u,v) \in \mathcal{Q}_{dt} \setminus S$, if we delete $q^\star(u,v)$ from the core then the result is the union of some sub-cores. In this way, the equivalence between subsets of $\mathcal{Q}_e$ and subsets of $\mathcal{Q}_{dt}$ can always be established. This is formalized in Proposition 1.

Proposition 1. *1. Given any $S \subseteq \mathcal{Q}_e$, let $S_e = S \setminus \{q^\star(u,v) : \exists q^\star(u_0,v_0) \in \mathcal{Q}_e \setminus S, q^\star(u,v) \cap q^\star(u_0,v_0) \neq \phi\}$ and $S_{dt} = \{q^2(u,v) : \exists q^\star(u_0,v_0) \in S_e, q^2(u,v) \in \mathcal{Q}_{dt}$ due to $q^\star(u_0,v_0)\}$. Then $S_e \equiv_d S_{dt}$.*

2. Given any $S \subseteq \mathcal{Q}_{dt}$, let $S_e = \mathcal{Q}_e \setminus \{q^\star(u,v) : \exists (u_0,v_0), q^2(u_0,v_0) \in S \wedge q^\star(u,v) \cap q^\star(u_0,v_0) \neq \phi\}$, and $S_{dt} = \{q^2(u,v) : \exists q^\star(u_0,v_0) \in S_e, q^2(u,v) \in \mathcal{Q}_{dt}$ due to $q^\star(u_0,v_0)\}$. Then $S_{dt} \equiv_d S_e$.

Proposition 1 guarantees the equivalence at the cost of smaller subsets. In some situations, we are satisfied with the weaker result, such as $S_{dt} \succeq S_e$ for some $S_e \subseteq \mathcal{Q}_e$. Because then if S_{dt} is safe, then S_e must also be safe, although the converse is not always true. The result in Proposition 2 is similar to Corollary 3 but gives only the sufficient condition. In Proposition 2, S_e can be found by examining each query in $\mathcal{Q}_e$ against the bipartition (C_1, C_2), taking time $O(mn)$, where $m = \mid \mathcal{Q}_e \mid$ and $n = \mid C \mid$.

Proposition 2. *For any $S_{dt} \subseteq \mathcal{Q}_{dt}$, let (C_1, C_2) be the bipartition of $G(C, S_{dt})$. Then $S_{dt} \succeq S_e$ holds, where $S_e \subseteq \mathcal{Q}_e$ satisfies that for any $q^\star(u,v) \in S_e$, $\mid q^\star(u,v) \cap C_1 \mid = \mid q^\star(u,v) \cap C_2 \mid = \mid q^\star(u,v) \mid /2$ holds.*

By Proposition 2 we can efficiently find a safe subset S_e of $\mathcal{Q}_e$ if a safe subset S_{dt} of $\mathcal{Q}_{dt}$ is given. The ideal choice of S_{dt} should maximize $\mid S_e \mid$. This is equivalent to computing the *combinatorial discrepancy* of the set system formed by C and $\mathcal{Q}_e$ [3]. The alternative approach is to maximize $\mid S_{dt} \mid$, which is equivalent to finding the maximal bipartite subgraph of $G(C, \mathcal{Q}_{dt})$.

Instead of those solutions that may incur high complexity, we can apply a simple procedure given in [16]. It takes the graph $G(C, \mathcal{Q}_{dt})$ as the input and outputs a bipartite subgraph. It starts from an empty vertex set and empty edge set and processes one vertex at each step. The unprocessed vertex is colored blue if at least half of the processed vertices to which it connects are red. It is colored red, otherwise. Any edge in the original graph is included in the output bipartite subgraph if it connects two vertices in different colors. The procedure terminates with a bipartite graph $G(C, \mathcal{Q}_{ds})$ satisfying that $\mid \mathcal{Q}_{ds} \mid \geq \mid \mathcal{Q}_{dt} \mid /2$. The procedure runs in $O(n^2) = O(m)$, where $n = \mid C \mid$ and $m = \mid \mathcal{Q}_e \mid$. Our ongoing work will address the effectiveness of this procedure through empirical results.

7 Discussion

A novel three-tier inference control model was proposed for OLAP systems in [29]. The results given in Sections 5 and 6 fit in this model perfectly. In this section, we briefly justify this claim but leave out more details due to space limitations.

The Three-Tier Inference Control Model of [29]. The objectives of the three-tier inference control model are to minimize the performance penalty of inference control methods and to make inference control less vulnerable to undetected external knowledge. This is achieved by introducing a new tier, *aggregation tier A*, to the traditional two-tier view (i.e., *data tier D* and *query tier Q*) of inference control. The three tiers are related by $R_{AD} \subseteq A \times D$, $R_{QA} \subseteq Q \times A$, and $R_{QD} = R_{AD} \circ R_{QA}$. The aggregation tier A satisfies three conditions. First, $\mid A \mid$ is comparable to $\mid D \mid$. Second, there exists partition $\mathcal{P}$ on A such that the composition of R_{AD} and the equivalence relation decided by $\mathcal{P}$ gives a partition on D. Finally, inferences are eliminated in the aggregation tier A.

The three-tier model gains its advantages through its three properties. Because $\mid A \mid$ is relatively small (suppose $\mid Q \mid >> \mid D \mid$), controlling inferences of A is easier than that of Q because of the smaller input to inference control methods. Because of the second property of A, inference control can be *localized* to the R_{AD}-related blocks of A and D, which further reduces the complexity. Moreover, any consequences of undetected external knowledge in some blocks are confined to these blocks, making inference control more *robust*. Finally, as the most expensive task of three-tier inference control, the construction of A can be processed off-line (i.e., before any query arrives). Because decomposing queries into pre-computed aggregations is a built-in capability in most OLAP systems, the online performance overhead of three-tier inference control is almost negligible.

Applicability of Our Results. Partitions of data sets based on the dimension hierarchies naturally compose the data tier. Each block in the partition corresponds to a core. The safe $\mathcal{Q}_{dt}$ (or its safe subsets S_{dt} if it is unsafe) composes each block of the aggregation tier. The query tier includes any arbitrary query derivable from the aggregation tier. If we characterize $\mathcal{Q}_e$ using the row vectors in $\mathcal{M}(\mathcal{Q}_e)$, then the query tier is the linear space they span. The relations R_{AD} and R_{QA} are both the derivability relation $\preceq_d$ given in Definition 3, and $R_{QD} = R_{AD} \circ R_{QA}$ is a subset of $\preceq_d$, because $\preceq_d$ is transitive.

In Section 5 we showed that $\mid \mathcal{Q}_{dt} \mid = O(n^2)$, where $n = \mid C \mid$, satisfying the first condition of the three tier model. Because $\mathcal{Q}_{dt}$ is defined separately on each core, the aggregation tier has a natural partition corresponding to the partition of the data tier, satisfying the second condition. The last condition is satisfied because we use the safe subsets of $\mathcal{Q}_{dt}$ when it is unsafe. Hence by integrating our results on the basis of the three tier model, we inherit all the advantages including negligible online performance overhead, and the robustness in the face of undetected external knowledge (that is, the damage caused by undetected external knowledge is confined to blocks of the partition of data tier and aggregation tier).

Moreover, our results provide better usability to OLAP systems than the cardinality-based approach in [29] does. Firstly, the cardinality-based conditions become invalid when MDR queries other than those contained in the data cube (i.e., skeleton queries) are answered. In this paper we allow any MDR queries if only they are safe. The MDR queries generalize data cubes and various data cube operations, such as slicing, dicing, roll up and drill down. Our answers to even MDR queries are precise, and the answered even MDR queries closely approximate the restricted odd ones. Secondly, when a data cube is unsafe, it is simply denied in [29]. However, in this paper we are able to give partial answers to an unsafe set of even MDR queries, implying better usability. Our methods for computing the partial answers are also efficient. Thirdly, we use necessary and sufficient conditions to determine safe even MDR queries, while the cardinality-based conditions are only sufficient. Therefore, we can provide more answers to users without privacy breaches than the methods of [29] do.

8 Conclusion and Future Direction

In this paper we have shown the infeasibility of applying several existing restrictions to MDR queries. We then proved the equivalence between the even MDR queries and a special set of sum-two queries. On the basis of this equivalence we are able to efficiently determine the compromisability of even MDR queries. We showed that the restricted odd MDR queries are closely approximated by the answered even ones. We showed that safe arbitrary queries can be efficiently determined. We can also maintain this equivalence when even MDR queries are unsafe. Our on-going work implements the proposed algorithms in order to explore their fine tunings. Another future direction is to investigate the aggregation operators other than SUM.

Acknowledgements

The authors are grateful to the anonymous reviewers for their valuable comments.

References

1. N.R. Adam and J.C. Wortmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 439–450, 2000.
3. J. Beck and V.T. Sós. Discrepancy theory. In R.L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of combinatorics*, pages 1405–1446. Elsevier Science, 1995.
4. L.L. Beck. A security mechanism for statistical databases. *ACM Trans. on Database Systems*, 5(3):316–338, 1980.
5. L. Brankovic, M. Miller, P. Horak, and G. Wrightson. Usability of compromise-free statistical databases. In *Proceedings of ninth International Conference on Scientific and Statistical Database Management (SSDBM '97)*, pages 144–154, 1997.
6. A. Brodsky, C. Farkas, D. Wijesekera, and X.S. Wang. Constraints, inference channels and secure databases. In *the 6th International Conference on Principles and Practice of Constraint Programming*, pages 98–113, 2000.
7. F.Y. Chin. Security in statistical databases for queries with small counts. *ACM Transaction on Database Systems*, 3(1):92–104, 1978.
8. F.Y. Chin, P. Kossowski, and S.C. Loh. Efficient inference control for range sum queries. *Theoretical Computer Science*, 32:77–86, 1984.
9. F.Y. Chin and G. Özsoyoglu. Security in partitioned dynamic statistical databases. In *Proc. of IEEE COMPSAC*, pages 594–601, 1979.
10. F.Y. Chin and G. Özsoyoglu. Auditing and inference control in statistical databases. *IEEE Trans. on Software Engineering*, 8(6):574–582, 1982.
11. L.H. Cox. Suppression methodology and statistical disclosure control. *Journal of American Statistical Association*, 75(370):377–385, 1980.
12. D.E. Denning and P.J. Denning. Data security. *ACM computing surveys*, 11(3):227–249, 1979.
13. D.E. Denning, P.J. Denning, and M.D. Schwartz. The tracker: A threat to statistical database security. *ACM Trans. on Database Systems*, 4(1):76–96, 1979.
14. D.E. Denning and J. Schlörer. Inference controls for statistical databases. *IEEE Computer*, 16(7):69–82, 1983.

15. D. Dobkin, A.K. Jones, and R.J. Lipton. Secure databases: protection against user influence. *ACM Trans. on Database Systems*, 4(1):97–106, 1979.
16. P. Erdös. On some extremal problems in graph theory. *Isarel Journal of Math.*, 3:113–116, 1965.
17. L.P. Fellegi. On the qestion of statistical confidentiality. *Journal of American Statistic Association*, 67(337):7–18, 1972.
18. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, crosstab and sub-totals. In *Proceedings of the 12th International Conference on Data Engineering*, pages 152–159, 1996.
19. D.T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *Proceedings 1997 ACM SIGMOD International Conference on Management of Data*, pages 73–88, 1997.
20. J. Kleinberg, C. Papadimitriou, and P. Raghavan. Auditing boolean attributes. In *Proc. of the 9th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 86–91, 2000.
21. Y. Li, L. Wang, and S. Jajodia. Preventing interval based inferece by random data perturbation. In *Proceedings of The Second Workshop on Privacy Enhancing Technologies (PET'02)*, 2002.
22. Y. Li, L. Wang, X.S. Wang, and S. Jajodia. Auditing interval-based inference. In *Proceedings of the 14th Conference on Advanced Information Systems Engineering (CAiSE'02)*, pages 553–568, 2002.
23. Y. Li, L. Wang, S.C. Zhu, and S. Jajodia. A privacy enhanced microaggregation method. In *Proceedings of the Second International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2002)*, pages 148–159, 2002.
24. F.M. Malvestuto and M. Mezzini. Auditing sum queries. In *Proceedings of the 9th International Conference on Database Theory (ICDT'03)*, pages 126–146, 2003.
25. J.M. Mateo-Sanz and J. Domingo-Ferrer. A method for data-oriented multivariate microaggregation. In *Proceedings of the Conference on Statistical Data Protection'98*, pages 89–99, 1998.
26. S. Rizvi and J.R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th Conference on Very Large Data Base (VLDB'02)*, 2002.
27. J. Schlörer. Security of statistical databases: multidimensional transformation. *ACM Trans. on Database Systems*, 6(1):95–112, 1981.
28. J.F. Traub, Y. Yemini, and H. Woźniakowski. The statistical security of a statistical database. *ACM Trans. on Database Systems*, 9(4):672–679, 1984.
29. L. Wang, D. Wijesekera, and S. Jajodia. Cardinality-based inference control in sum-only data cubes. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS'02)*, pages 55–71, 2002.
30. L. Wang, D. Wijesekera, and S. Jajodia. Olap means on-line anti-privacy. Technical Report, 2003. Available at http://ise.gmu.edu/techrep/2003/.
31. L. Wang, D. Wijesekera, and S. Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. Technical Report, 2003. Available at http://ise.gmu.edu/techrep/2003/.

Passive Attack Analysis for Connection-Based Anonymity Systems

Andrei Serjantov and Peter Sewell

University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue
Cambridge CB3 0FD, United Kingdom
First.Last@cl.cam.ac.uk

Abstract. In this paper we consider low latency connection-based anonymity systems which can be used for applications like web browsing or SSH. Although several such systems have been designed and built, their anonymity has so far not been adequately evaluated.
We analyse the anonymity of connection-based systems against passive adversaries. We give a precise description of two attacks, evaluate their effectiveness, and calculate the amount of traffic necessary to provide a minimum degree of protection against them.

1 Introduction

Systems for anonymous interaction are a basic building block for application-level privacy. The anonymity properties these systems aim to provide are subtle: in contrast to most security protocols, they must cover statistical traffic analysis attacks. A number of anonymity systems have been designed, starting from [Cha81]. They can be divided into two classes:

– Message-based (mix) systems, for asynchronous (email) messages. They provide anonymity by delaying and mixing messages; email can tolerate substantial delay. There is a significant body of work on their design [Cot94,GT96,DDM03] and implementation [MC00,DDM02].
– Connection-based systems, for low-latency bidirectional communication (e.g. SSH connections and web browsing). There are several implemented designs [GRS99,RP02,FM02]. Although these are also sometimes called mix systems, current designs do not do any mixing as such, so we choose not to use this term in the paper.

Analysis of these systems is crucial: users need more than a "warm fuzzy feeling" that they are anonymous. For message-based systems, we have well-understood threat models and both qualitative [BPS00,Ray00] and (some) quantitative [SD02,SDS02] analysis. For connection-based systems, on the other hand, the threats are harder to characterise – the low-latency constraint makes these systems vulnerable to powerful timing attacks. Qualitative analyses include [BMS01]. Quantitative analysis has so far been limited to evaluating the impact of compromised nodes on the anonymity provided. [STRL00,WALS02].

In this paper we examine the protection such systems can provide against a passive attacker (i.e. one who watches the network rather than tries to compromise nodes). Some systems, MorphMix [RP02] for example, explicitly state that they are vulnerable to such an adversary; our work may provide insight into ways of strengthening them. Others, notably Tarzan [FM02], employ an expensive dummy traffic policy in an effort to protect against this adversary. We show that it is possible to avoid this.

It is important to note that in this paper we consider connection-based anonymity systems which are running on top of standard Internet protocols, i.e. packet-switched networks. We do not consider schemes over circuit-switched networks like [PPW91].

2 Systems and Usage

We begin by outlining the application scenario, high-level anonymity goals, and system architecture that we consider in our analysis. The latter is a distillation of the key choices of Onion Routing, Tarzan and MorphMix [GRS99,FM02,RP02].

Scenario. We are primarily considering systems for anonymous web browsing. A number of *users*, running anonymity clients, connect through the system to some *web servers* (not running special software). HTTP requests and responses both travel through the system.

System goals. Such a system should:

1. provide usable web browsing, with no more than a few seconds additional latency; and
2. make it hard for an attacker to determine what any given user is browsing[1]. In particular, as we discuss below, it should protect a user against an attacker who can observe all traffic on the path of their connection. Note that to do this, the adversary does not need to observe all traffic in the anonymity system.

The goals clearly involve a trade-off: the more delay, the higher the (potential) anonymity.

Architecture. We make some basic assumptions about the system structure:

– The system consists of a number of *nodes*. Some designs have a 'classic' architecture, with relatively few nodes, whereas others have a 'P2P' architecture, with nodes run by each user. Each node has logical *links* to some (not necessarily all) other nodes, along which it forwards traffic. Links are implemented above inter-node TCP connections between IP/port addresses, link-encrypted. To protect against node compromise, each connection passes through several nodes. Nodes also accept connections from end-user clients.

[1] The system need not protect against the attacker determining *that* the user is browsing, or which web servers are being accessed through the anonymity system. The system should, of course, protect against the webserver determining who is browsing the website, unless it is compromised by an application-level feature (e.g. cookies).

- To protect against the simple passive observer, who can bitwise compare traffic entering and leaving a node, traffic is onion-encrypted (as first suggested in [Cha81]). This also protects against some node compromise attacks.
- The length of messages remains observable, so the data is divided into fixed-length *cells*. Typically these are small (in the Onion Routing design each cell carries 128 bytes of data).
- "Onion connection" setup is expensive, so each client/server communication is routed via the same sequence of nodes. (Application proxying may reduce the number of communications, e.g. fetching all objects of a webpage in one communication.)
- Routes may be chosen either by the end-user or by the network.

This architecture broadly follows the design of 2nd generation Onion Routing [ord], Tarzan [FM02], and MorphMix [RP02]. Some of our results are also applicable to JAP [JAP] and the Freedom Network [BGS00].

Adding *dummy traffic* is a standard technique used in message-based anonymity systems e.g. Mixmaster. For connection-based systems, however, practical experience shows the bandwidth requirements of nodes are large; the additional cost of dummies must be minimised. Accordingly, in this paper we assume that inter-node links do not involve dummies (though it turns out to be beneficial to apply some padding to the links between the client and the first node). We leave for future work the question of how a given quantity of dummy traffic can be most effectively used.

3 Threat Models

Prior work on threat models for connection-based systems has focused on the threat of malicious nodes, looking at how anonymity is affected by the fraction of attacker-controlled nodes [Shm02,STRL00].

In this paper we focus on the threat of traffic analysis by a *passive* observer. Earlier notions of "global passive" attacker, as used in analysis of message-based systems, are too vague for connection-based systems. The threats must be stated more precisely: the quality (time accuracy) of the traffic data available to different global passive attackers may vary considerably, making different traffic analyses possible. We leave analysis of *active* attacks to future work.

There are several different low-level mechanisms an attacker might use to obtain traffic data, differing in the quality of data they make available, and in the effort required.

- Attacker-controlled nodes. Outside our scope.
- By applying legal (or sublegal) pressure to an ISP, a high-resolution traffic monitor can be installed on a machine on the same collision domain as a node. This could capture all IP packets travelling to and from other nodes, with precise (sub-millisecond) timestamps; that data could be forwarded on-line to an analysis machine. Note that if nodes are distributed among judicial domains, it is hard to attack a substantial proportion of them.

- By compromising a machine in the same collision domain as a node the same data could be captured, though here there may be difficulties in surreptitiously forwarding it to the analyser.
- By installing non-intrusive fibre taps 'in the field', on the fibres that carry traffic between nodes [Hod91], one can capture similar data, but here, as there are typically routers between a node and an external fibre, some timing accuracy will be lost (several router delay variances). How many such attackers are required to intercept all node-to-node communications depends on the topology, but typically examining just backbone fibres will not suffice.
- Traffic data can also be obtained by compromising a router on each of the inter-node links and placing traffic monitoring code there. However, here the attacker is more likely to get per link packet counts (over large fractions of a second) rather than per-packet data with timestamps. These can be retrieved via the standard SNMP protocol. More accuracy can be obtained by compromising routers closer to each node.

Broadly, all these attackers gain access to the same class of data – the number of packets that travel between pairs of nodes (on anonymity-system logical links) during particular time intervals. The packet counting interval determines what kinds of traffic analysis the attacker can perform: taking long intervals amounts to low-pass filtering of the data, erasing informative high-frequency components.

A further distinction is between *per-interval* and *waveform* analysis. In the former, each packet-counting interval is treated separately – the attacker can forget the data for each interval after it has been analysed – whereas in the latter a substantial region of the traffic waveform must be considered. The latter may obviously be more expensive to mount.

4 Analysis: Lone Connection Tracking

Our first analysis is based on packet counting. We recall that traffic travels down a connection in small cells. Consider a node in the system. During a particular time interval the number of packets on each of the connections travelling through it is highly likely to be different. This attack requires the delay introduced by the node to be small, compared to the size of the time interval, so the number of incoming and outgoing packets of the node on each connection will be very similar. They will not be identical as some packets will have been in the node before the interval started and some will remain after the interval ends.

A passive attacker can observe the number of packets of a connection which arrive at the node and leave the node only if this connection is *lone* – it is the only one travelling down that link – on its incoming and outgoing links during the time interval. This scenario is illustrated on Figure 1. It is clear that the numbers of packets on links from D to the node X and from X to T are very similar, so the attacker can be confident that the connection(s) from D have been forwarded to T. Naturally, there is a possibility that he is mistaken: one of the connections from A, B or C carried 1079 packets and was forwarded to T, while the connection(s) from D was forwarded to Q, R or S. However, the

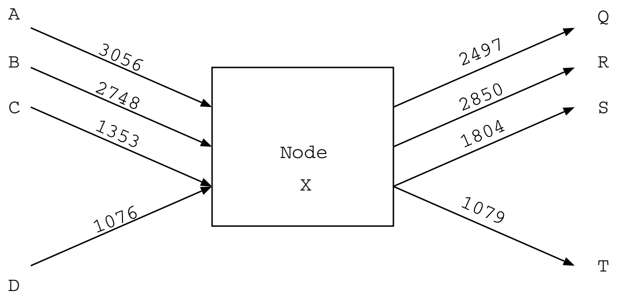


Fig. 1. Each arrow represents an incoming or an outgoing link. The number on the arrow represents the number of packets observed by an attacker on the link over one time period.

probability of this is very small (we do not calculate it here) as we assume that the number of packets on each of the incoming connections is highly likely to be different. We can further reduce this probability by doing the same observations during a different time interval and checking the results are consistent.

Note that this attack does not require a global packet counting attacker, merely one who observes all the links a connection travels on.

This attack is based on assumptions, some of which we have touched on already:

– The packet counting interval is much larger than the mix delay. This is necessary as otherwise the packets inside the mix at the starts and ends of packet counting intervals will make the incoming and outgoing packet counts dissimilar.
– The packet counting interval is much smaller than the mean time between new connections being set up on this pair of links. The longer the time interval, the more likely there is to be a new connection initiated which will traverse an incoming or an outgoing link, thereby ruining the packet counts and thus the attack. Note that if the adversary is unable to obtain packet counts for short enough time periods (e.g. due to extracting packet counts via SNMP), he loses some opportunity for attacks.

It may seem that the attacker can just as easily count packets coming in from the users to the first node of the connection and try to correlate this with the packet counts coming out of the anonymity system to the webservers[2]. Such an attack is less likely to succeed (and easier to protect against) for the following reasons:

– In the description of the attack on a single node, we required that the packet counting interval should be much larger than the node delay. Thus, in the case of mounting the attack on the anonymity system as a whole, the packet

[2] This is the extreme version of packet counting across several nodes.

counting interval will have to be made much larger than the delay on all the nodes of a connection together plus all the link delays. This increases the chances of the user initiating another connection to the same first node, thereby confusing the packet counts. Implementors of connection based systems should note that this is a good and cheap defence strategy (though it relies on the first node being uncompromised).

– A small amount of padding between the user and the first node protects against the attack. This requires much less bandwidth than padding each link in the anonymity system as suggested by [Ren03]. Of course, it would be desirable to also pad the link from the last node to the webserver, but this is impossible as the webserver is not running anonymity software.
– It may be possible to arrange the first link not to be observable by the attacker (e.g. run a node at the edge of a corporate network and ensure that everyone inside the company uses it as their first node).

Having shown that lone connections allow the attacker to compromise anonymity, we now calculate how many such connections a system is likely to have under different conditions (and thus how likely a user's connection is to be compromised). First, we derive an approximation, and then examine the subject in more detail using a simulator. Finally we suggest ways of defending against this attack.

4.1 Mean-Based Analysis

Assume the users initiate on average c connections per second, each forwarded along ℓ links (inside the network) and that there are n nodes in the anonymity system. Furthermore assume each connection has duration dur.

Thus on average at any instant there are $c \times dur$ connections. Each connection exists on ℓ links, so on average there are $c \times dur \times \ell$ link-occupancies. If there is a link between each pair of nodes, there are roughly $n \times n$ links[3]. On average there are $c \times dur \times \ell/(n \times n)$ connections per link. It is clear that the absolute lower bound of the number of connections per link is 1, and for a good anonymity system this number should be much greater.

Let us illustrate this with an example. Suppose we have a system with $n = 30$ nodes, the users initiate connections through $\ell = 3$ network links (or 4 nodes), each lasting $dur = 2$ seconds. If each node can talk to every other, then around 150 connection initiations per second are necessary for this system to provide at least some anonymity.

4.2 Definitions

It is clear that the approximations calculated in the previous section are rather crude. We now proceed to define lone connections formally and show how to work out the fraction of lone connections of a particular system.

[3] It is debatable whether routes with the same node occurring twice consecutively should be allowed.

First, define the anonymity system graph as a set of nodes G with $|G| = n$ and a set of edges (links) E, with each edge being a pair of nodes. A path (a connection), then, is a sequence of edges. Take all connections c_i (of length l_i) which are open during a particular packet counting interval and let $g = \{\!|[e_{1,1} \dots e_{1,l_1}]; [e_{2,1} \dots e_{2,l_2}]; \dots |\!\}$ be the multiset of paths of these connections[4]. We can easily express the number of connections on each link resulting from such a configuration.

$$f_g(e) = \sum_{p \in g} \text{occurrences of } e \text{ in } p$$

A connection is lone when it is lone on all the links it is going through. Now calculating the set of lone connections in a configuration is straightforward:

$$\mathsf{lone} = |\{\!|p|p \in g \wedge \forall e \in p.f_g(e) = 1|\!\}|$$

We can also find the fraction of lone connections: $\frac{\mathsf{lone}}{|g|}$.

We now go on to define the probability of a connection going through the anonymity system being lone.

First, let us assume some parameters of the anonymity system.

- $\Phi(c)$, the probability that c connections go through the anonymity system during the same interval.
- $\Psi(j)$, the probability that a route going through it is chosen to have length j.
- The graph of the anonymity system is given by G, E.
- The maximum number of connections which can go through a system is $\mathsf{max\_c}$ and the maximum route length is $\mathsf{max\_rt}$.

Now define $\mathcal{G}([l_1, \dots, l_c])$, to be the set of all multisets of paths of lengths $[l_1, \dots, l_c]$ in the graph G, E.

$$\mathcal{G}([l_1, \dots, l_c]) = \{m | m = \{\!|[e_{1,1} \dots e_{1,l_1}]; [e_{2,1} \dots e_{2,l_2}]; \dots ; [e_{c,1} \dots e_{c,l_c}]|\!\} \wedge \\ e_{x,y} \in m \Rightarrow e_{x,y} \in E\}$$

Now, the probability P of a particular connection being unmixed is:

$$P = \sum_{c \in 0 \dots \mathsf{max\_c}} \Phi(c) \times \sum_{\substack{L=[l_1,\dots,l_c] \\ \forall i.l_i \leq \mathsf{max\_rt}}} \prod_{l \in L} \Psi(l) \times \sum_{g \in \mathcal{G}(L)} \frac{|\{p|p \in g \wedge \forall e \in p.f_g(e) = 1\}|}{|g|}$$

There are several assumptions which we have made implicitly in the above formula. Firstly, the probability distribution for path lengths of all connections is the same, $\Psi(l)$. Secondly, paths are chosen uniformly at random from all the possible paths (of length l) through the graph G, E [5]. Finally, all the path lengths

[4] Paths can be identical, so we tag them with a unique integer.

[5] Hence the number of connections a node will handle is determined by the number of links it has to other nodes.

and all the paths themselves are chosen by the connection initiators independently.

Although the above formula defines the probability of a connection going through an anonymity system unmixed, it is hard to see the quantitative implications of it directly. We therefore make a simulation of the anonymity system.

4.3 Simulator Results

We have constructed a simulator which uses the definitions above to calculate the fraction of lone connections. Given a graph of nodes connected by links, and the number of connections we wish to simulate, it picks a route length for each connection ($\Psi(j)$ is assumed to be a uniform distribution between a minimum and a maximum value) and then generates the routes themselves. Then it calculates the fraction of lone connections (using the definitions above). Clearly, the fraction of lone connections going through the network is also the probability that a particular user's connection is going to be observed by the global packet counting attacker.

For example, let us take a peer to peer anonymity system with 100 nodes (each user running a node) all connected to each other. Suppose each of the 100 users initiates a connection ($\Phi(100) = 1$ during the packet counting interval we are considering) through a minimum of 2 and a maximum of 4 network links. This system, provides very low anonymity – around 92% of the connections going through it are lone.

A graph of the number of nodes vs the probability of connection compromise is shown in Figure 2. There are 60 connections going through the network and each connection is going through 2 network links.

It is worth noting that the fraction of lone connections is not the only measure of anonymity we could have used. Indeed, although it conveys a very clear message to the user (the probability of the connection they are about to establish being observable), it also suffers from some disadvantages. First, it does not indicate how many other connections a particular connections has been mixed with as an anonymity set (or the information theoretic metric of [SD02]) does. It is worth pointing out that if a connection is lone on some, but not all of its links, its anonymity set is very much reduced (which is not reflected in the probability of it being compromised). Secondly, the designers of the anonymity system might like to know the probability of any one or more connections being compromised – a much stronger property. We leave calculating these metrics and analysing the attack in detail to (rather tedious) future work.

4.4 Protection

As we saw in the previous section, the packet counting attack on the lone connections is quite powerful against some systems. Here we examine ways of protecting against it.

Firstly, more traffic (and/or fewer nodes) makes the system much less vulnerable to the attack. Modifying the system from the example above to one with 20

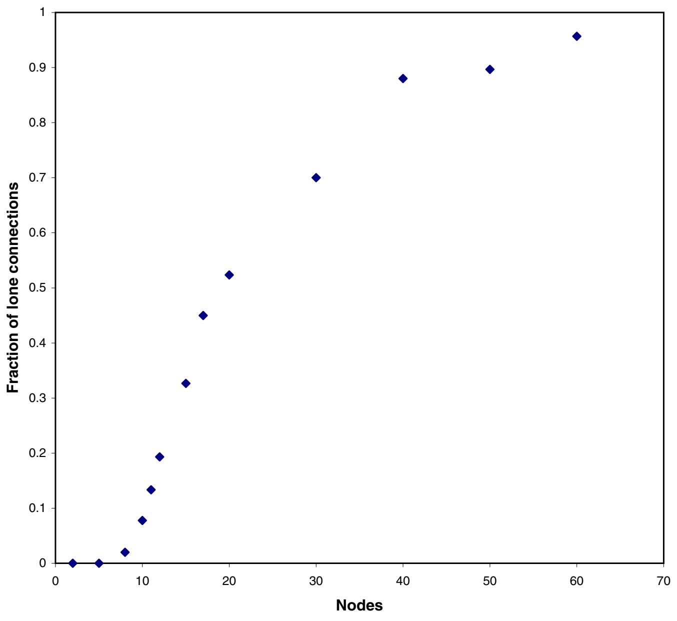


Fig. 2. Graph of the number of nodes in an anonymity system vs the fraction of lone connections.

nodes with 200 connections going through it (and keeping the route length the same at between 2 and 4 links) reduces the fraction of compromised connections from 92% to 2.5%.

Secondly, increasing the route length helps increase the total volume of traffic in the network, but also has the undesirable effect of increasing latency. For example, doubling the route length in our example above (100 nodes, 100 connections, route length of 4 to 8 network links) reduces the probability of a connection being compromised from around 92% to 72%. The graph showing how route length affect the fraction of lone connections is show in Figure 3.

Thirdly, and most importantly, we can design the architecture of the system to suit the amount of traffic we expect to flow through it. If there is very little traffic, a cascade ought to be used. If there is slightly more, a restricted route architecture (see [Dan03]) can be employed to dramatically decrease the fraction of lone connections. For instance, for an anonymity system of 100 nodes, each able to forward traffic to 5 others (with route length of between 2 and 4 links and 100 connections), the fraction of lone connections is reduced to around 17%. This is still, however, unacceptable and suggests that making every client run a node is not a good choice for a strong anonymity system.

The fact that peer to peer anonymity systems do not provide much anonymity against the global passive attacker is an important conclusion which may seem counterintuitive. However, it can be easily explained when we think of the fact that the connections have to "fill up" the network graph at least twice. If we

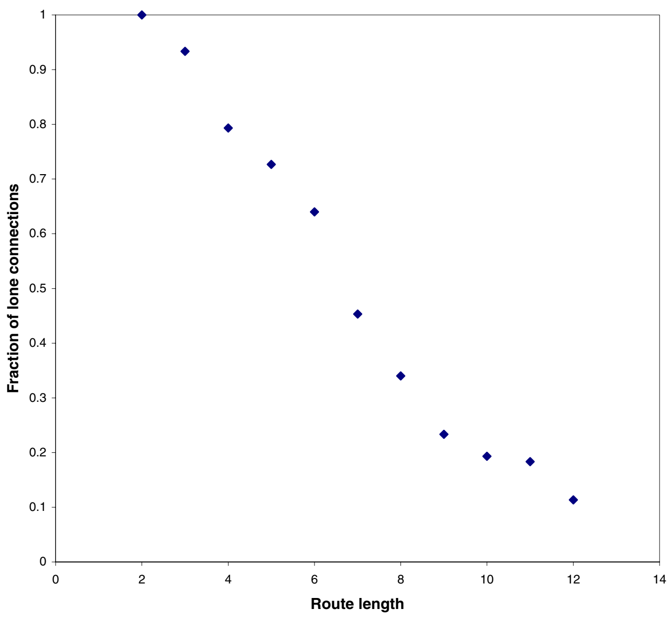


Fig. 3. Graph of the route length vs the fraction of lone connections.

run a node at every client, the number of connections will be within a constant factor of number of nodes, kn. The number of links in a fully connected network roughly n^2, so the anonymity of such a network will be low. This is clear even from the crude calculations in Section 4.1.

As well as designing the system in a way which suits the expected level of traffic, we need to be able to handle daily or weekly variations in the number of connections established through the system. This may be possible by dynamically reconfiguring the network topology from cascade to restricted routes to full network. Of course, short term (daily) variations in traffic could not be handled in this way; it should be handled by suggesting to the users the number of nodes their connections should go through to stay anonymous.

5 Analysis: Connection-Start Tracking

Our second attack is based on tracking the increase in the volume of traffic from an incoming to an outgoing link of a node which results from data starting to flow on a new connection. This increase happens when the webserver is starting to send the webpage data in response to a request made by a client. We call such an increase a "connection start". We note that the propagation of such a connection start through a node is observable to a packet counting attacker, even if the connection is not lone. If the node does not delay traffic significantly (as current systems do not), the attacker will observe a node in a steady state;

a start of connection arriving followed by a start of connection leaving, and will deduce where the new connection has come from and been forwarded to.

Hence, nodes must delay traffic (but still provide low latency communications). The most appropriate mixing strategy here is the SG-Mix of Kesdogan (see [KEB98]). It is easy to analyse, handles each packet separately and does not rely on batching. We proceed to describe this mix and examine how it can help us protect against the above attack.

The SG-Mix mix treats each packet (cell) independently. When a cell arrives, the mix draws a random value from an exponential distribution with parameter μ and delays the cell by that time. The mean delay is of the mix is thus $1/\mu$.

Assume the users initiate (on average) c connections per second, each going through ℓ nodes. The system consists of n nodes. Write λ for the mean rate of arrival of starts of connections (per second) to a particular node. We have $\lambda = c\ell/n$.

Assume further that the arrivals of the starts of connections to the node are Poisson distributed (with parameter λ).

Now, the attacker tracks a connection through a mix iff:

1. When the start of the connection arrives, the mix is "empty of starts of connections". This means that there has not been an incoming start of connection not followed by an outgoing one.
2. Having arrived on an incoming link, the start of the connection leaves the mix whilst no other start of a connection has arrived.

This is essentially the $n-1$ attack scenario described in [KEB98], though performed here for starts of connections instead of individual asynchronous messages. We want to choose the parameters λ and μ such that the probability of the attacker tracking a start of connections through all the mixes is small.

First, consider the probability that a connection is tracked through one node.

$$\frac{e^{-\frac{\lambda}{\mu}}}{1+\frac{\lambda}{\mu}}$$

The probability of the attacker tracking a particular connection which is going through ℓ mixes is:

$$\left(\frac{e^{-\frac{\lambda}{\mu}}}{1+\frac{\lambda}{\mu}}\right)^{\ell}$$

substituting in the expression for λ from above gives:

$$\left(\frac{e^{-\frac{c\ell}{n\mu}}}{1+\frac{c\ell}{n\mu}}\right)^{\ell}$$

A user of this system would incur a delay of roughly $2\ell/\mu$ seconds for onion connection setup, ℓ/μ for a request and ℓ/μ for a response, or a total *connection delay* of: $4\ell/\mu$.

We will now do some order-of-magnitude calculations to see how much traffic needs to go through the anonymity system to get acceptable delay and anonymity values.

Clearly, as long as $\ell \geq 3$ and $cl/nm \geq 1$, the probability of tracking a connection is low (< 0.006). Hence, $c\ell/n\mu \geq 1$ or $c\ell \geq n\mu$.

Suppose $\ell = 3$ and the maximum acceptable delay is 2 seconds, hence $4\ell/\mu = 2$, hence $\mu = 2\ell = 6$. Substituting in, we get $c \geq 2n$. This implies that the users of the system have to initiate twice as many connections per second as there are nodes.

Suppose we have U users browsing every day. If each browses 100 pages a day, $c = 100/(3600 \times 24)$

Now suppose we have an anonymity system of 30 nodes. We find the number of users U needed for the system to provide anonymity. $U \times 100/(3600 \times 24) \geq 2 \times 30$, or $U \geq 2 \times 30 \times 36 \times 24 = 51000$. This is a realistic target for a small to medium size anonymity system.

Naturally, these calculations are rather crude as they involve the mean amount of traffic (and suppose that the traffic is evenly distributed throughout the day). More traffic is required to to protect against traffic troughs (e.g. at night).

It is worth considering the quality of traffic data the adversary has to have access to to mount such an attack. If a timestamp for every packet is available, the attack can be mounted with maximum effectiveness. However, if the adversary can only do packet counting over some time intervals, then the time interval must be much longer than the node delay and much smaller than the interarrival times of starts of connections *to a node*. Note that this is more precision than was required for the lone connections attack (the time interval there had to be much less than the interarrival times of connections *on a single link*).

5.1 Working with Richer Traffic Features

Before we considered starts of connections and showed that if these are allowed to propagate through the network, then a certain level of traffic is required to maintain anonymity. Now we consider how the attacker could use more general traffic features (spikes) to track individual connections. This is an example of a waveform analysis – data from several intervals will be required.

Let us consider a simple case of a node with 2 incoming and 2 outgoing links. The adversary sees a spike on one of the incoming links (say from A) and one of the outgoing links (to Q) some time later. He knows that both the links which exhibited spikes have lone connections on them[6], but the other links (from B and to R) contain many connections, so some spikes may be hidden. The smart adversary does not jump to conclusions about a correlation between the links with spikes, but instead calculates the probability of it.

There are two possibilities: Either the attacker is correct and the spike from A really went to Q, or the attacker is mistaken and there was a hidden spike

[6] This constraint is easily relaxed, we include it for clarity of exposition

which came in from B and went to Q, while the spike from A got forwarded to R and hidden in the traffic.

The probability of the former is 1/2 (assuming the connection from A was equally likely to be forwarded to Q and R). The probability of the latter is $P(\text{spike}) \times 1/2$. Hence, the attacker is correct with probability $\frac{1}{1+P(\text{spike})}$. The probability of a spike occurring on a link is low, so the attacker of the attacker being correct is high.

A complete analysis is not presented here – we would need to examine real connection traffic to determine the probability of spikes occurring and determine what other kinds of traffic features we might make use of. It is notable that interactive applications like SSH are much more vulnerable to this kind of attack than web browsing as the traffic is much less uniform. In general one would expect to use signal-processing techniques – filtering the signal to frequency ranges known to include identifiable features and/or calculating running correlations between signals and common feature patterns. We leave this for future work.

6 Discussion and Solutions

In the previous sections we looked at two powerful attacks which can be mounted on connection-based anonymity systems by passive attackers, quantitatively evaluated their effectiveness and assessed potential protection measures.

Unlike all previous analyses, we stayed clear of using vague and costly proposals of adding dummy traffic to the system, instead calculating the amount of user connections required to maintain anonymity. This approach is crucial for building efficient, fast and therefore deployable connection based anonymity system, whilst still providing anonymity to the users.

However, we have not examined all the attacks which the adversaries can potentially mount against connection-based anonymity systems. In particular, in this paper we have not considered the “first and last node” attack or any active attacks. We comment upon them briefly here.

The “first and last node” attack involves the attacker compromising the first and the last node of a particular connection. He can now filter padding from the client to the first node (if there was any) and modify traffic travelling in both directions. In particular, he can insert a signal into the inter-arrival times of cells of a particular connection and then look for it (low-pass filtered to account for the variances in network and mix delays) on the other side. As the packets are small, the signal is likely to carry a substantial amount of information and help the attacker succeed. Note that an active attacker who can modify traffic on links (but has not compromised any nodes has the same capability).

There are several potential countermeasures which will help make this attack less powerful. First, longer routes will help reduce the amount of signal which propagates from the first to the last node. Secondly, increasing the packet size (and thus decreasing the number of packets) will help reduce the size of the signal which can be inserted into the connection. In the limit, if all webpages fit into one packet, active attacks become ineffective (though this comes with a massive efficiency loss).

We also briefly mentioned traffic shaping as a countermeasure to the "lone connections" attack. It is worth noting that such a traffic shaping policy would have to make all the connections in the anonymity system have the same profile, which is likely to be expensive in terms of introducing delays or bandwidth (dummy traffic). We have not investigated this mostly because protection against the attacks outlined could be achieved by cheaper means.

One of the implications of the results presented here is that (peer to peer) anonymity systems which involve all the users running nodes are impractical simply because there is not enough traffic to fill all the links. Therefore, it is evident that adding nodes provides *less* anonymity (contrary to popular belief) against the global passive attacker. Whether this statement is true for the case of partial attackers remains the subject of future work.

Another interesting line of research is to see whether an anonymity system could be built using an architecture which elevates a subset of the P2P nodes to a "supernode" status. The role of such supernodes would be to perform mixing, while the other, "lesser" nodes would simply forward traffic. This may yield a good compromise between the performance and scalability of current P2P architectures and resistance against passive attackers.

7 Related Work

As mentioned before, there is relatively little quantitative analysis of connection-based anonymity systems. The notable exception is [STRL00] which gives a detailed account of the security of the first generation of the Onion Routing system against compromised nodes.

To the best of our knowledge, the first work which describes the packet counting attack is the analysis by Back, Möller and Stiglic [BMS01], however, they fail to point out the crucial requirement of the connection being lone on its link. (Although under some conditions we may be able to decompose the sums of numbers of packets on different links uniquely into the number of packets belonging to each connection, this is unlikely to be important in practice). Another recent work [Ren03] analyses packet counting attacks but remains vague about the assumptions on node delay and details of connections travelling on links, and proposes a constant dummy traffic policy which turns out to be costly.

The connection-start tracking problem was observed in [PPW91]; there it is solved by dividing the connection into a number of "time slice channels". This scheme requires extra overhead, and its effectiveness remains to be evaluated.

There are also systems which provide anonymous connections for web browsing [SBS02] which do not follow the "mix" architecture of Chaum, but they also lack quantitative analyses of the anonymity provided.

8 Conclusion

We examined in some detail two attacks which can be mounted by passive adversaries on connection-based anonymity systems. These compromise existing anonymity systems completely. However, the threats can be analysed and can be protected against without resorting to dummy traffic and keeping the delay to users' connections acceptable.

We note that these threats to connection-based anonymity systems (some of which are currently in the process of being implemented and deployed) are practical and realistic, and the designers should take them into account, especially as the methods of protection need not be costly.

Finally, this paper shows that quantitative analysis of connection-based anonymity systems is just as feasible as of message-based ones. Furthermore, such analysis is required to develop and evaluate methods of protection against real threats. As a promising direction for future work, we suggest that mounting real attacks on implemented (and deployed) anonymity systems will provide further insight into the measures necessary to keep anonymity systems anonymous.

Acknowledgements

We acknowledge support from a Royal Society University Research Fellowship (Sewell), EPSRC grant GRN 24872 and EC FET-GC project IST-2001-33234 PEPITO. We also thank Tuomas Aura, Richard Clayton, George Danezis, Andreas Pfitzmann, Marc Rennhard, Paul Syverson for thoughtful comments.

References

BGS00. P. Boucher, I. Goldberg, and A. Shostack. Freedom system 2.0 architecture. `http://www.freedom.net/info/whitepapers/`, 2000. Zero-Knowledge Sytems, Inc.

BMS01. A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding Workshop*, pages 245–257. LNCS 2137, 2001.

BPS00. O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In *Workshop on the Design Issues in Anonymity and Observability*, LNCS 2009, pages 10–29. 2000.

Cha81. D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

Cot94. L. Cottrell. Mixmaster and remailer attacks, 1994. `http://www.obscura.com/~loki/remailer/remailer-essay.html`.

Dan03. G. Danezis. Mix-networks with restricted routes. In *Privacy Enhancing Technologies*, LNCS 2760. Dresden, Germany, 2003.

DDM02. G. Danezis, R. Dingledine, and N. Mathewson. Type III (Mixminion) Mix Protocol Specifications, 2002. `http://mixminion.net/minion-spec.txt`.

DDM03. G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Security and Privacy*. 2003.

FM02. M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Computer and Communications Security (CCS)*. 2002.

GRS99. D. Goldschlag, M. Reed, and P. Syverson. Onion Routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–41, 1999.

GT96. C. Gülcü and G. Tsudik. Mixing Email with *Babel*. In *Internet Society Symposium on Network and Distributed Sytem Security*, pages 2–16. 1996.

Hod91. H. Hodara. Secure fiberoptic communications. In *Symposium on Electromagnetic Security for Information Protection*. Rome, Italy, 1991.

JAP. The JAP project. `http://anon.inf.tu-dresden.de/index_en.html`.

KEB98. D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Information Hiding Workshop*. LNCS 1525, 1998.

MC00. U. Moeller and L. Cottrell. *Mixmaster Protocol Version 3*, 2000. `http://www.eskimo.com/~rowdenw/crypt/Mix/draft-moeller-v3-01.txt`.

ord. Onion Routing developers mailing list. `http://archives.seul.org/or/dev/`.

PPW91. A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, pages 451–463. 1991.

Ray00. J. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Workshop on the Design Issues in Anonymity and Observability*, LNCS 2009, pages 10–29. 2000.

Ren03. M. Rennhard. Practical anonymity for the masses with mix-networks. Technical Report 157, ETH Zurich,Switzerland, 2003.

RP02. M. Rennhard and B. Plattner. Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Workshop on Privacy in the Electronic Society (WPES)*. Washington, DC, USA, 2002.

SBS02. R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE Security and Privacy*. 2002.

SD02. A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *Privacy Enhancing Technologies*, LNCS 2482. 2002.

SDS02. A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In *Information Hiding Workshop*, LNCS 2578. 2002.

Shm02. V. Shmatikov. Probabilistic analysis of anonymity. In *15th IEEE Computer Security Foundations Workshop*, pages 119–128. 2002.

STRL00. P. F. Syverson, G. Tsudik, M. G. Reed, and C. E. Landwehr. Towards an analysis of Onion Routing security. In *Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009. 2000.

WALS02. M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *ISOC Symposium on Network and Distributed System Security*. 2002.

Rapid Mixing and Security of Chaum's Visual Electronic Voting[*]

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

[1] Institute of Mathematics, Wrocław University of Technology,
ul. Wybrzeże Wyspiańskiego 27
50-370 Wrocław, Poland
{gomulkie,mirekk}@im.pwr.wroc.pl
klonowsk@ulam.im.pwr.wroc.pl
[2] CC Signet

Abstract. Recently, David Chaum proposed an electronic voting scheme that combines visual cryptography and digital processing. It was designed to meet not only mathematical security standards, but also to be accepted by voters that do not trust electronic devices.

In this scheme mix-servers are used to guarantee anonymity of the votes in the counting process. The mix-servers are operated by different parties, so an evidence of their correct operation is necessary. For this purpose the protocol uses *randomized partial checking* of Jakobsson et al., where some randomly selected connections between the (encoded) inputs and outputs of a mix-server are revealed. This leaks *some* information about the ballots, even if intuitively this information cannot be used for any efficient attack.

We provide a rigorous stochastic analysis of how much information is revealed by randomized partial checking in the Chaum's protocol. We estimate how many mix-servers are necessary for a fair security level. Namely, we consider probability distribution of the permutations linking the encoded votes with the decoded votes given the information revealed by randomized partial checking. We show that the variation distance between this distribution and the uniform distribution is $\mathcal{O}\left(\frac{1}{n}\right)$ already for a *constant* number of mix-servers (n is the number of voters). This means that a constant number of trustees in the Chaum's protocol is enough to obtain provable security. The analysis also shows that certain details of the Chaum's protocol can be simplified without lowering security level.

Keywords: electronic voting, mix network, randomized partial checking, Markov chain, rapid mixing, path coupling

1 Introduction

Recently, there have been a lot of discussions about electronic voting. This is caused by the problems with the traditional voting procedures: inevitable errors that occur during counting by humans, unreadable or ambiguous votes, dishonest committees, cases of selling the votes, and very high costs.

[*] partially supported by KBN grant 0 T00A 003 23

Electronic voting may provide accuracy and higher efficiency at a lower cost. However, even though a lot of research on electronic voting have been done, some severe drawbacks have been overlooked for a long time. The problem is that a voter has to trust that the computer that he uses for elections has not been tampered. He would like to receive some kind of a material "receipt" that would convince him that his vote is included in the final outcome. On the other hand, existence of receipts may allow selling votes, which is a severe threat to democratic systems. Second, we do not want anyone to know our vote. Even if the votes are secured with strong cryptography, potentially some side channel information may be used to reveal the voters' preferences (in a simple scheme the time of inserting an encoded vote and the time of publishing a corresponding plaintext of the vote can reveal the voters choice).

Chaum's Electronic Voting Procedure. David Chaum [5] proposes a fairly practical scheme designed to meet the demands mentioned above. The issue of getting the voter's trust is resolved by using ideas of visual cryptography [10]. A voter is given a two layer sheet made of a translucent plastic material - and his vote is clearly visible until the layers are separated. It is up to him whether he chooses to keep the top or the bottom layer as the receipt – both encode his vote safely, and none of them can be read without the other, which is destroyed right after the voter leaves the booth.

All the votes can be safely published for instance on a web page, so each voter can download his vote's image and compare it with his receipt to make sure that nothing wicked has taken place.

Appropriate cryptographic procedures ensure that the vote can be recovered only by cooperating trusted committees. Let us describe the procedure without going into the details which are irrelevant for the rest of the paper (an interested reader is referred to [5]). There are k trustees $\mathcal{C}_1, \ldots, \mathcal{C}_k$. Each vote is encoded so that it must be processed by all trustees before it can be counted. Namely, to get a plaintext T of a ciphertext C each trustee has to apply its decoding function D_i

$$T = D_k\left(D_{k-1}\left(\ldots D_1(C)\ldots\right)\right),$$

where D_i is a decoding function of $\mathcal{C}_i$ depending on a secret value kept by $\mathcal{C}_i$.

So, during the decoding process each trustee $\mathcal{C}_i$ partially decrypts all (partially decrypted) votes received from $\mathcal{C}_{i-1}$, permutes the results at random, and sends the list obtained to trustee $\mathcal{C}_{i+1}$. Of course, without permuting the results an adversary would find the voter's preference by comparing the list of encrypted votes with the list of the plaintexts.

To exclude a possibility that a trustee tampers with the votes, at the end of the decoding procedure, each trustee is obliged to point values of the permutation applied for a half of its input positions. In other words, connection between decrypted (input) and partially decrypted ballots (output) is revealed. This set of positions is chosen at random or by other trustees. Moreover trustee shows all information needed by verification - e.g. random strings used, so other trustees by simple re-encryption can easily check if trustee behaved correctly with pointed ballots. Thanks to this procedure trustee would be catch with high probability if it replaced even a few ballots. This technique, introduced in [9] is called *randomized partial checking*.Due to details of encryption scheme

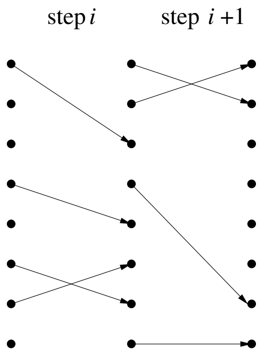


Fig. 1. Connections revealed by a trustee during randomized partial checking

used, it is possible to check that these votes are decoded properly. Therefore, probability that a forgery remains hidden equals $\frac{1}{2}$ for each vote (per stage), so probability that e.g. 50 votes were changed without being detected is negligible (2^{-50}).

It can be easily seen that the procedure described above does not ensure privacy: although it is rather unlikely, there may exist a path consisting of revealed values of consecutive permutations that uncovers the origin of a vote. For that reason, the revealing scheme of randomized partial checking is more sophisticated. Each trustee must perform at least two steps of decoding and permuting so that it could show one half of the first one, and then show "another" half of the second one. More precisely, if π_1, π_2 are the permutations applied by the trustee for the list of $2n$ encoded votes then for a chosen set A (A is a set of n indices) the trustee reveals $\pi_1(i)$ for each $i \in A$ and $\pi_2(j)$ for each $j \notin \pi_1(A)$ (see Fig. 1). In this way it is guaranteed that no path of length 3 can be disclosed by randomized partial checking .

Such a solution has also a weak point. For the sake of simplicity, let us assume that there are only 2 different kinds of votes and call them *black* and *white*. Let a *stage* be the part of processing executed by a single trustee (from now on we assume it consists of two decoding/permuting steps). Assume there are $2n$ votes and exactly one of them is "black". Now, let us consider the stages in the reverse order: The plaintext of the black vote comes out of the last stage. Although we do not know where *exactly* the black vote was before the last stage, we know *for sure* that it was somewhere within certain n positions. Therefore, from the point of view of an external observer for some positions the probability that they hosted the black vote before the last stage equals $\frac{1}{n}$, and for some positions this probability is 0. Then we consider the second last stage – and since it is independent from the last stage it may happen that a lot of positions where the black vote may have been hosted are connected with position inside the same half of another stage. If so, the probability distribution of the black vote's location would be quite far from the uniform distribution over $2n$ positions. Of course, eventually probabilities approach the same value $\frac{1}{2n}$, but we need to go through some number of stages.

Chaum ([5]) proposes the following solution to avoid this problem: each stage is divided into four decoding steps executed by the same trustee. Then the trustee reveals a half of the connections from the first permutation, and "another half" from the second

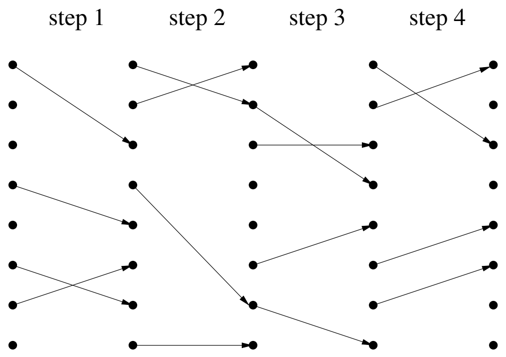


Fig. 2. Revealing connections in a stage consisting of 4 decoding steps

permutation, as described above. In the next step a set B of indices is chosen so that it contains $n/2$ elements from $\pi_2(\pi_1(A))$ and $n/2$ elements from the complement of this set. After that, trustee reveals $\pi_3(i)$ for each $i \in B$ and $\pi_4(j)$ for each $j \notin \pi_3(B)$.

It can be easily seen that this scheme ensures that our "black vote" is distributed uniformly over all $2n$ positions.

Problem Statement. Although the reasoning about a single "black" vote is quite convincing it does not mean that the scheme is secure. It only says that privacy of a single voter is achieved: it does not show automatically that, for instance, an adversary cannot conclude with fair probability that two voters have the same preferences.

What we really need is a much stronger result saying that very little information concerning voting preferences is leaking in the revealing process for any outcome of elections. Let us formulate this demand in terms of probability theory: let Π denote the permutation so that $\Pi(i) = j$ if the ith ciphertext processed to $\mathcal{C}_1$ corresponds to the jth plaintext vote published by $\mathcal{C}_k$. To be perfectly safe, we should prove that Π, conditioned by information obtained from the revealing process, still has a uniform distribution. There is a simple counting argument that shows that it is not possible. However, what we really need is to prove that the probability distribution of Π is close enough to the uniform distribution. It is not clear how many stages are necessary for this purpose. This question has not been resolved by the former work except some informal discussion.

1.1 New Results

Throughout the paper $\mathcal{L}(X)$ denotes probability distribution of a random variable X.

Let Π_i denote the random variable that represents the permutation of the votes after i steps of decoding: $\Pi_i(j) = s$ means that the jth encoded vote (from the list given to $\mathcal{C}_i$) corresponds to the partially decoded vote on position s in the output list of $\mathcal{C}_i$. Our goal is to estimate the size of k such that $\mathcal{L}(\Pi_k)$ is very close to the uniform distribution. We use the standard measure of discrepancy between two probability distributions μ_1

and μ_2 over a finite space Ω, so-called *total variation distance*, defined as follows:

$$\|\mu_1 - \mu_2\| = \frac{1}{2} \sum_{\omega \in \Omega} |\mu_1(\omega) - \mu_2(\omega)| \, .$$

Theorem 1 (Main result) *There exists $T = \mathcal{O}(1)$ such that the variation distance between $\mathcal{L}(\Pi_T)$ and the uniform distribution is $\mathcal{O}\left(\frac{1}{n}\right)$.*

We prove this result for a modified version of the Chaum's protocol in which a stage consists of two instead of 4 decoding steps - which shows that taking 4 steps was an unnecessary complication.

An important (and a little bit unexpected) corollary of Theorem 1 is the following fact:

Corollary 1. *For achieving high security level a constant number of stages is enough no matter how large the population of voters is.*

2 Model

2.1 Decoding Process as a Stochastic Process

The decoding process can be considered as a discrete stochastic process where step i is executed by a trustee C_i independently (in the stochastic sense) from the other trustees. We assume that decoding the votes from the list obtained from C_{i-1} is perfectly secure, that is, for an adversary not knowing the secret key of C_i recoding is a purely random function. Additionally, trustee C_i chooses uniformly at random two permutations: $\eta_{i,1}$ and $\eta_{i,2}$. The outcome of recoding of the first substage is permuted according to $\eta_{i,1}$: the ciphertext from position j is moved to position $\eta_{i,1}(j)$ for $j \le 2n$. Similarly, $\eta_{i,2}$ is used to permute the elements after the second substage. Finally, a set A_i of n indices is chosen uniformly at random and the values $\eta_{i,1}(j)$ for $j \in A_i$ and $\eta_{i,2}(j)$ for $j \notin \eta_{i,1}(A_i)$ and revealed to the public.

Let us consider a passive adversary observing decoding process. Her aim is to break privacy of voting (i.e. she wants to get some knowledge about probability distribution $\mathcal{L}(\Pi_k)$). It is easy to see that from an adversary's point of view the process can be regarded as a process of mixing $2n$ items so that during stage i:

1. the items on positions $j \notin A_i$ are permuted at random,
2. the items on positions $j \in A_i$ are permuted at random,
3. all items are permuted in public.

Of course, set A_i is revealed in this step, so the probability distribution $\mathcal{L}(\Pi_k)$ is a random variable on sets A_i and permutations used at substeps 3.

We depict each substage in a special way: we put all positions from A_i at the top and the rest at the bottom - this does not change anything, since substep 3 is executed.

Let us consider the first substage (see Fig. 3 and 4). Since n elements located on positions from A_1 are permuted at random, they become indistinguishable from an adversary's point of view, and therefore we shall call them Black items. The remaining

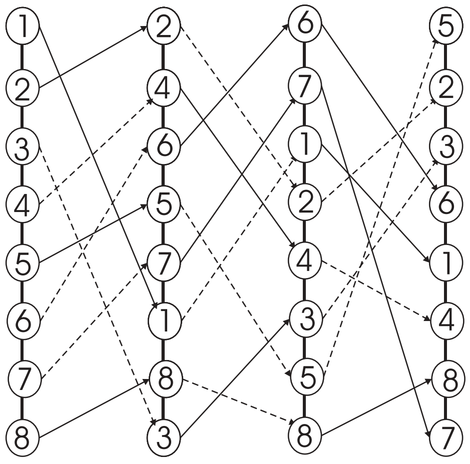


Fig. 3. Permutations used at the first three decoding steps, only solid lines are revealed

items also become indistinguishable, so we call them White items. After the first step an adversary can only determine positions of Black and White items. All other information is hidden from her. It is easy to extend this way of thinking for next stages - in this way we process only black and white items and ask what is their final distribution over $2n$ positions. We show in the next section that we can confine ourselves to Black and White items instead of regarding all votes.

2.2 Permutations of White and Black Items

Reduction to Black and White Items. An external adversary that observes public data on execution of the protocol may try to get some information of voters' preferences. What she can do is at most to compute probability distributions Π_i. Instead of that she may consider a stochastic process starting right after the first decoding step during which the same permutations are applied as to the lists of encoded votes, but instead of the encoded votes she considers permuting Black and White items. Since the permutations are only partially revealed, she may only derive probability distribution over possible configurations of Black and White items after each decoding step. Below we make quite an easy but technically very useful observation that it suffices to consider probability distribution of configuration of the White and Black items in order to show Theorem 1.

Let $\mathbb{P}_n$ be set of possible permutations of n Black and n White items. From now on we consider the permutation of Black/White items immediately after decoding step t as a random variable Υ_t taking values in $\mathbb{P}_n$. Let η_U denote a uniform distribution over $\mathbb{P}_n$.

Each element $p \in \mathbb{P}_n$ corresponds to a subset of $\mathbb{S}_{2n}$. Namely, for $\pi \in \mathbb{S}_{2n}$ we write $\pi \in p$, when $\pi^{-1}(i)$ is Black if and only if $p(i)$ is Black for each $i \leq 2n$. Clearly, for each $p \in \mathbb{P}_n$ there are $n! \cdot n!$ permutations π such that $\pi \in p$.

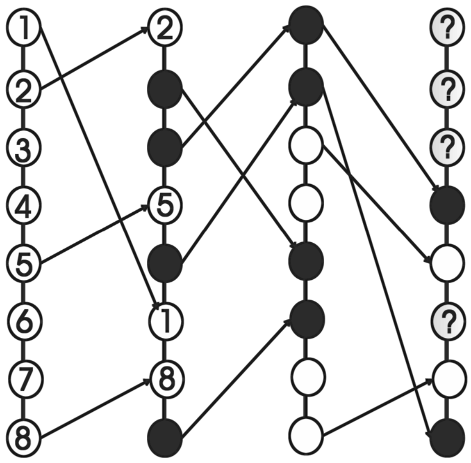


Fig. 4. Adversary view of the first three decoding steps, Black and White items are depicted

The following lemma shows a relationship between random variables Υ_k and Π_k. This relationship simplifies the proof of Theorem 1 and enables applying coupling techniques.

Lemma 1. $\|\Upsilon_k - \eta_U\| = \|\Pi_k - \mu_U\|$.

Proof.

$$\|\Upsilon_k - \eta_U\| = \frac{1}{2}\sum_{p\in\mathbb{P}_n} |\Upsilon_k(p) - \eta_U(p)| \overset{*}{=}$$

$$\frac{1}{2}\sum_{p\in\mathbb{P}_n}\left|\sum_{\pi\in p}\Pi_k(\pi) - \frac{1}{\binom{2n}{n}}\right| \overset{**}{=} \frac{1}{2}\sum_{p\in\mathbb{S}_{2n}}\left|\Pi_k(\pi) - \frac{1}{(2n)!}\right| = \|\Pi_k - \mu_U\| \; .$$

Equations (*) and (**) hold, since for each $p \in \mathbb{P}_n$ all permutations $\pi \in p$, are equally probable. □

From Lemma 1 we see that to prove $\|\Pi_k - \mu_U\|$ is small it suffices to show that $\|\Upsilon_k - \eta_U\|$ is small.

Stationary Distribution of Υ_t. One can see that $\mathbf{M} = (\Upsilon_t)_{t\in\mathbb{N}_+}$ is a time-dependent Markov chain. Moreover η_U is its unique stationary distribution, because for each adjacency matrix P_t of $\mathbf{M}$, η_U is the only probability distribution, that solves equation $xP_t = x$.

So we see that the proof of Theorem 1 reduces to analysis of convergence rate of Markov chain $\mathbf{M}$ - namely, we need to show that this chain has so called *rapid mixing* property.

For technical reasons it will be important that there is a metric function

$$\Delta : \mathbb{P}_n \times \mathbb{P}_n \longrightarrow \{0, 1 \ldots n\} \; .$$

It is defined as follows: for each $p_1, p_2 \in \mathbb{P}_n$ let $\Delta(p_1, p_2)$ is a minimal number of transpositions necessary to go from p_1 to p_2.

Distribution of White and Black Items. We shall use the following technical fact:

Claim 1 *If X is a random variable with hypergeometric probability distribution:*

$$\textbf{\textit{Pr}}\,[X = k] = \frac{\binom{n}{k}\binom{n}{n-k}}{\binom{2n}{n}}$$

Then there exists such n_0 so that for each $n > n_0$

$$\textbf{\textit{Pr}}\left[|X - n/2| > n^{2/3}\right]$$

is negligibly small, that is smaller than $1/n^3$. (It is sound to assume that $n_0 = 100$.)

Proof of Claim 1 is based on Stirling's formula and roughly estimated probability values of hypergeometrical distribution.
From the claim above we get immediately that with high probability the positions of A_i, $i \le k$ contain not less than $n/2 - n^{2/3}$ and no more than $n/2 + n^{2/3}$ Black items. And so, from now on we consider *only* permutations satisfying these conditions.

3 Rapid Mixing via Path Coupling

The methods for showing convergence rate of discrete Markov chains have been developed rapidly over the past decade. We use here one of the newest methods, so-called *path coupling*, a powerful extension of well-known *coupling*. Below we describe briefly coupling and path coupling; further details can be found in [2] and [3].

3.1 Coupling and Path Coupling

Let $\mathbf{M} = (\mathcal{Y}_t)_{t\in\mathbb{N}}$ be a discrete-time (possibly time-dependent) Markov chain with a finite state space $\mathbf{S}$ that has a unique stationary distribution μ. Let $\mathcal{L}_Y(\mathcal{Y}_t)$ denote the probability distribution of $\mathcal{Y}_t$, given that $\mathcal{Y}_0 = Y$. The standard measure of the convergence is *mixing time*, defined as:

$$\tau_{\mathbf{M}}(\varepsilon) = \min\left\{T : \forall Y \in \mathbf{S}, \forall t \ge T\ \|\mathcal{L}_Y(\mathcal{Y}_t) - \mu\| \le \varepsilon\right\} .$$

Coupling. A *coupling* [1] for a Markov chain $(\mathcal{Y}_t)_{t\in\mathbb{N}}$ is a stochastic process $(Y_t, Y_t^\star)$ on the space $\mathbf{S} \times \mathbf{S}$ such that each process Y_t and $Y_t^\star$ considered separately is a faithful copy of $\mathcal{Y}_t$. In other words, $\mathcal{L}_Y(\mathcal{Y}_t) = \mathcal{L}_Y(Y_t) = \mathcal{L}_Y(Y_t^\star)$ for each $Y \in \mathbf{S}$. The *Coupling Lemma* [1], says that

$$\|\mathcal{L}_Y(\mathcal{Y}_t) - \mu\| \le \mathbf{Pr}[Y_t \ne Y_t^\star]$$

for the worst choice of the initial states Y_0 and $Y_0^\star$. So, if we want to show convergence of a Markov chain, we can do this by constructing an appropriate coupling. Of course processes Y_t and $Y_t^\star$ are usually dependent – constructing a proper dependence that forces the chains Y_t and $Y_t^\star$ to converge could be the most difficult part of estimating mixing time.

Path Coupling. Analyzing process $(Y_t, Y_t^\star)$ on whole space $\mathbf{S} \times \mathbf{S}$ can be very cumbersome. Fortunately, Bubley and Dyer [2] introduced *path coupling* – a powerful extension of *coupling* that allows one to consider a coupling only for a particular subset of $\mathbf{S} \times \mathbf{S}$.
Let $\Delta : \mathbf{S} \times \mathbf{S} \longrightarrow \mathbb{N}$ be a metric and let D be the largest distance according to metrics Δ. Further, let

$$\Gamma = \{(Y_t, Y_t^\star) \in \mathbf{S} \times \mathbf{S} : \Delta(Y_t, Y_t^\star) = 1\} \ .$$

In order to use path coupling we need to assume that for all $(Y_t, Y_t^\star) \in \mathbf{S} \times \mathbf{S}$, if $\Delta(Y_t, Y_t^\star) = r$, then there exist a sequence (a "path") $Y = \Lambda_0, \Lambda_1, \ldots, \Lambda_r = Y^\star$ with $(\Lambda_{i-1}, \Lambda_i) \in \Gamma$ for $0 \leq i < r$. In [2] Bubley and Dyer proved the following Path Coupling Lemma (we present here a simplified version):

Lemma 2. *Assume that there exist a coupling $(Y_t, Y_t^\star)$ for process $(\mathcal{Y}_t)_{t\in\mathbb{N}}$ such that for some real $\beta < 1$ we have $\mathbf{E}[\Delta(Y_{t+1}, Y_{t+1}^\star)] \leq \beta$ for all $(Y_t, Y_t^\star) \in \Gamma$ and for all $t \in \mathbb{N}$. Then,*

$$\tau_{\mathbf{M}}(\varepsilon) \leq \lceil \ln(D\varepsilon^{-1}) / \ln \beta^{-1} \rceil \ .$$

In particular, it follows from Path Coupling Lemma that if

$$\mathbf{E}[\Delta(Y_{t+1}, Y_{t+1)}^\star)] \leq 1/n^c$$

for some $c > 0$ and $D = \mathcal{O}(n)$, then

$$\tau_{\mathbf{M}}\left(\tfrac{1}{n}\right) \leq \mathcal{O}(1) \ .$$

4 Security Analysis

In this chapter we prove Theorem 1. Construction of an appropriate coupling is the main technical problem here. Let us note that technicalities of the proof presented here are related to the proofs from papers [6] and [7].

4.1 Path Coupling Construction

According to Lemma 1 it suffices to estimate stopping time of the process $\mathbf{M} = (\Upsilon_t)_{t\in\mathbb{N}_+}$. For this purpose we consider two processes $(\Upsilon_t, \Upsilon_t^\star)$ such that $\Delta\left(\Upsilon_t, \Upsilon_t^\star\right) = 1$ – for such a pair we need to find a proper coupling. Now let us execute a single step t (for $t > 2$) of these processes consisting of a stage of the protocol under consideration. Note that A_t and the public permutation are the same for both processes. Let the positions in A_t be called the *upper half*, and the remaining positions be called the *lower half*. By Claim 1, with overwhelming probability, each half contains at least $n - n^{2/3}$ White and at least $n - n^{2/3}$ Black items.

We shall determine $\Upsilon_t^\star$ depending on Υ_t so that the distance between these two processes does not grow and with overwhelming probability it becomes zero at the next step.

Obviously, it suffices to care about the movements of Black items - the White items fill the remaining places. The Black items that are located at the same positions for Υ_t

and $\Upsilon_t^\star$ are called *regular* Black items, the black items that are on different positions are called extra Black items. According to our assumptions there is one extra Black item for the first process and one extra Black item for the second process.

If the extra Black items are inside the same half, then it is trivial to define a proper coupling: if the first process uses permutation π in this half, then the second process applies $\pi \circ (u, v)$, where (u, v) denotes a transposition on positions u and v of the extra Black items. In the second half the same permutations are used by both processes. Such a choice guarantees that the processes become identical after this step.

The crucial case is when the extra Black items do not belong to the same half. So assume without a loss of generality that for Υ_t the extra Black item is in the upper half and for $\Upsilon_t^\star$ the extra Black item is in the lower half.

Now we define permutations applied by the second process in the lower and in the upper half given the permutations chosen by the first process. It suffices to deal with Black items only.

- the regular Black items are moved in the second process exactly as for the first process,
- the extra Black item in the second process is moved according to a more complicated procedure to be described below.

It is obvious that in this way the distance between Υ_{t+1} and $\Upsilon_{t+1}^\star$ is 1. However, we shall guarantee with high probability that the extra Black items will be in the same half and at the next step the processes become identical. (So in order to use Path Coupling Lemma in the form stated, we can compose a new Markov chain which steps consist of two steps of $\mathbf{M}$.)

Now we shall consider the extra Black items. First let us look at the extra Black item in the upper half: since the permutations are chosen uniformly at random, it is uniformly distributed over all m_1 possible positions that are not occupied by the regular Black items there. Let these positions be called White. By Claim 1 we may assume that

$$n/2 - n^{2/3} < m_1 < n/2 + n^{2/3} .$$

Also by Claim 1 we may assume that

- the number of White positions in the upper half that are connected to the upper half of the next decoding step is a $k_1 \geq m_1/2 - m_1^{2/3}$,
- $k_2 \geq m_1/2 - m_1^{2/3}$ White positions from the upper half are linked to the lower half of the next decoding step.

Similarly, we may assume that in the lower half

- $k_3 \geq m_2/2 - m_2^{2/3}$ White positions are linked with the upper half,
- $k_4 \geq m_2/2 - m_2^{2/3}$ White positions are linked with the lower half.

Now let $k = \min\{k_1, k_2, k_3, k_4\}$. Let $\varepsilon_1 = m_1 - 2k$ and $\varepsilon_2 = m_2 - 2k$. Among all the k_i White positions for each i we choose $4k$ *privileged* positions (see Fig. 5) so that there are

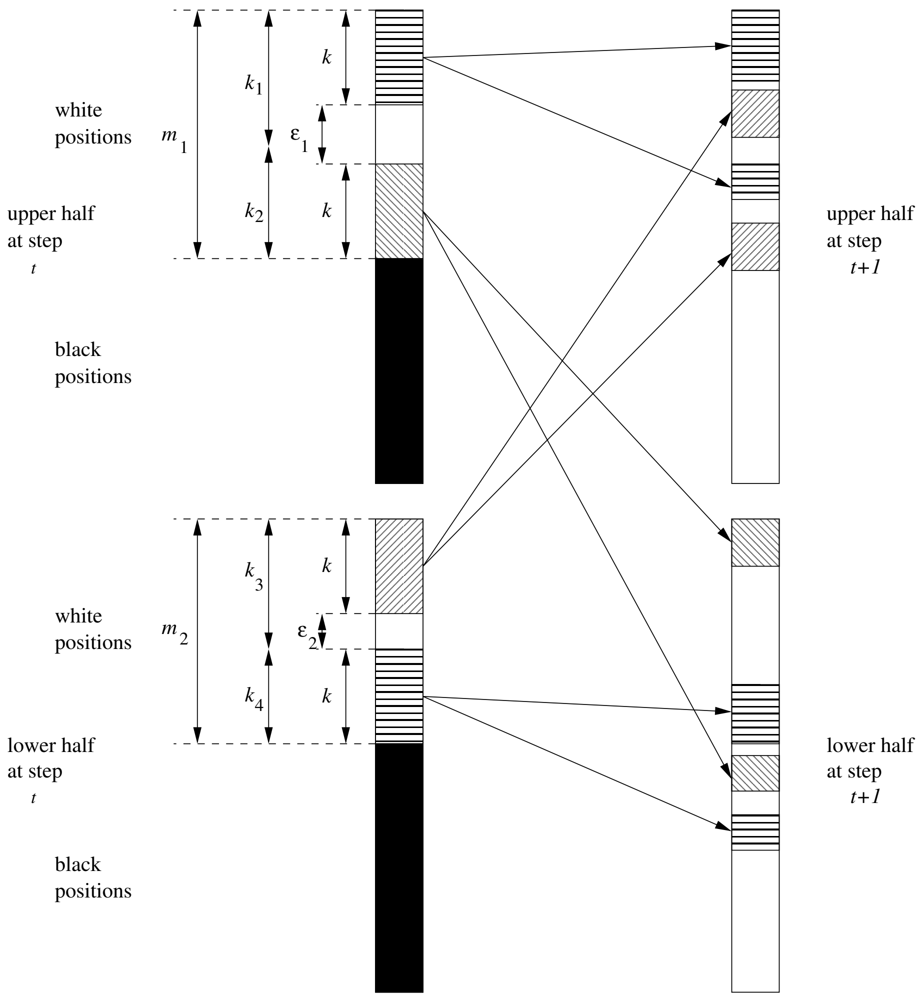


Fig. 5. Classification of White positions

- k privileged positions in the upper half that are linked to the upper half of the next step,
- k privileged positions in the upper half that are linked with the lower half,
- k privileged positions in the lower half that are linked to the upper half of the next step,
- k privileged positions in the lower half that are linked with the lower half.

Finally, there are ε_1 unprivileged positions in the upper half and ε_2 unprivileged positions in the lower half. In general we cannot guarantee to which halves these positions are connected.

Now we look at the extra Black item in the upper half and determine the movement of the extra Black item in the lower half accordingly. No matter what we do, we must guarantee that the extra Black item in the lower half is distributed uniformly over all m_2 White positions in the lower half - otherwise the coupling would be incorrect – the second process would not be a copy of $\mathbf{M}$.

The following cases are possible:

Case A: The extra Black item of the first process is on an unprivileged position (probability ε_1/m_1).

Case B: The extra Black item of the first process is on a privileged position (probability $1 - \varepsilon_1/m_1$).

In case A we choose the position of the extra Black item of the second process in the lower half uniformly at random among all White positions there.

In case B we perform the following steps:

1. we toss a (non-symmetric) coin to decide whether to place the extra Black item from the lower half on an unprivileged position (probability ε_2/m_2) or a privileged one (probability $1 - \varepsilon_2/m_2$).
2. If we have chosen to place the extra Black item on an unprivileged position, we choose such a position uniformly at random.
3. If we have decided to put the extra Black item on a privileged position, we look at the movement of the extra Black item of the first process in the upper half. If it is placed on the jth position linked to the upper half (lower half) at the next step, then we place the extra Black item in the lower half on the jth position linked to the upper half (lower half). In this case we assure that the extra Black items will go to the same half of the next decoding step (so the processes will be coupled during the next decoding step).

4.2 Correctness and Coupling Probability

First observe that the extra Black item in the lower half reaches each White position in the lower half with the same (marginal) probability. Indeed, for any non-privileged position the probability equals:

$$\frac{\varepsilon_1}{2k+\varepsilon_1} \cdot \frac{1}{2k+\varepsilon_2} + \left(1 - \frac{\varepsilon_1}{2k+\varepsilon_1}\right) \cdot \frac{\varepsilon_2}{2k+\varepsilon_2} \cdot \frac{1}{\varepsilon_2} = \frac{1}{2k+\varepsilon_2} = \frac{1}{m_2} .$$

For a privileged position this probability equals:

$$\frac{\varepsilon_1}{2k+\varepsilon_1} \cdot \frac{1}{2k+\varepsilon_2} + \left(1 - \frac{\varepsilon_1}{2k+\varepsilon_1}\right) \cdot \frac{2k}{2k+\varepsilon_2} \cdot \frac{1}{2k} = \frac{1}{2k+\varepsilon_2} = \frac{1}{m_2} .$$

So the coupling is correct.

With probability

$$p \geq \frac{2k}{2k+\varepsilon_1} \cdot \frac{2k}{2k+\varepsilon_2} = \left(1 - \frac{\varepsilon_1}{2k+\varepsilon_1}\right) \cdot \left(1 - \frac{\varepsilon_2}{2k+\varepsilon_2}\right)$$

during one decoding step the extra Black items are placed so that they are in the same half (and the processes get coupled in the next step for sure). Since $\varepsilon_i \leq (2n)^{2/3}$, one can easily show that $p \geq 1 - 16\sqrt[3]{4}\frac{1}{\sqrt[3]{n}}$. Then

$$E\left(\Delta(\Upsilon_{t+2}, \Upsilon^{\star}_{t+2})\right) \leq 16\sqrt[3]{4}\frac{1}{\sqrt[3]{n}} = \beta\ .$$

Thus, according to Path Coupling Lemma

$$\tau_{\mathbf{M}}(\varepsilon) \leq \left\lceil \ln(D\varepsilon^{-1})/\ln\beta^{-1} \right\rceil$$

$$\tau_{\mathbf{M}}\left(\tfrac{1}{n}\right) \leq \left\lceil 2\ln n/\tfrac{1}{3}\ln n - \ln 4 \right\rceil = O(1)\ .$$

This concludes the proof of Theorem 1.

5 Conclusions

We provide a rigorous proof that using mix-networks for Chaum's electronic elections meet high level demands on privacy: the connection between the plaintext votes and their ciphertexts remains almost purely random. This is a strong argument for using such a scenario in practice, provided that all technical problems (special printers and so) are solved. Furthermore, without loosing *privacy* we can divide whole mix-cascade into rounds including two mixing steps, instead of grouping into batches of four (as it was originally proposed in [5]). In this way we reduce the decoding complexity.

Even if the results are stated in general terms with the number of voters denoted by n, the analysis works as well for small values of n. So for practical applications concrete values may be derived easily. They may be used to choose the optimal number of trustees for a given security level and the number of voters.

Of course, such a security analysis may be applied to many mix networks as well. The convergence rate depends very much on that how large fraction of all ciphertexts goes through single mixes.

References

1. Aldous, D.: Random Walks of Finite Groups and Rapidly Mixing Markov Chains. In: Azéma, J., Yor, M. (eds.): Séminare de Probabilités XVII 1981/82. Lecture Notes in Mathematics, Vol. 986. Springer-Verlag, Berlin (1983) 243-297
2. Bubley, B., Dyer, M.: Path Coupling: A Technique for Proving Rapid Mixing in Markov Chains. In: Proceedings of the 38th Symposium on Foundations of Computer Science. Miami Beach, FL, 19-22 October 1997. IEEE Computer Society Press, Los Alamitos, CA 223-231
3. Bubley, R., Dyer, M.: Faster Random Generation of Linear Extensions. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms. San Francisco, CA, 25-27 January 1998. SIAM, Philadelphia, PA 355-363
4. Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. In: Communications of the ACM. (1981) 24(2):84-88
5. Chaum, D.: Secret-Ballot Receipts and Transparent Integrity. Better and Less-costly Electronic Voting at Polling Places. http://www.vreceipt.com/article.pdf

6. Czumaj, A., Kutyłowski, M.: Delayed Path Coupling and Generating Random Permutations. In: Random Structures and Algorithms. (2000) 17(3-4): 238-259
7. Czumaj, A., Kanarek, P., Kutyłowski, M., Loryś K.: Switching Networks for Generating Random Permutations. In: Switching Networks: Recent Advances. Kluwer Academic Publishers, (2001)
8. Czumaj, A., Kanarek, P., Kutyłowski, M., Loryś, K.: Delayed Path Coupling and Generating Random Permutations via Distributed Stochastic Processes. 10th Annual Symposium on Discrete Algorithms, ACM-SIAM, New York- Philadelphia, (1999) 271-280
9. Jakobsson, M., Juels, A., Rivest, R.R.: Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. http://www.rsasecurity.com/rsalabs/staff/bios/mjakobsson/rpcmix/rpcmix.pdf
10. Naor, M., Shamir, A.: Visual Cryptography. In: De Santis, A. (ed): Advances in Cryptology – Eurocrypt '94. Lecture Notes in Computer Science, Vol. 950. Springer-Verlag, Berlin (1995) 1-12

Towards Accountable Management of Privacy and Identity Information

Marco Casassa Mont, Siani Pearson, and Pete Bramhall

Hewlett-Packard Laboratories, Trusted Systems Laboratory
BS34 8QZ, Bristol, UK
{marco.casassa-mont,siani.pearson,pete.bramhall}@hp.com

Abstract. Digital identities and profiles are valuable assets: they are more and more relevant to allow people to access services and information on the Internet. They need to be secured and protected. Unfortunately people have little control over the destiny of this information once it has been disclosed to third parties. People rely on enterprises and organizations for its management. In most cases this is a matter of trust. This paper describes an approach to make organizations more accountable, provide strong but not impregnable privacy enforcement mechanisms and allow users to be more involved in the management of the privacy of their confidential information. As part of our ongoing research, we introduce a technical solution based on "sticky" privacy policies and tracing services that leverages Identifier-based Encryption (IBE) along with trusted platform technologies such as TCPA (TCG) and Tagged Operating Systems. Work is in progress to prototype this solution.

1 Introduction

Digital identities and profiles represent and model characteristics and attributes of people: they can include valuable data, such as personal details, social details, financial and business information. They are increasingly relevant to enable Internet transactions and interactions among people and service providers, enterprises and government institutions.

Digital identities and profile information are precious to organisations: they can be used to improve and customise services and provide strategic and marketing information. On the other hand, misuses and unauthorised leakages of this information can violate people's privacy, cause frauds and encourage spamming.

People perceive and address security and privacy issues related to their digital identities in different ways, ranging from completely ignoring them (and indiscriminately disclosing their personal data) to being so concerned to prevent them from using any Internet and web-based application.

Identity and privacy management solutions are going to play a key role in protecting identities and profiles, creating more awareness, enforcing good management practices and accountability, helping to detect criminal activities and supporting forensic analysis. These solutions need to simplify users' experience so that people can have a better experience in dealing with the management of the privacy of their confidential data. When people are not willing to be involved in the active protection and management of their digital assets, trusted entities could do this on their behalf and could provide people with easy-to-use tools to monitor the situation.

This paper presents an approach to make organizations more accountable, provide strong but not impregnable privacy enforcement mechanisms and allow people (or third parties acting on their behalf) to be more involved in the management of the privacy of their personal information. As part of our ongoing research, we introduce a technical solution based on "sticky" privacy policies and tracing services that leverages Identifier-based Encryption (IBE) [1,2,3], TCPA [4] and Tagged OS [5] technologies. Work is in progress to prototype this solution.

2 Addressed Problems

This paper focuses on the problems of enforcing privacy, making organizations more accountable when dealing with confidential data and allowing people to be more involved in the management of the privacy of their data.

In order to describe the involved issues, we refer to an e-commerce scenario. In no way are the aspects we highlight limited to this sector, as they are common to financial, government and enterprise areas.

Fig. 1 shows a scenario where users deal with electronic transactions that span across multiple e-commerce sites:

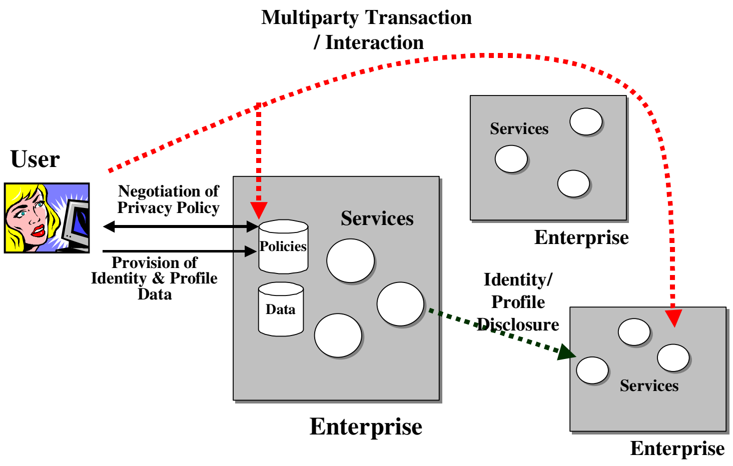

Multiparty Transaction / Interaction
User
Negotiation of Privacy Policy
Provision of Identity & Profile Data
Policies
Data
Services
Enterprise
Services
Enterprise
Identity/ Profile Disclosure
Services
Enterprise

Fig. 1. A Multiparty Transaction

In this scenario a person initially provides their digital identity and profile information to an e-commerce site in order to access their web services, possibly after accepting (or negotiating) a set of privacy and data protection policies. Then the user logs in and interacts with these services. It might happen that other web sites or organizations need to be involved: the e-commerce site might have to disclose personal data to third parties (such as suppliers, information providers, governments and financial institutions, etc.) in order to fulfill the specific transaction. The involved parties might have no prior agreements with the user or they might not belong to the same web of trust. Users might be conscious of this or they might be oblivious of this as the e-commerce site mediates for the required interactions.

Little has been done so far to directly involve users, or entities acting on their behalf, in the management of their privacy, especially when multiparty interactions and transactions take place.

Users lack control over their personal information after the initial disclosures. Other involved parties (such as delegates, e-commerce sites or enterprises) also lack control over the confidential information they manage on behalf of their customers, in particular when they disclose it to other organisations. Once data is disclosed it can be misuses by any of the involved parties. It is hard to make organizations accountable of their behaviours.

3 Related Work

A great deal of work has been done in this area to provide a legislative framework. Organizations that deal with people's personal and confidential data interpret existing laws and deploy ad-hoc solutions to address privacy management and data protection.

However, privacy and data protection laws are hard to enforce, especially when personal information spreads across boundaries. Complexity arises due to the fact that these laws can differ quite substantially depending on geographical aspects. For example in the US privacy laws restrict what the government and organisations in specific sectors (such as health care and financial services) can do with personal data but they introduce few restrictions on trading of personally identifiable information by private enterprises. In the European Union, people can consent to have their personally identifiable information used for commercial purposes but the default is to protect that information and not allow it to be used indiscriminately for marketing purposes.

In general, users have little understanding or knowledge of these laws and their implications. There is a shortage of tools and mechanisms that allow users - or trusted entities acting on their behalf - to explicitly define their own privacy policies and check if they are fulfilled.

Mechanisms such as W3C's Platform for Privacy Preferences (P3P) [6] allow users to define simple privacy policies but only for point-to-point interactions. There is little control over the subsequent fulfillment of these policies.

Liberty Alliance [7] and Microsoft [8] efforts in federated identity management are (for the time being) based on a closed web of trusts. Identity providers must be part of trusted clubs and be compliant with predefined privacy policies.

This approach limits scalability and flexibility of the allowed interactions and transactions.

Relevant work towards a more fine-grained control over the “privacy management” of personal information has been described by [9,10]. In paper [9] the authors define a privacy control language that includes user consent, obligations and distributed administration to describe privacy policies, referred to as “sticky policies”. In paper [10] the authors describe a platform for enterprise privacy practices (E-P3P). When submitting data to an enterprise, a user consents to the applicable privacy policies along with selected opt-in and opt-out choices. Privacy policies are associated to users’ data and drive access control decisions and privacy enforcement at the enterprise site. A privacy language, EPAL [11], derived by this initial work, has been submitted to W3C for its standardisation.

Papers [9,10] and the EPAL proposal do not describe how the “stickiness” of privacy policies (i.e. the strong association of privacy policies to confidential data) is going to be achieved, after the disclosure of confidential data and when this data is exchanged across enterprise boundaries. Users need to trust the enterprise when disclosing their data. Leakages of personal and confidential information might still happen, despite data protection laws and privacy policies, because of lack of security or the dishonesty of some of the involved intermediaries.

In general, it is hard to provide strong mechanisms to enforce privacy, especially in multiparty contexts: once disclosed, confidential data can potentially be misused by any of the involved parties. A step forward in addressing this problem consists in making the involved parties more accountable and mitigate the risks by providing fine-grained mechanisms to enforce privacy aspects at different levels of abstraction, including application, platform and OS levels.

4 Proposed Approach

This section introduces a model to address part of the problems described in section 2 and provides a technical solution based on this model.

Specifically, we describe our work on “sticky” privacy policies, i.e. privacy policies strongly associated to personal data; on mechanisms for their strong but not impregnable enforcement; and on mechanisms for making the involved parties more accountable. We leverage some of the aspects described in [9,10].

We are aware that once confidential data is disclosed (and it is in clear), it can potentially be misused. It is very hard to fully prevent this: our proposed approach does not solve this problem. Nevertheless we believe that the provision of accountability mechanisms along with the usage of trusted platforms can mitigate some of the involved risks.

Work is in progress both in terms of research and development of a prototype.

4.1 Privacy and Accountability Model

The privacy and accountability model proposed in this paper includes the following key aspects:

- **Confidentiality:** obfuscation of confidential (personal) information before its transmission to an uncontrolled host, to protect its content;
- **Strong association of privacy policies to confidential data:** association of "tamper resistant" privacy policies to obfuscated data, defined by users or trusted entities acting on their behalf. The "stickiness" of these policies is guaranteed *at least* till the first disclosure of the confidential data. Any tampering with these policies will prevent the access to the content of obfuscated data;
- **Policy compliance check:** disclosures of confidential data are subordinated to the fulfillment of privacy policies' constraints. A strong but not impregnable policy evaluation and enforcement mechanisms is provided by Tracing and Auditing Authorities (trusted by the users) and trusted platforms;
- **Accountability management:** auditing and tracing of disclosures of confidential data via Tracing and Auditing Authorities;
- **User involvement:** active involvement of users (if desired) during the process of defining privacy policies and disclosing their confidential data.

Fig. 2 maps this model in the e-commerce scenario described in the previous section.

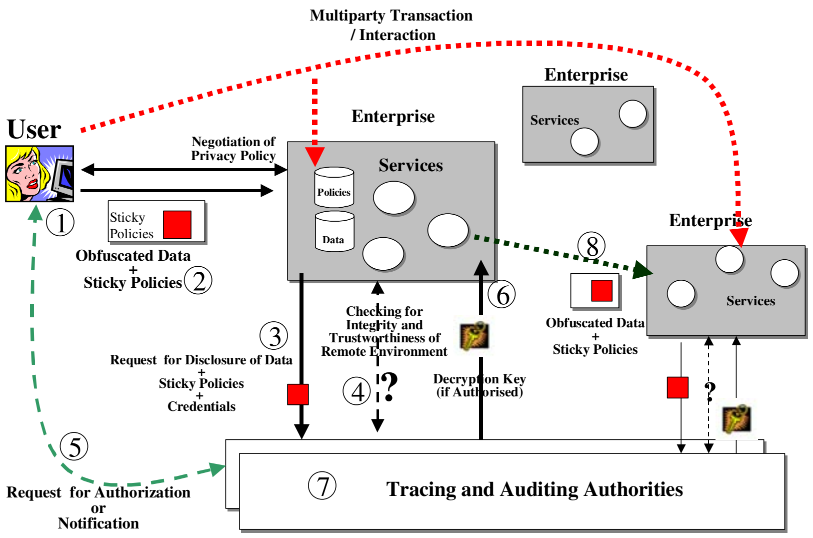

Multiparty Transaction / Interaction
User
Enterprise
Enterprise
Services
Enterprise
Services
Services
Negotiation of Privacy Policy
Policies
Data
Sticky Policies
Obfuscated Data + Sticky Policies
1
2
3
4
5
6
7
8
Request for Disclosure of Data + Sticky Policies + Credentials
Checking for Integrity and Trustworthiness of Remote Environment
Decryption Key (if Authorised)
Obfuscated Data + Sticky Policies
?
?
Request for Authorization or Notification
Tracing and Auditing Authorities

Fig. 2. Proposed Privacy and Accountability Model

In our model users make use of graphical tools (1) to: locally author their privacy policies in a fine-grained way; obfuscate their confidential data by directly using these policies; stick (strongly associate) these policies to the obfuscated data.

Users can have an active role in choosing the Tracing and Auditing Authorities (TAA) they trust. Part of the negotiation process between a user and the data receiver (i.e. an e-commerce site, an enterprise, etc.) consists in choosing TAA trusted by both parties. In absence of an agreement the interaction might not take place.

Some of the above activities can be automated, by using predefined policy templates, scripts and opt-in/opt-out selections: alternatively they can be delegated to third parties, trusted by users.

Digital packages (2) containing obfuscated data along with their privacy policies are created at the user's site and provided to requestors (for example e-commerce sites). These digital packages might contain a superset of the required information, to reduce the number of users' interactions. Selective disclosure [12] of (any part of) their contents will be authorised depending on needs.

A requestor (3) has to interact to the TAA to retrieve the decryption keys to access (de-obfuscate) the confidential data. In doing this the requestor has to provide information and credentials as described by the privacy policies.

The owner of the confidential information can be actively involved in the disclosure process (5) by asking for their authorizations or by sending notifications, according to the agreed privacy policies. In our model nothing prevents the owner of confidential information from being a TAA.

The release of decryption keys (6) by the TAA to access the obfuscated data to a requestor only happens after the requestor demonstrates it can satisfy the associated privacy policies. The TAA can checks (4) for the integrity and trustworthiness of the requestor's credentials and verify some integrity aspects of their IT environment (via trusted platforms), i.e. whether they are, or are not, in accordance with privacy policies. More details on the mechanisms are provided in the next section.

Multiple TAAs can be used in the above process in order to minimise the risks due to the reliance on only a third party.

The TAA (7) logs and audits the disclosures of confidential data every time it is directly involved in the issuance of IBE decryption keys. This is certainly going to happen the very first time the (obfuscated) digital package is transmitted by the user to the receiver (as only the TAA can provide the decryption key).

Unfortunately, once confidential information is disclosed to a requestor, it can be potentially misused without the TAA having a chance to intervene.

Despite this, we believe that the log and audit of initial disclosures increase the accountability of the requestors by creating evidence about their knowledge of users' confidential data. It is in the interest of an honest requestor to send the obfuscated data package to a third party (8) (for example during a multi-party interaction) instead of clear-text data, by using the same process described above. However, unwanted disclosures can happen because of security problems, mistakes or hacking activities. In case of "dishonest" receivers the evidence collected by the TAA can possibly be used for forensic analysis, to pin down their responsibilities.

The usage of trusted platforms at the receiver's site can help to reduce the involved risks by allowing the TAA to check for aspects related to the integrity of the receiver's environments (prior to the disclosure of confidential data) and enforcing part of the privacy policies directly at the platform and OS levels. More details on the mechanisms are provided in the next section.

It is important to notice that our model is orthogonal to authentication mechanisms: it does not provide an authentication mechanism but it can be jointly used with any of the existing mechanisms. If authentication is required by the remote party, once users are authenticated, they can provide their confidential information, as described in our model. The main goal of our approach is to protect the privacy of confidential information as specified by privacy policies, and create audit trails of its disclosures.

4.2 Technical Aspects

This section describes a technical implementation of the above model. It leverages three key technologies:

- Identifier-based Encryption (IBE) [1,2,3]: it is an emerging cryptographic schema where any kind of string - including a name, a role, terms and conditions, etc. - can be used as encryption key. The generation of the corresponding IBE decryption key can be postponed in time. A Trust Authority (TA) generates this decryption key on the fly, under specific circumstances. More than one trust authority can be used, if required, to address trust management issues.
- Trusted Computing Platform Alliance (TCPA, now TCG) technology [4,13]: it provides hardware mechanisms and tools to check the integrity of computer platforms and their installed software.
- Tagged Operating System technology [5]: it provides trusted operating system (OS) mechanisms and tools to associate low level privacy labels to data and directly enforce and manage them at the OS level. The "stickiness" of a label to the content, not to the content holder, such as a file, is an important feature as it ensures that even when the data is copied around, the label follows it as well. In the operating system each privacy label is directly matched to a set of rules (low level policies) that express requirements for how data with that label can be used. These rules dictate how the labelled data should be handled by applications. The advantage of having this type of privacy policy enforcement mechanism lies in that controls apply uniformly across different applications and cannot easily be subverted by them.

In our approach privacy policies are represented as "IBE encryption keys". The "Tracing and Auditing Authority" is basically a function provided by a "Trust Authority".

IBE encryption keys can define any kind of privacy constraints or terms and conditions, at different levels of abstractions (including the application and platform levels). At the very base an IBE encryption key is a string: it is self-explanatory and it is directly used to encrypt confidential data.

An IBE encryption key does stick with the encrypted data. Any alteration or tampering of this string will make impossible for the Trust Authority to generate the correct IBE decryption key. In this perspective, privacy policies are effectively "sticky" privacy policies, strongly associated to confidential data.

No secret needs to be generated and exchanged between users and the receivers of their confidential information. The Trust Authority (TA) will generate the IBE decryption key on the fly, only when required and only if the privacy policies' constraints are met.

The remaining part of this section describes architectural aspects and provides more details on specific aspects.

Fig. 3 shows the architecture of a system implementing our model.

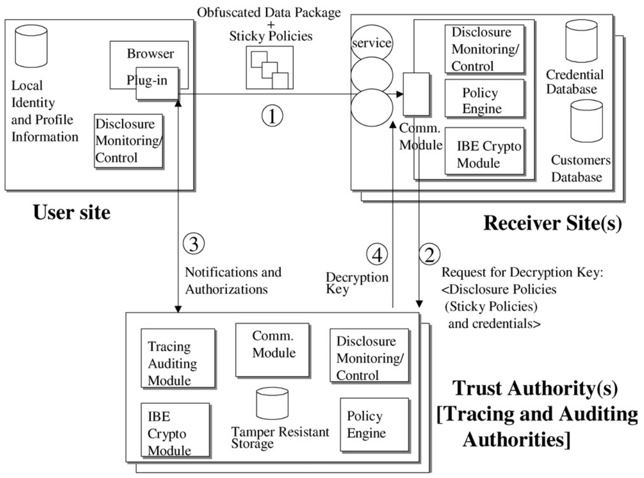

Obfuscated Data Package
+
Sticky Policies
Local Identity and Profile Information
Browser
Plug-in
Disclosure Monitoring/ Control
User site
①
service
Comm. Module
Disclosure Monitoring/ Control
Policy Engine
IBE Crypto Module
Credential Database
Customers Database
Receiver Site(s)
③
Notifications and Authorizations
④
Decryption Key
②
Request for Decryption Key: <Disclosure Policies (Sticky Policies) and credentials>
Tracing Auditing Module
Comm. Module
Disclosure Monitoring/ Control
IBE Crypto Module
Tamper Resistant Storage
Policy Engine
Trust Authority(s) [Tracing and Auditing Authorities]

Fig. 3. High Level Architecture

Messaging protocols (1)-(4) are carried out in order, and involve transfer of the information indicated in the directions shown by the arrows.

Identity or profile information is encrypted with privacy policies (1), before its transmission to third parties, by web browser plug-ins or trusted applications. These policies are used as IBE encryption keys and might include:

- References to logical names of identity and profile attribute(s);
- Disclosure constraints;
- Actions (i.e. notification of the owner in case of multiparty disclosure);
- Expiration dates, etc.

To obtain a valid IBE decryption key (2), the receiver needs to interact with the TA(s) indicated by the privacy policies. In doing this, the receiver has to provide information and credentials (including authentication credentials, business related information, company/individual policy related to data disclosure, usage and storage, software state, platform configuration etc.) as described by these policies.

A TA will issue a decryption key (4) if it acknowledges the compliance with the privacy policies. Before doing this it might interact with the information owner (3) to ask for his/her authorization or to send a notification. The TA traces and stores the information exchanged during these interactions in audit trails, as evidence for future contentions or forensic analysis.

More details follow on how privacy policies (IBE encryption keys) are used and authored; how a strong but not impregnable enforcement of privacy policies can be provided by the TA and trusted platforms; how multiple TAs can be involved; how information owners can themselves act as a TA; how non-compliance can be tracked.

Properties of "Sticky" Privacy Policies. Users' confidential information is exchanged by means of data packages containing encrypted data and their associated privacy policies. A simple example of such a data package is shown in fig. 4. Privacy policies can specify two categories of constraints:

- **"soft" constraints:** they are enforced via Trust Authorities (TAs). They can potentially be violated, once users' information has been disclosed. The involved risks are mitigated (and accountability underpinned) by TAs' tracing and auditing mechanisms.
- **"strong" constraints:** they are strongly enforced by trusted platform mechanisms such as the TCPA integrity checking mechanisms and tagged OSs. They are harder to violate, if the integrity and policy conformance of the hosting environment have been verified.

The importance of user's data dictates which "mixture" of the two categories of constraints needs to be used.

In fig. 4 example, the data package contains only a confidential attribute (for example a credit card number). The associated privacy policy contains:

- *An encrypted "identifier" of the owner.* This can be any type of information, including the owner's e-mail address, URL, etc. Note that a "reference name" (a pseudonym, for example) has been used as an IBE encryption key to encrypt the owner identifier. Only the competent Trust Authority will be able to retrieve the owner's identifier and use it (for example, to notify the owner of a disclosure or ask for their authorization).
- *The name of the attached confidential attribute.*
- *An expiration date:* date after which the Trust Authority will not issue anymore the decryption key.
- *Multiple type of constraints and actions:* It can be any kind of constraint or obligation to be satisfied by the receiver. In the above example the requestor has to use a trusted platform, strongly authenticate to the Trust Authority (for example by using PKI-based X.509 identity certificates [14]) and declare the intended usage of the attribute. An additional constraint involves the notification of the owner in case of disclosure.

```
<data package>
  <data component>  // Identity and profile - attribute 1
   <sticky policy> // disclosure policy – IBE encryption key
      <attribute>
         name of the identity or profile attribute
      </attribute>
      <Trust Authority>
        address and location of the Trust Authority
      </Trust Authority>
      <owner>
       //reference name – IBE  encryption key
        <reference name> pseudonym1 </reference name>
        //encrypted call back address by using user's reference name
        <owner's details>
           encrypted call back address
       <owner's details>
      </owner>
      <validity> expiration date  </validity>
      <constraint>
          X.509_authentication_required
      </constraint>
      <constraint>
         allow_sharing_of_data
      </constraint>
      <constraint>   // Simple constraint on remote platform
         required_remote_TCPA_trusted_platform
      </constraint>
       <action>
          notify_owner
        </action>
    </sticky policy>
    <encrypted data>
        encrypted attribute value,  using the above policy as IBE encryption key
     </encrypted data>
   </data component>
 </data package>
```

Fig. 4. Example of "Sticky" Privacy Policies

Privacy policies can be used to allow a selective disclosure of aggregations and combinations of confidential information; they can be associated in a fine-grained way to any kind of attributes. To each sub privacy policy can be associated a correspondent IBE decryption key.

They can be composed and extended in a very flexible way. We use an XML-based representation as a matter of convenience. Emerging languages and data formats, such as signed XML and SAML, can be used. Any kind of constraint, obligation and permission can be added (including logical statements), as long as they are programmatically interpretable and the Trust Authority (TA) and the receivers' systems understand its semantics.

The receiver of the encrypted information (for example an identity provider or an e-commerce site) can programmatically interpret the associated disclosure policies by means of a policy engine.

We are currently working to refine key aspects of our privacy policies, including hierarchies of policies, composition of policies and their mapping at different levels of abstraction (service, application, system and OS).

Policy Authoring. It must be simple for users, or trusted entities acting on their behalf, to author privacy policies. We envisage the usage of policy templates containing hierarchies of privacy policies along with instructions on how to refine their components. Fig. 5 shows these aspects.

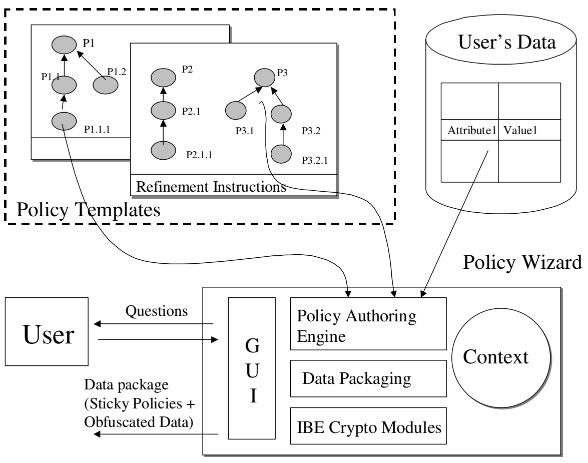


Fig. 5. Privacy Policy Templates and Authoring

Privacy policies defined at different hierarchical points (within policy templates) might have different "level of refinements" of their components, including their constraints, actions, etc. For example privacy policy P.2.1 might contain more specific and restrictive constraints than P.2.

Policy templates define categories (classes) of privacy policies. They are interpreted and managed by a policy-authoring tool. This tool interacts with people via intuitive GUI by asking simple questions: it also makes use of contextual information to make decisions.

Privacy policies can be refined and composed, within and across templates, depending on needs. Preliminary work in this area is described by [15]. Privacy policies (IBE encryption keys) are associated to user's personal data, to be obfuscated. Policy authoring, data encryption and data packaging functionalities can be supplied by integrated applications, such as web browser add-ins.

Policy Enforcement by Trusted Platforms. TCPA integrity checking mechanisms [4] can be used to check that a platform is a trusted computing platform, that the software state of this platform is conformant with constraints defined by privacy policies and that the platform implements defined privacy management mechanisms. This could be provided by using similar mechanisms within Microsoft's NGSCB (Palladium) [16].

In case trusted computing platforms are available, cross integrity checks can be performed on the platforms by the involved parties. For example, the TA can check the integrity of the receiver's computing platform and allow the TA('s) platform to be checked out by the user and/or the recipient of the data.

To be effective, this process requires that at least all the systems involved in the management of confidential data (at the receiver site) are trusted platforms and that they satisfy the privacy polices. This can be a strong requirement for the time being, as trusted platforms are not ubiquitous. In addition not all these systems are going to be externally exposed, so their integrity cannot be directly verified (for example by the TA).

This is currently an open issue investigated by HP laboratories. Tagged operating systems can be jointly used with TCPA platforms to address part of these issues.

Tagged operating systems (OSs), such as [5], increase security and trust by tagging confidential data (for example, as specified by privacy policies) and executing associated rules at the OS level. The rules are enforced through the policy modules designed as an add-in mechanism within the operating system.

Rules can define constraints on the usage of this data (for example by disallowing it to be copied) and restricting its transmission to other platforms (for example by only allowing the transmission to predefined IP addresses).

This enforcement is invisible to both applications and users. As applications have no access to the tags and associated rules, the information owners are assured that their data is protected even if computer systems are infected with malicious code, or penetrated by intruders. The technology required to implement the above solution is currently available at HP Laboratories [5].

If tagged OSs are jointly used with TCPA platforms, their integrity can be checked upfront. Tags and rules (i.e. aspects of privacy policies) could be set by external entities, for example by the TA, to ensure that confidential information is not propagate on platforms that are not compliant with privacy policies.

Policy Enforcement by Trust Authorities. The TA interprets privacy policies via a policy engine and makes sure that the associated constraints are satisfied before generating and issuing the IBE decryption key. This mechanism provides a "soft" enforcement of the privacy policies as it relies on (needs to trust) the receiver to take care of confidential information, once it is in clear.

If the data is disclosed by the receiver to a third party using the mechanisms described in this paper (which of course, it might not be – it could be given to a third party by many different means), the TA could check that this disclosure has been carried out according to the specification of the (original) disclosure policy, and both refuse to release the decryption key to any third party and report the receiver's behaviour in some appropriate way.

As described in the previous section, the TA can leverage TCPA and tagged OSs to ensure that part of the policy enforcement can be done upfront, before any confidential data is disclosed, as specified by privacy policies.

It is important to notice that only a widespread usage of trusted platforms can enable the checks and the enforcement of privacy policies as described in this section. In general, it is going to be very hard to completely prevent a receiver from misusing confidential data. There will always be a requirement for some degrees of trust somewhere in the system. Hence the importance of having trust authorities that can audit and trace at least the initial disclosures of confidential data and can create evidence to be used in courts of laws and for forensic analysis.

Accountability Management. Receivers of personal and confidential information need to be compliant with privacy policies, as defined by the information owners. They need to make the required steps in order to demonstrate their compliance to the relevant TAs.

If the receiver discloses data in a way that is not contemplated by the privacy policies, there is an audit trail (at the TA(s) site(s)) showing that they interpreted these policies and provided information and credentials as required by them.

In case of identity or profile thefts, the audit information can be used to pin down a list of potential "offenders" and carry on forensic analysis. The tracing and auditing of disclosures makes the information receivers more accountable.

Multiple TAs. To enable an electronic transaction involving user's confidential data, the receiver might send obfuscated data or any portion of it to another third party, for example a service provider. It might decide to encrypt portions of this data by using additional policies. This third party has to interact again with a TA as described above.

The receiver may have to use multiple TAs in order to access the data. For example, one TA might be competent with respect to security platforms and other might be competent in privacy, so it would make sense for both to carry out checks before allowing an entity to access data. In this case, the user might encrypt the data using a disclosure policy that specifies that it is necessary to use two (or more) sub IBE decryption keys in order to decrypt the data, and each of the TAs would provide one of these keys. Multiple keys might be needed to decrypt the same piece of data, or different data fields might be encrypted using different keys.

There is another case where multiple TAs might be needed: when data is forwarded from the receiver on to another entity. Here, there are two different types of case: either the receiver uses the same TA, in which case it could just send on the encrypted message it received from the original sender (or, if desired, it could use a different disclosure policy and therefore obtain a different encryption); or it uses a different TA, in which case the third party would have to apply to that TA to get the decryption key, etc., as described above.

The IBE encryption schema supports the management of encryption keys where the correspondent decryption keys are the result of the composition of sub-decryption keys obtained by multiple TAs.

Owners of identity and profile information can run their own TA services to have first hand understanding of what happens to their information and make ultimate decisions. Alternatively, users can periodically interact with the TA to monitor the disclosure state of their confidential information.

5 Discussion

The idea of using third parties to mediate the access to confidential information is not new. There are well-known related issues, including why a person or an organisation should trust a third party. Multiple approaches have been analysed and described in the literature, including branding, certifications and seals, presence on the market and historical information. This (fundamental) aspect is not covered in this paper as it is out of its scope. From our perspective enterprises and organisations that are trusted in the real world can be trust authorities.

Multiple Trust Authorities can be involved in order to minimise the risk of having to rely only on an entity. In our specific case, multiple Tracing and Auditing Authorities could be used in the process of issuing IBE decryption keys: information owners can run their trust authorities. Multiple IBE sub-decryption keys can be recombined into an IBE encryption key.

The usage of cryptography and, specifically, encryption mechanisms to preserve the confidentiality of personal data is also common practice.

We believe that the value we bring in this area is in the mechanisms we provide to associate "sticky" privacy policies to confidential data via the IBE technology; the active interaction model adopted to force requestors to be traced (audited), at least at the very first disclosure; the TCPA technology used to check the integrity of remote IT environments along with the enforcement of (part of) privacy policies via tagged OSs.

It is important to notice that once confidential information has been disclosed to a requestor and it is in clear text (at the requestor site), it can be potentially misused. The usage of trusted computing platforms and tagged OSs can reduce the involved risks by doing preemptive trust and security checks along with enforcement of policies at the OS level. In addition, in case of leakages and misbehaviors, the tracing and auditing information collected by TAs can be used for forensic analysis to pin down responsibilities.

In terms of obfuscation of users' data, traditional RSA cryptography (based on public/private keys), PKCS7 enveloping techniques and PKI [14] can be used to provide functionalities similar to IBE. For example the Trust Authority might have a X.509 identity certificate: this certificate can be used to encrypt a symmetric key, generated by the user. This symmetric key can be used to encrypt users' confidential information along with a hash value derived from the associated privacy policies.

IBE technology simplifies the management of obfuscated data at the client site and pushes the complexity to the enterprise (remote) site. This removes a major practical burden. In case of multiple TAs, we believe that our approach based on IBE is simpler than analogous approach based on RSA (public key) technology, because of the easiness by which multiple IBE sub-decryption keys can be combined in the IBE decryption key.

The Trust Authority (Tracing and Auditing Authority) is the right place to manage accountability, by tracing and auditing disclosures. Requestor – at least the ones having an obfuscated data package – do need to interact with the Trust Authority to obtain an IBE decryption key. They need to provide their contextual credentials, as mandated by the privacy policies: this information is logged and can be used as evidence to make them accountable. The auditing and tracing effort is effective also to audit users' behaviors, as the Trust Authority is a trusted bridge between users and receivers.

Current literature, including [9,10], recommends that enterprises define their own privacy and security policies, in a way that it is compliant with laws and legislation. To programmatically deal with these policies they need policy engines integrated with traditional authentication and access control components. The model and technical solution described in this paper are complementary to the above aspects.

6 Current and Future Work

IBE and TCPA technologies are currently available at HP Laboratories and on the market. In particular the Trusted Systems Laboratory (HP Laboratories, Bristol) has implemented an optimised version of the IBE code that provides IBE cryptography functions with a performance comparable to RSA-based code. TCPA chips and PCs are available on the market.

We have simple implementations of most of the components required by our technical solution, including a *Trusted Authority (TA) service*, IBE algorithms to support multiple TAs, an *add-in* to author privacy policies and a policy driven (and context aware) authorization engine [17]. Work is in progress to build a non-repudiable logging and auditing system [18].

We also have a working prototype of the tagged OS [5] to enforce (parts of) privacy policies directly at the OS level.

Our aim is to refine our model and learn by building and deploying the system in a real-life environment.

7 Conclusions

It is important to protect and preserve people's privacy on the Internet, against unwanted and unauthorised disclosure of their confidential data. Despite laws, legislations and technical attempts to solve this problem, at the moment there are no solutions to address the whole set of involved issues.

In this paper we specifically address three important issues: letting users (or trusted entities acting on their behalf) be more involved in the management of the privacy of their personal data; making organisations more accountable of their behaviours, whilst dealing with users' confidential data; provide strong but not impregnable privacy enforcement mechanisms.

Our research is in progress. We introduced and described a model based on "sticky" privacy policies, i.e. "tamper resistant" privacy policies strictly associated to obfuscated confidential data, along with trusted tracing services (trust authorities) that check and audit data disclosures. We described a technical solution where IBE technology coupled with TCPA and tagged OSs are used to address the above problems and reduce the involved risks.

These core technologies are available at the HP Laboratories, Bristol, along with simple implementations of most of the required solution components. Work is in progress to prototype our solution.

References

1. D. Boneh, M. Franklin: Identity-based Encryption from the Weil Pairing. In: Crypto 2001. (2001)
2. L. Chen, K. Harrison, A. Moss, D. Soldera, N. P. Smart: Certification of Public Keys within an Identity Based System. In: Proc. 8th ACM Conference on Computer and Communications Security. LNCS, Springer-Verlag (2002) 332–333
3. C. Cocks: An Identity Based Encryption Scheme based on Quadratic Residues. Technical report, Communications Electronics Security Group (CESG), UK (2001)
4. TCPA: Trusted Computing Platform Alliance Main Specification v1.1. `http://www.trustedcomputing.org` (2001)
5. Y. Beres, C. I. Dalton: Dynamic Label Binding at Runtime. In: Proceeding of New Security Paradigms Workshop, August 2003. (2003)
6. W3C: The Platform for Privacy Preferences 1.0 specification (P3P 1.0). `http://www.w3.org/tr/p3p` (2002)
7. Liberty Alliance: Liberty Alliance Project. `http://www.projectliberty.org/` (2002)
8. Microsoft: Microsoft .NET Passport. `http://www.microsoft.com/netservices/passport/` (2002)
9. G. Karjoth, M. Hunter: Privacy Policy Model for Enterprises, IBM Research, Zurich. In: 15th IEEE Computer Foundations Workshop. (2002)
10. G. Karjoth, M. Schunter, M. Waidner: Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data. In: 2nd Workshop on Privacy Enhancing Technologies. LNCS, Springer-Verlag (2002)
11. P. Ashley, S. Hada, G. Karjoth, C. Powers, M. Schunter: Enterprise Privacy Authorization Language (EPAL). Technical report, IBM (2003)
12. S. Pearson: A Trusted Mechanism for User Self-Profiling in E-Commerce. Selected Papers from Special Track on Privacy and Protection with Multi-Agent Systems, LNAI journal, Springer (2003)
13. S. Pearson (ed.): Trusted Computing Platforms. Prentice Hall (2002)
14. R. Housley, W. Ford, W. Polk, D. Solo: RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL profile. Technical report, IETF (1999)
15. M. Casassa Mont, A. Baldwin, C. Goh: POWER Prototype: Towards Integrated Policy-based Management. In: NOMS 2000. (2000)
16. Microsoft: Microsoft NGSCB: Next Generation Secure Computing Base, Technical FAQ (2003)
17. M. Casassa Mont, R. Brown: PASTELS project: Trust Management, Monitoring and Policy-driven Authorization Framework for E-Services in an Internet based B2B environment. Technical report, HP Labs, HPL-2001-28 (2001)
18. A. Baldwin, S. Shiu: Enabling Shared Audit Data. In: Proceedings of the 6th Information Security Conference. LNCS, Springer-Verlag (2003)

A Toolkit for Managing Enterprise Privacy Policies

Michael Backes, Birgit Pfitzmann, and Matthias Schunter

IBM Zurich Research Laboratory, Rüschlikon, Switzerland
{mbc,bpf,mts}@zurich.ibm.com

Abstract. Enterprise privacy enforcement allows enterprises to internally enforce a privacy policy that the enterprise has decided to comply to. An enterprise privacy policy often reflects different legal regulations, promises made to customers, as well as more restrictive internal practices of the enterprise. Further, it may allow customer preferences. Hence it may be authored, maintained, and audited in a distributed fashion.
Our goal is to provide the tools for such management of enterprise privacy policies. The syntax and semantics is a superset of the Enterprise Privacy Authorization Language (EPAL) recently proposed by IBM. The basic definition is refinement, i.e., the question whether fulfilling one policy automatically fulfills another one. This underlies auditing of a policy against an old or new regulation or promise and transferring data into a realm with a different policy. It is also the semantic basis for composition operators. We further define such composition operators for different purposes. Our main focus it to combine usability for enterprises, e.g., by treating multiple terminologies, incomplete data, and different types of errors and defaults, with the formal rigor needed to make privacy compliance meaningful and predictable.

1 Introduction

An increasing number of enterprises make privacy promises to customers or, at least in the US and Canada, fall under new privacy regulations. To ensure adherence to these promises and regulations, enterprise privacy technologies are emerging [8]. An important tool for enterprise privacy enforcement is formalized enterprise privacy policies [10, 17, 16]. Compared with the well-known language P3P [19] intended for privacy promises to customers, languages for the internal privacy practices of enterprises and for technical privacy enforcement must offer more possibilities for fine-grained distinction of data users, purposes, etc., as well as a clearer semantics.

Although the primary purpose of enterprise privacy policies is enterprise-internal use, many factors speak for standardization of such policies: First, it would allow certain technical parts of regulations to be encoded into such a standardized language once and for all. Secondly, a large enterprise with heterogeneous repositories of personal data could then hope that enforcement tools for all these repositories become available that allow the enterprise to consistently enforce at least the internal privacy practices chosen by the CPO (chief privacy officer). Thirdly, with increasingly dynamic e-business, data will be exchanged between enterprises, and enterprise boundaries change due to mergers, acquisitions, or virtual enterprises. Then the sticky-policy paradigm stressed in papers like [17] must be enforced. It states that the policy under which data have been collected has to govern the use of these data at all times. This also requires compatible enterprise privacy enforcement mechanisms. For these reasons, IBM has recently proposed an Enterprise Privacy Authorization Language (EPAL) [1] as an XML specification for public comments and possible subsequent input to standardization.

An enterprise privacy policy often reflects different legal regulations, promises made to customers, as well as more restrictive internal practices of the enterprise. Further, it may allow customer preferences. Hence it may be authored, maintained, replaced, and audited in a distributed fashion. In other words, one will need a life-cycle management system for the collection of enterprise privacy policies. While such thoughts occur as motivation in most prior work on enterprise privacy policies, no actual definitions and algorithms needed for these management tools have been proposed.

The overall goal of this article is therefore to provide a comprehensive range of tools for designing and managing privacy policies in an enterprise. We do this concretely for the IBM EPAL proposal. However, for a scientific paper we cannot use the lengthy XML syntax, but have to use a corresponding abstract syntax presented in [2] (which, like EPAL, is based on [17]). Our paper reflects recent updates made between the earlier abstract [2] and the published specification and XML Schema [1], so that it is currently as close as possible to EPAL. Further, we do not abstract from conditions in contrast to [2] so that we can define a semantics for incomplete context data, which is useful both in general practice and specifically for refinements and composition of policies from different realms. In spite of the current closeness to EPAL, we continue to call the abstract language E-P3P as in [2] to avoid confusion with possible changes to EPAL.

The first tool we define is policy refinement. Intuitively, one policy refines another if using the first policy automatically also fulfills the second policy. It is thus the fundamental notion for many situations in policy management. For instance, it enables verification that an enterprise policy fulfills regulations or adheres to standards set by consumer organizations or a self-regulatory body, assuming only that these coarser requirements are once and for all also formalized as a privacy policy. Similarly, it enables verification that a detailed policy for a part of the enterprise (defined by responsibility or by technology) refines the overall privacy policy set by the company's CPO. The verification can be done in the enterprise or by external auditors, such as [21].

When a policy is first designed, refinement may be achieved in a constructive way, e.g., by starting with the coarse policy and only adding details by certain provably refining syntactic means. However, if a regulation changes or the enterprise extends its operation to new sectors or countries, the enterprise has to verify that its existing policy still complies with the new or additional regulations. Hence a definition of refinement between two arbitrary policies is needed. Sticky policies are another application of general refinement: Here data are transferred from the realm of one policy into another (where the transfer must of course be permitted by the first policy), and the second realm must enforce the first policy. However, the enforcement mechanisms (both organizational and technical) in the second realm will often not be able to deal with arbitrary policies for each obtained set of data. In this case, one realm must perform a refinement test before the data are transferred, i.e., one has to verify that the policy of the second realm refines the policy of the first, at least for the restriction of the first policy to the data types being transferred.

Composition is the notion of constructively combining two or more policies; typically the goal is that the resulting policy refines them all. For instance, an enterprise might first take all applicable regulations and combine them into a minimum policy. A general promise made to customers, e.g., an existing P3P translated into the more general language, may be a further input. In enterprise parts that support detailed preferences of individuals, such preferences may be yet another policy to be composed with the others, yielding one final policy per individual. (In contrast, simple preferences may be represented as a set of Boolean opt-in or opt-out choices, and treated as context data by conditions within a single policy.) Typical applications where detailed preferences are needed are wallet-style collections of user data for the purpose of transfer to other enterprises, and collaborative tools such as team-rooms.

Composition is not a simple logical AND for powerful enterprise privacy policies as in EPAL, e.g., because of the treatment of obligations, different policy scopes, and default values. Moreover, refinement and composition turn up two basic questions about the meaning of a privacy policy, which are not answered by the abstract semantics of an individual policy. The first question is the meaning of a positive ruling in privacy policies. Intuitively, negative rulings are understood to be definite; e.g., if a policy states that certain data are not used for email marketing, then no such email marketing should happen. The intuition is different for most positive rulings: If a policy allows third-party email marketing, it is typically not seen as a promise to actually do marketing, neither to the owners of the email addresses nor to the third parties. However, if one decides to represent access rights for data subjects to their data, such as the right to see all their data or to correct mistakes, with the normal policy mechanisms, then these positive rulings must be mandatory. The second question is related: If a privacy policy, like EPAL, is formulated with precedences to enable easy formulations of positive and negative exceptions, then within a policy, neither negative nor positive rules are "final", i.e., can be considered isolated from the policy. In contrast, in compositions, one may want to retain an entire original policy as final. We solve both these problems by allowing mandatory sub-policies. This allows us to distinguish final decisions from decisions that may be overturned by other rules, and thus to represent all the cases just discussed. We extend the notion of composition and refinement to these two-part policies.

Further Related Literature. The core contribution of new privacy-policy languages [10, 17, 16], compared with other access-control languages, is the notion of purpose and purpose-bound collection of data, which is essential to privacy legislation. Other necessary features that prevent enterprises from simply using their existing access-control systems are obligations and conditions on context information. Individually, these features were also considered in recent literature on access control, e.g., purpose hierarchies in [5], obligations in [4, 14, 20], and conditions on context information in [22]. However, we need them all in one language, and even for the individual features the detailed semantics needed in practice, such as with multiple terminologies, typically does not exist yet, and thus nor does a comparable toolkit. Policy composition has been treated before, in particular for access control [6, 7, 9, 13, 15, 22], systems management [18], or IPSEC [11]; however none of these papers does it for the general policies we need and several do not have a clear underlying semantics. The publications closest to our treatment of incomplete data are those on information-disclosure-minimal negotiation of access-control policies, e.g., [3, 12].

2 Syntax and Semantics of E-P3P Enterprise Privacy Policies

Privacy policies define the purposes for which collected data can be used, model the consent a data subject can give, and may impose obligations onto the enterprise. They can formalize privacy statements like "we use data of a minor for marketing purposes only if the parent has given consent" or "medical data can only be read by the patient's primary care physician". In this section, we present the abstract syntax and semantics E-P3P of IBM's EPAL privacy policy language [1]. Compared with [2], we abstract less from conditions and obligations, so that we can present a more detailed semantics.

2.1 Hierarchies, Obligations, and Conditions

We start by defining the models of hierarchies, obligations, and conditions used in E-P3P, and operations on them as needed in later refinements and compositions.

For conveniently specifying rules, the data, users, etc. are categorized in E-P3P as in many access-control languages. This also applies to the purposes. In order to allow structured rules with exceptions, categories are ordered in hierarchies; mathematically they are forests, i.e., multiple trees. For instance a user "company" may group several "departments", each containing several "employees". The enterprise can then write rules for the whole "company" with exceptions for some "departments".

Definition 1 (Hierarchy). *A* hierarchy *is pair $(H, >_H)$ of a finite set H and a transitive, non-reflexive relation $>_H \subseteq H \times H$, where every $h \in H$ has at most one immediate predecessor (parent). As usual we write $\geq_H$ for the reflexive closure.*

For two hierarchies $(H, >_H)$ and $(G, >_G)$, we define

$$(H, >_H) \subseteq (G, >_G) :\Leftrightarrow (H \subseteq G) \wedge (>_H \subseteq >_G);$$
$$(H, >_H) \cup (G, >_G) := (H \cup G, (>_H \cup >_G)^*);$$

where * denotes the transitive closure. Note that a hierarchy union is not always a hierarchy again. $\diamond$

E-P3P policies can impose obligations, i.e., duties for the enterprise. Examples are to send a notification to the data subject after each emergency access to medical data, or to delete data after a given time. Obligations are not structured in hierarchies, but by an implication relation. For instance, an obligation to delete data within 30 days implies that the data are deleted within 60 days. The overall obligations for a rule in E-P3P are written as sets of individual obligations, which must have an interpretation in the application domain. As multiple obligations may imply more than each one individually, we define the implication (which must also be realized in the application domain) on these sets. We also define how this relation interacts with vocabulary extensions.

Definition 2 (Obligation Model). *An* obligation model *is a pair* $(O, \rightarrow_O)$ *of a set O and a relation* $\rightarrow_O \subseteq \mathfrak{P}(O) \times \mathfrak{P}(O)$*, spoken* implies*, on the powerset of O, where* $\bar{o}_1 \rightarrow_O \bar{o}_2$ *for all* $\bar{o}_2 \subseteq \bar{o}_1$*, i.e., fulfilling a set of obligations implies fulfilling all subsets.*

For $O' \supset \mathfrak{P}(O)$*, we extend the implication to* $O' \times \mathfrak{P}(O)$ *by* $((\bar{o}_1 \rightarrow_O \bar{o}_2) :\Leftrightarrow (\bar{o}_1 \cap \mathfrak{P}(O) \rightarrow_O \bar{o}_2))$. ◇

The decision formalized by a privacy policy can depend on context data. Examples are a person's age or opt-in consent. In EPAL this is represented by conditions over data in so-called containers [1]. The XML representation of the formulas is taken from [22], which corresponds to a predicate logic without quantifiers. In the abstract syntax in [2], conditions are abstracted into propositional logic, but this is too coarse for our purposes. Hence we extend E-P3P to be closer to EPAL by formalizing the containers as a set of variables with domains, and the conditions as formulas over these variables.

Definition 3 (Condition Vocabulary). *A* condition vocabulary *is a pair* $Var = (V, Scope)$ *of a finite set V and a function assigning every* $x \in V$*, called a* variable*, a set* $Scope(x)$*, called its* scope.

Two condition vocabularies $Var_1 = (V_1, Scope_1)$*,* $Var_2 = (V_2, Scope_2)$ *are* compatible *if* $Scope_1(x) = Scope_2(x)$ *for all* $x \in V_1 \cap V_2$*. For that case, we define their* union *by* $Var_1 \cup Var_2 := (V_1 \cup V_2, Scope_1 \cup Scope_2)$. ◇

In the future, one might extend this to a full signature in the sense of logic, i.e., including predicate and function symbols. In EPAL, this is hidden in user-defined functions that may occur in the XACML conditions. For the moment, we assume a given universe of predicates and functions with fixed domains and semantics.

Definition 4 (Condition Language). *Let a condition vocabulary* $Var = (V, Scope)$ *be given.*

- *The* condition language $C(Var)$ *is the set of correctly typed formulas over V using the assumed universe of predicates and functions, and in the given syntax of predicate logic without quantifiers.*
- *The free variables of a formula* $c \in C(Var)$ *are denoted by* $free(c)$*. Here these are all variables of c.*
- *A (partial)* assignment *of the variables is a (partial) function* $\chi \colon V \rightarrow \bigcup_{x \in V} Scope(x)$ *with* $\chi(x) \in Scope(x)$ *for all* $x \in V$*. The set of all assignments for the set Var is written* $\mathfrak{Ass}(Var)$*; that of all partial assignments* $\mathfrak{Ass}_{\subseteq}(Var)$*.*
- *For* $\chi \in \mathfrak{Ass}(Var)$*, let* $eval_\chi \colon C(Var) \rightarrow \{true, false\}$ *denote the evaluation function for conditions given this variable assignment. This is defined by the underlying logic and the assumption that all predicate and function symbols come with a fixed semantics.* ◇

An important aspect of our semantics is the ability to deal meaningfully with underspecified requests. This means that a condition might not be evaluatable since only a subset of the variables used in the conditions has been assigned. This is important not only in federated scenarios as introduced in [9], but also for overall in-enterprise policies. For instance, some rules of a policy may need the age of a person or its employee role, while for many people no age or employee role is known in the enterprise. This typically does no harm because other rules apply to these persons. For such situations, we will need to know whether a condition can still become $true$ or $false$, respectively, when a partial assignment is extended. Hence we define extensions.

Definition 5 (Extension of partial assignments). *Let a condition vocabulary $Var = (V, Scope)$ be given. If $\chi \in \mathfrak{Ass}_{\subseteq}(Var)$ is defined on $U \subseteq V$, let*

$$Ext(\chi, Var) := \{\chi^* \in \mathfrak{Ass}(Var) \mid \forall u \in U\colon \chi^*(u) = \chi(u)\}$$

denote the set of extensions *of χ.* $\diamond$

2.2 Syntax of E-P3P Policies

An E-P3P policy is a triple of a vocabulary, a set of authorization rules, and a default ruling. The vocabulary defines element hierarchies for data, purposes, users, and actions, as well as the obligation model and the condition vocabulary. Data, users and actions are as in most access-control policies (except that users are typically called “subjects” there, which in privacy would lead to confusion with data subjects), and purposes are an important additional hierarchy for the purpose binding of collected data.

Definition 6 (Vocabulary). *A* vocabulary *is a tuple $Voc = (UH, DH, PH, AH, Var, OM)$ where UH, DH, PH, and AH are hierarchies called user, data, purpose, and action hierarchy, respectively, and Var is a condition vocabulary and OM an obligation model.* $\diamond$

As a naming convention, we assume that the components of a vocabulary called Voc are always called as in Definition 6 with $UH = (U, >_U)$, $DH = (D, >_D)$, $PH = (P, >_P)$, $AH = (A, >_A)$, $Var = (V, Scope)$, and $OM = (O, \rightarrow_O)$, except if explicitly stated otherwise. In a vocabulary called Voc_i all components also get a subscript i, and similarly for superscripts. A rule set contains authorization rules that allow or deny operations. A rule basically consists of one element from each vocabulary component. Additionally, it starts with an integer precedence, and ends with a ruling.

Definition 7 (Ruleset and Privacy Policy). *A* ruleset *for a vocabulary Voc is a subset of $\mathbb{Z} \times U \times D \times P \times A \times C(Var) \times \mathfrak{P}(O) \times \{+, \circ, -\}$.*

A privacy policy *or* E-P3P policy *is a triple (Voc, R, dr) of a vocabulary Voc, a rule-set R for Voc, and a* default ruling *$dr \in \{+, \circ, -\}$. The set of these policies is called EP3P, and the subset for a given vocabulary $EP3P(Voc)$.* $\diamond$

In EPAL, precedences are only given implicitly by the textual order of the rules. Hence our explicit precedences, and the fact that several rules can have the same precedence, make E-P3P a superset of EPAL. The rulings $+$, $\circ$, and $-$ mean ‘allow’, ‘don’t care’, and ‘deny’. The ruling $\circ$ was not yet present in [2]. In EPAL, it is called ‘obligate’ because it enables rules that do not make a decision, but only impose additional obligations. An example is a global rule “Whenever someone tries to access my data, I want to receive a notification”.

As a naming convention, we assume that the components of a privacy policy called Pol are always called as in Definition 7, and if Pol has a sub- or superscript, then so do the components.

2.3 Semantics of E-P3P Policies

An E-P3P request is a tuple (u, d, p, a) which should belong to the set $U \times D \times P \times A$ for the given vocabulary. Note that E-P3P and EPAL requests are not restricted to "ground terms" as in some other languages, i.e., minimal elements in the hierarchies. This is useful if one starts with coarse policies and refines them because elements that are initially minimal may later get children. For instance, the individual users in a "department" of an "enterprise" may not be mentioned in the CPO's privacy policy, but in the department privacy policy. For similar reasons, we also define the semantics for requests outside the given vocabulary. We assume a superset $\mathcal{S}$ in which all hierarchy sets are embedded; in practice it is typically a set of strings or valid XML expressions.

Definition 8 (Request). *For a vocabulary Voc, we define the set of* valid requests *as* $Req(Voc) := U \times D \times P \times A$. *Given a superset* $\mathcal{S}$ *of the sets* U, D, P, A *of all considered vocabularies, the set of* all requests *is* $Req := \mathcal{S}^4$. ◇

The semantics of a privacy policy Pol is a function $eval_{Pol}$ that processes a request based on a given, possibly partial, assignment.

The evaluation result is a pair $(r, \bar{o})$ of a ruling (decision) and associated obligations. Our semantics extends that of [2] in three ways. First, we have to deal with the new partial assignments in the conditions of rules. Secondly, the ruling $\circ$ that was added to the rule syntax gets a semantics; as explained above it is used to make obligations without enforcing a decision. Thirdly, the ruling r may not only be $+$, $\circ$, or $-$ as in a rule, but also $scope\_error$ or $conflict\_error$. This denotes that the request was out of scope of the policy or that there was a conflict among applicable rules. The reason for distinguishing these errors is that out-of-scope errors can be eliminated by enlarging the policy, in contrast to conflict errors. This will become important for policy composition.

The semantics is defined by a virtual pre-processing that unfolds the hierarchies and a request processing stage. Note that this is only a compact definition of the semantics and not an efficient real evaluation algorithm.

Definition 9 (Unfolded Rules). *For a privacy policy* $Pol = (Voc, R, dr)$, *the* unfolded rule set $UR(Pol)$ *is defined as follows:*

$$\begin{aligned}
URdown(Pol) :=\ & \{(i, u', d', p', a', c, \bar{o}, r) \mid \exists (i, u, d, p, a, c, \bar{o}, r) \in R \\
& \quad \text{with } u \geq_U u' \wedge d \geq_D d' \wedge p \geq_P p' \wedge a \geq_A a'\}; \\
UR(Pol) :=\ & URdown(Pol) \\
& \cup\ \{(i, u', d', p', a', c, \bar{o}, -) \mid \exists (i, u, d, p, a, c, \bar{o}, -) \in URdown(Pol) \\
& \quad \text{with } u' \geq_U u \wedge d' \geq_D d \wedge p' \geq_P p \wedge a' \geq_A a\}.
\end{aligned}$$
◇

Note that 'deny'-rules are inherited both downwards and upwards along the four hierarchies while 'allow'-rules are inherited only downwards. The reason is that the hierarchies are considered groupings; if access is forbidden to an element of a group, it is also forbidden for the group as a whole.

Next we define which rules are applicable for a request given a partial assignment of the condition variables. These (unfolded) rules have the user, data, purpose, and action as in the request. Positive rules are only defined to be applicable if they evaluate to $true$ for all extensions of the partial assignment χ. Negative and don't-care rules are defined to be applicable whenever the conditions could still become true. For instance, if a rule forbids access to certain data for minors, a child should not be able to obtain access by omitting its age, and obligations from don't-care rules for children should apply.

Definition 10 (Applicable Rules). *Let a privacy policy* $Pol = (Voc, R, dr)$*, a request* $q = (u, d, p, a) \in Req(Voc)$*, and a partial assignment* $\chi \in \mathfrak{Ass}_{\subseteq}(Var)$ *be given. Then the set of* applicable rules *is*

$$\begin{aligned} &AR(Pol, q, \chi) := \\ &\{(i, u, d, p, a, c, \bar{o}, +) \in UR(Pol) \mid \forall \chi^* \in Ext(\chi, Var)\colon eval_{\chi^*}(c) = true\} \\ \cup\ &\{(i, u, d, p, a, c, \bar{o}, r) \in UR(Pol) \mid r \in \{-, \circ\} \wedge \\ &\quad \exists \chi^* \in Ext(\chi, Var)\colon eval_{\chi^*}(c) = true\}. \end{aligned}$$

◇

For formulating the semantics, we need the maximum and minimum precedence in a policy.

Definition 11 (Precedence Range). *For a privacy policy* $Pol = (Voc, R, dr)$*, let* $max(Pol) := \max\{i \mid \exists (i, u, d, p, a, c, \bar{o}, r) \in R\}$*, and similarly* $min(Pol)$*.* ◇

We can now define the actual semantics, i.e., the result of a request given a partial assignment. Recall that rules with ruling $\circ$ are provided to allow obligations to accumulate before the final decision; this is done in a set $\bar{o}_{acc}$.

Definition 12 (Semantics). *Let a privacy policy* $Pol = (Voc, R, dr)$*, a request* $q = (u, d, p, a) \in Req$*, and a partial assignment* $\chi \in \mathfrak{Ass}_{\subseteq}(Var)$ *be given. Then the* evaluation result $(r, \bar{o}) := eval_{Pol}(q, \chi)$ *of policy* Pol *for* q *and* χ *is defined by the following algorithm, starting with* $\bar{o}_{acc} := \emptyset$*. Every "return" aborts the algorithm.*

– Out-of-scope testing. *If* $q \notin Req(Voc)$*, return* $(r, \bar{o}) := (scope\_error, \emptyset)$*.*
– Processing by precedence. *For each precedence level* $i := max(Pol)$ *down to* $min(Pol)$*:*
  • Accumulate obligations. *For each applicable rule* $(i, u, d, p, a, c, \bar{o}', r) \in AR(Pol, q, \chi)$*, set* $\bar{o}_{acc} := \bar{o}_{acc} \cup \bar{o}'$*.*
  • Conflict detection. *If two conflicting rules* $(i, u, d, p, a, c_1, \bar{o}_1, +)$ *and* $(i, u, d, p, a, c_2, \bar{o}_2, -)$ *exist in* $AR(Pol, q, \chi)$*, return* $(conflict\_error, \emptyset)$*.*
  • Normal ruling. *If at least one rule* $(i, u, d, p, a, c, \bar{o}', r) \in AR(Pol, q, \chi)$ *with* $r \neq \circ$ *exists, return* $(r, \bar{o}_{acc})$*.*
– Default ruling. *If this step is reached, return* $(r, \bar{o}) := (dr, \bar{o}_{acc})$*.*

We also say that policy Pol rules $(r, \bar{o})$ *for* q *and* χ*, omitting* q *and* χ *if they are clear from the context.* ◇

3 Refinement of Privacy Policies

In this section, we define the notion of refinement for E-P3P policies. As explained in the introduction, refinement is the foundation of almost all operations on policies. We further define policy equivalence and show that it equals mutual refinement.

Refining a policy Pol_1 means adding more details, both rules and vocabulary, while retaining its meaning with respect to the original vocabulary. Our notion of refinement allows policy Pol_2 to define a ruling if Pol_1 does not care. Additionally, it is allowed to extend the scope of the original policy and to define arbitrary rules for the new elements. In all other cases, the rulings of both policies must be identical. This also comprises the ruling $\mathit{conflict\_error}$. For new elements however, we have to capture that if they are appended to the existing hierarchies, there could exist applicable rules for these elements if they were already present, and newly added rules for these elements could influence existing elements as well. As an example, a rule for a "department" may forbid its "employees" to access certain data for marketing purposes. Now if a new employee is added, this rule should as well be applicable; furthermore, defining a new rule for this case with higher precedence, e.g., granting the new employee an exception to the department's rule should obviously not yield a refinement any more. In our definition of refinement, we therefore do not evaluate each policy on its own vocabulary but on the joint vocabulary of both policies. Since joining two vocabularies, i.e., joining their respective hierarchies, might not yield another vocabulary, we introduce the notion of compatible vocabularies.

Definition 13 (Compatible Vocabulary). *Two vocabularies Voc_1 and Voc_2 are* compatible *if their condition vocabularies are compatible and all hierarchy unions $UH_1 \cup UH_2$, $DH_1 \cup DH_2$, $PH_1 \cup PH_2$, and $AH_1 \cup AH_2$ are hierarchies again.*

We define the union *of two compatible vocabularies as* $Voc_1 \cup Voc_2 := (UH_1 \cup UH_2, DH_1 \cup DH_2, PH_1 \cup PH_2, AH_1 \cup AH_2, Var_1 \cup Var_2, OM_1 \cup OM_2)$. ◇

Dealing with the respective obligations is somewhat more difficult. Intuitively, one wants to express that a finer policy may also contain refined obligations. However, since a refined policy might contain additional obligations, whereas some others have been omitted, it is not possible to simply compare these obligations in the obligation model of the original policy. (Recall that we also use refinement to compare arbitrary policies; hence one cannot simply expect that all vocabulary parts of the refined policy are supersets of those of the coarser policy.)

As an example, let the obligation model of the coarser policy contain obligations $o =$ "delete in a week" and o_1 = "delete in a month" with the implication $o \rightarrow_{O_1} o_1$. The refined policy contains o_2 = "delete immediately" and o as above with $o_2 \rightarrow_{O_2} o$. Now o_2 should be a refinement of o_1, but this cannot be deduced in either of the obligation models. Hence both obligation models have to be used, i.e., one has $o_2 \rightarrow_{O_2} o \rightarrow_{O_1} o_1$. We define this as *obligation refinement.* In order to obtain a meaningful refinement from the point of view of Pol_1, the relation $\rightarrow_{O_2}$ has to be certified by a party trusted by the maintainer of Pol_1.

Definition 14 (Obligation Refinement). *Let two obligation models* $(O_i, \rightarrow_{O_i})$ *and* $\bar{o}_i \subseteq O_i$ *for* $i = 1, 2$ *be given. Then* $\bar{o}_2$ *is a* refinement *of* $\bar{o}_1$*, written* $\bar{o}_2 \prec \bar{o}_1$*, iff the following holds:*

$$\exists \bar{o} \subseteq O_1 \cap O_2 \colon \bar{o}_2 \rightarrow_{O_2} \bar{o} \rightarrow_{O_1} \bar{o}_1. \qquad \diamond$$

We are now ready to introduce our notion of policy refinement.

Definition 15 (Policy Refinement). *Let two privacy policies* $Pol_i = (Voc_i, R_i, dr_i)$ *for* $i = 1, 2$ *with compatible vocabularies be given, and set* $Pol_i^* = (Voc_i^*, R_i, dr_i)$ *for* $i = 1, 2$*, where* $Voc_i^* = (UH_1 \cup UH_2, DH_1 \cup DH_2, PH_1 \cup PH_2, AH_1 \cup AH_2, Var_i, OM_i)$*. Then* Pol_2 *is a* refinement *of* Pol_1*, written* $Pol_2 \prec Pol_1$*, iff for every assignment* $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$ *and every authorization request* $q \in Req$ *one of the following statements holds, where* $(r_i, \bar{o}_i) := eval_{Pol_i^*}(q, \chi)$ *for* $i := 1, 2$*:*

- $(r_1, \bar{o}_1) = (r_2, \bar{o}_2) = (conflict\_error, \emptyset)$.
- $(r_1, \bar{o}_1) = (scope\_error, \emptyset)$.
- $r_1 \in \{+, -\}$ *and* $r_2 = r_1$ *and* $\bar{o}_2 \prec \bar{o}_1$.
- $r_1 = \circ$ *and* $r_2 \in \{+, \circ, -\}$ *and* $\bar{o}_2 \prec \bar{o}_1$. $\diamond$

Besides this rather strict notion of refinement, we can also define a notion of *weak refinement*, denoted by $\tilde{\prec}$, where the refining policy may be less permissive than the original policy. The only difference to Definition 15 is that $+$ is treated like $\circ$ in the fourth statement instead of like $-$ in the third statement. Weak refinement corresponds to the intuition that a policy Pol_1 implements a privacy promise or requirement to use data at most for certain purposes, so that a refining policy Pol_2 can only restrict that usage. However, while weak definition prevents misuse, it does not preserve guaranteed access rights: For instance, Pol_1 may guarantee an individual the right to read her data while policy Pol_2 does not. Strong refinement therefore seems the more useful notion for E-P3P with its 3-valued logic where $\circ$, meaning 'don't-care', is also a valid ruling. In contrast one might choose weak refinement for a 2-valued policy language with only the rulings $+$ and $-$. We therefore concentrate on strong refinement.

After refinement, we now introduce a notion of equivalence of policies. Similar to policy refinement, we start with the equivalence of obligations.

Definition 16 (Obligation Equivalence). *Let two obligation models* $(O_i, \rightarrow_{O_i})$ *and* $\bar{o}_i \subseteq O_i$ *for* $i = 1, 2$ *be given. Then* $\bar{o}_1$ *and* $\bar{o}_2$ *are* equivalent*, written* $\bar{o}_1 \equiv \bar{o}_2$*, iff* $\bar{o}_1 \prec \bar{o}_2$ *and* $\bar{o}_2 \prec \bar{o}_1$*.* $\diamond$

The relation $\equiv$ is clearly symmetric.

Definition 17 (Policy Equivalence). *Two privacy policies* Pol_1 *and* Pol_2 *with compatible vocabularies are* equivalent*, written* $Pol_1 \equiv Pol_2$*, iff for every assignment* $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$ *and every request* $q \in Req$ *we have*

$$r_1 = r_2 \text{ and } \bar{o}_1 \equiv \bar{o}_2$$

for the evaluation results $(r_1, \bar{o}_1) := eval_{Pol_1}(q, \chi)$ *and* $(r_2, \bar{o}_2) := eval_{Pol_2}(q, \chi)$*.* $\diamond$

Clearly, policy equivalence is a symmetric relation, since obligation equivalence is symmetric. We can now establish the following theorem.

Theorem 1. *Two privacy policies Pol_1, Pol_2 are equivalent if and only if they are mutual refinements. Formally,*

$$Pol_1 \equiv Pol_2 \Leftrightarrow Pol_1 \prec Pol_2 \wedge Pol_2 \prec Pol_1. \qquad \square$$

Proof. Let an assignment $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$ and an authorization request $q \in Req$ be given. Note that $Pol_1 \equiv Pol_2$ implies $Req(Voc_1) = Req(Voc_2)$, as otherwise there exists $q \in Req(Voc_1) \setminus Req(Voc_2)$ without loss of generality such that $eval_{Pol_2}(q, \chi) = (scope\_error, \emptyset) \neq eval_{Pol_1}(q, \chi)$. Similarly, we can show that $Pol_1 \prec Pol_2 \wedge Pol_2 \prec Pol_1$ implies $Req(Voc_1) = Req(Voc_2)$, as for $q \in Req(Voc_1) \setminus Req(Voc_2)$, we have $eval_{Pol_2}(q, \chi) = (scope\_error, \emptyset) \neq eval_{Pol_1}(q, \chi)$, which contradicts $Pol_2 \prec Pol_1$. Therefore, we have $Pol_i = Pol_i^*$ for $i = 1, 2$, with Pol_i^* as in Definition 15, i.e., we can consider the evaluation of Pol_i instead of Pol_i^* to show refinement. Hence let $(r_i, \bar{o}_i) := eval_{Pol_i}(q, \chi) = eval_{Pol_i^*}(q, \chi)$ for $i = 1, 2$ be the corresponding rulings.

"$\Rightarrow$" Since policy equivalence is symmetric, it is sufficient to show that Pol_2 refines Pol_1. If $(r_1, \bar{o}_1) = (conflict\_error, \emptyset)$ then also $(r_2, \bar{o}_2) = (conflict\_error, \emptyset)$ because $Pol_1 \equiv Pol_2$. If $(r_1, \bar{o}_1) = (scope\_error, \emptyset)$, nothing has to be shown. Now let $r_1 \in \{+, \circ, -\}$. Policy equivalence implies $r_2 = r_1$ and $\bar{o}_2 \equiv \bar{o}_1$; this is sufficient for refinement.

"$\Leftarrow$" We distinguish the following cases:

$(r_1, \bar{o}_1) = (conflict\_error, \emptyset)$. Then we also have $(r_2, \bar{o}_2) = (conflict\_error, \emptyset)$ since Pol_2 refines Pol_1. This implies $\bar{o}_1 \equiv \bar{o}_2$.

$(r_1, \bar{o}_1) = (scope\_error, \emptyset)$. If $r_2 \neq r_1$ we immediately obtain that Pol_1 is not a refinement of Pol_2. Thus $r_2 = scope\_error$. This implies $\bar{o}_2 = \emptyset$ and thus $\bar{o}_2 \equiv \bar{o}_1$.

$r_1 \in \{+, -\}$. Then $r_2 = r_1$ since Pol_2 refines Pol_1. Further, since Pol_1 and Pol_2 are mutual refinements, we have $\bar{o}_1 \prec \bar{o}_2$ and $\bar{o}_2 \prec \bar{o}_1$ and thus $\bar{o}_1 \equiv \bar{o}_2$.

$r_1 = \circ$. Assume for contradiction that $r_2 \in \{+, -, scope\_error, conflict\_error\}$. In this case Pol_1 is no refinement of Pol_2 any longer. Further, as in the previous case, we have $\bar{o}_1 \prec \bar{o}_2$ and $\bar{o}_2 \prec \bar{o}_1$ and thus $\bar{o}_1 \equiv \bar{o}_2$. ■

4 Composition of Privacy Policies

In this section, we introduce two notions of composition of E-P3P policies, i.e., the merging of two somehow compatible policies.

In an enterprise, policies may be defined on multiple levels in a management hierarchy. A chief privacy officer (CPO) may define enterprise-wide mandatory policy rules that implement the applicable privacy laws. In addition, the CPO can define defaults that apply if a department does not define its own rules. A department can then define its own privacy policy rules. These rules override the default rules but are overruled by the mandatory rules of the CPO. In order to allow such distributed authoring and maintenance of privacy policies, we now introduce a notion of policy composition. If two policies are composed, both rule-sets are enforced. By defining that one policy has a higher precedence than the other, one can define one way to resolve conflicts. For such precedence shifts and for dealing with default values, we start with the notion of the *normalization* of a policy.

4.1 Policy Normalization

Recall that the default ruling of a policy determines the result if no rule applies for a given request although the request is in the scope of the policy. When composing policies, different default rulings must be resolved first. This is simple if the scope is the same or the default ruling is the same. To resolve the more challenging cases, we first convert the default ruling of a policy into a set of normal rules. These new rules have the default ruling as their ruling, lowest precedence, no obligations and conditions, and they cover the root elements of all hierarchies.

Definition 18 (Policy with Removed Default Ruling). *Let* $Pol = (Voc, R, dr)$ *be a privacy policy and* $\underline{i} \in \mathbb{Z}$. *Then the* policy with removed default ruling *for* Pol *wrt.* $\underline{i}$ *is the following policy* $rmDR(Pol, \underline{i})$*:*

If $dr = \circ$, *then* $rmDR(Pol, \underline{i}) := Pol$.

Else for every hierarchy $XH = (H, >_H)$, *let* $roots(XH) := \{x \in H \mid \neg\exists x' \in H\colon x' >_H x\}$. *Then* $rmDR(Pol, \underline{i}) := (Voc, R', \circ)$ *with* $R' := R \cup DR$ *and*

$$DR := \{(\underline{i}, u, d, p, a, \emptyset, \emptyset, dr) \bigm| u \in roots(UH) \wedge\ d \in roots(DH) \\ \wedge\ p \in roots(PH) \wedge\ a \in roots(AH)\}.$$

We abbreviate $rmDR(Pol) := rmDR(Pol, min(Pol) - 1)$. $\diamond$

We now show that a policy with removed default ruling is equivalent to the original policy if $\underline{i}$ is smaller than all the precedences in the original policy.

Lemma 1. *Let* $Pol = (Voc, R, dr)$ *be a privacy policy and* $\underline{i} \in \mathbb{Z}$ *with* $\underline{i} < min(Pol)$. *Then* $rmDR(Pol, \underline{i}) \equiv Pol$. *In particular, this implies* $rmDR(Pol) \equiv Pol$. $\square$

Proof. Let a request $q \in Req$ and an assignment $\chi \in \mathfrak{Ass}_{\subseteq}(Var)$ be given. Since Pol and $rmDR(Pol)$ have the same vocabulary, we can show refinement using the evaluation of Pol and $rmDR(Pol)$ instead of Pol^* and $rmDR(Pol)^*$ as defined in Definition 15. Equal vocabularies also imply that either both policies rule $(scope\_error, \emptyset)$ or none of them.

If Pol rules $(conflict\_error, \emptyset)$ then so does $rmDR(Pol, \underline{i})$, since the rules in DR have lower precedence than all rules in R. Furthermore, all rules in DR have identical ruling; hence they cannot induce a conflict error. Thus either both policies rule $(conflict\_error, \emptyset)$ or none of them.

If a rule ρ from R applies to this request, it applies for both policies, since every rule of Pol is also contained in the ruleset of $rmDR(Pol, \underline{i})$. Moreover, every rule in DR has lower priority than ρ by construction. Hence neither a rule in DR nor the default ruling applies. Thus Pol and $rmDR(Pol, \underline{i})$ output the same pair $(r, \bar{o})$. Conversely, if a rule $\rho \in DR$ applies, this means that no rule of R applies, but the request is in scope of the policy. In this case Pol outputs $(dr, \bar{o}_{acc})$, where $\bar{o}_{acc}$ is the set of obligations accumulated while processing R. The policy $rmDR(Pol, \underline{i})$ applies the rule ρ, which also yields $(dr, \bar{o}_{acc})$ since no obligation is added by any rule in DR. ■

Next, we introduce an operation for changing the precedence of the rules of a policy, e.g., to overcome possible conflicts when merging the policy with another one. As a collective change for all rules seems useful, we define a precedence shift, which adds a fixed number to the precedence of all rules in a policy. This is particularly useful for the example at the beginning of this section, where the department policy can be shifted downwards to have lower precedences than the policy of the CPO.

Definition 19 (Precedence Shift). *Let $Pol = (Voc, R, dr)$ be a privacy policy and $j \in \mathbb{Z}$. Then $Pol + j := (Voc, R + j, dr)$ with $R + j := \{(i + j, u, d, p, a, c, \bar{o}, r) \mid (i, u, d, p, a, c, \bar{o}, r) \in R\}$ is called the* precedence shift *of Pol by j. We define $Pol - j := Pol + (-j)$.* ◇

Lemma 2. *A privacy policy Pol is equivalent to $Pol + j$ for all $j \in \mathbb{Z}$.* □

Proof. By Theorem 1, it is sufficient to show that Pol refines $Pol + j$ for all $j \in \mathbb{Z}$.

Let j, a request $q \in Req$, and an assignment $\chi \in \mathfrak{Ass}_{\subseteq}(Var)$ be given. By construction, the rules that apply to this request in Pol and $Pol + j$ can only differ in their precedence. Furthermore, if a rule applies in Pol, it also applies after the precedence shift, since the precedence of *all* rules in $Pol + j$ has been shifted similarly. Applying the same rule in both policies in particular yields the same set of obligations. Hence Pol refines $Pol + j$. ■

We now define the normalization of a policy. This corresponds to a precedence shift yielding a default ruling of precedence 0. Normalized policies are used in the definition of the ordered composition of two policies defined below.

Definition 20 (Normalized Policy). *Let Pol be a privacy policy. Then $\mathrm{norm}_0(Pol) := rmDR(Pol - min(Pol) + 1, 0)$ is called the* normalized policy *for Pol.* ◇

Lemma 3. *For every privacy policy Pol, we have $\mathrm{norm}_0(Pol) \equiv Pol$.* □

Proof. By definition, the precedence of all rules in $\mathrm{norm}_0(Pol)$ is greater than 0. Hence the claim follows from Lemmas 1 and 2. ■

4.2 Definition of Composition

We now have the tools ready to turn our attention to policy composition. The simplest case of merging two policies with compatible vocabularies is to compute the union of the two rule sets. This direct composition assumes that the precedences used in both policies have a common meaning. However, translating the default ruling of a policy is tricky: If elements are in the scope of only one policy, the default ruling of this policy should apply. If elements are covered by both policies, a conflict between the two default rulings needs to be resolved.

Definition 21 (Direct Composition). *Let Pol_1 and Pol_2 be two privacy policies with compatible vocabularies. Let $m := \min\{min(Pol_1), min(Pol_2)\}$ and $Pol'_i := (Voc_i, R'_i, \circ) := rmDR(Pol_i, m)$ for $i = 1, 2$. Then*

$$Pol_1 \bigcup Pol_2 := (Voc_1 \cup Voc_2, R'_1 \cup R'_2, \circ)$$

is called the direct composition *of Pol_1 and Pol_2.* ◇

In our second type of composition, the rules of one policy, here Pol_2 should be applied preferably. Hence the rules of the other policy are downgraded using a precedence shift. This also applies to the default ruling of the preferred policy, i.e., the downgraded policy is only used where it extends the scope of the preferred policy, or if no rule of the preferred policy applies and the default ruling is $\circ$.

Definition 22 (Ordered Composition). *Let Pol_1 and Pol_2 be two privacy policies with compatible vocabularies. Let $(Voc_1, R''_1, \circ) := rmDR(Pol_1 - max(Pol_1) - 1)$, and $(Voc_2, R'_2, \circ) := \mathrm{norm}_0(Pol_2)$. Then*

$$Pol_1 \overset{<}{\bigcup} Pol_2 := (Voc_1 \cup Voc_2, R''_1 \cup R'_2, \circ)$$

is called the ordered composition *of Pol_1 under Pol_2.* ◇

By the intuitive introduction to ordered compositions, an ordered composition should serve as a refinement of Pol_2. This is captured in the following lemma.

Lemma 4. *For all privacy policies Pol_1 and Pol_2 with compatible vocabularies, we have $Pol_1 \overset{<}{\bigcup} Pol_2 \prec Pol_2$.* □

Proof. Let $Pol_i =: (Voc_i, R_i, dr_i)$ for $i := 1, 2$, and we use all notation from Definition 22. Let a request $q = (u, d, p, a)$ and an assignment $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$ be given. Let Pol_2^* and $(Pol_1 \overset{<}{\bigcup} Pol_2)^*$ be defined according to Definition 15. Note further that $(Pol_1 \overset{<}{\bigcup} Pol_2)^* = Pol_1 \overset{<}{\bigcup} Pol_2$ since their vocabularies are equal. We distinguish four cases:

1. Pol_2^* rules $(conflict\_error, \emptyset)$: In this case, two rules of the same precedence i collided in Pol_2^*. Since i is larger than the precedence of any rule of R''_1 by the shifts, we also get a conflict in $Pol_1 \overset{<}{\bigcup} Pol_2$, and an output $(conflict\_error, \emptyset)$.
2. Pol_2^* rules $(scope\_error, \emptyset)$: Nothing has to be shown for this case.
3. Pol_2^* rules $(r_2, \bar{o}_2)$ with $r_2 \neq \circ$: This means that a rule $(i_2, u, d, p, a, c_2, \bar{o}_2, r_2)$ from R'_2 is applicable. Because of $r_2 \neq \circ$ the evaluation function will stop at the precedence level i_2. Since the precedences of R''_1 are always less then zero by construction whereas $i_2 \geq 0$ by normalization, the same rules applies in $Pol_1 \overset{<}{\bigcup} Pol_2$ and we obtain identical outputs, i.e., $Pol_1 \overset{<}{\bigcup} Pol_2$ also rules $(r_2, \bar{o}_2)$. Note that several rules might have already occurred that added obligations only. However, they occur in both Pol_2^* and $Pol_1 \overset{<}{\bigcup} Pol_2$ and hence do not cause any harm.
4. No rule of R'_2 fits the current request, but the request is in the scope of Pol_2^*. In this case, the (expanded) default ruling of Pol_2 applies for both $Pol_1 \overset{<}{\bigcup} Pol_2$ and Pol_2^*, since this ruling has higher precedence than any rule in R''_1. If $dr_2 \neq \circ$,

both policies rule $(\circ, \bar{o}_2)$ for some obligation set $\bar{o}_2$, which is again equal for both policies since they processed the same set of rules so far. If $dr_2 = \circ$ then the evaluation function starts searching for matching rules in R_1''. Here Pol_2^* outputs $(\circ, \bar{o}_2)$, whereas $Pol_1 \uplus^{<} Pol_2$ outputs $(r, \bar{o}_2 \cup \bar{o}_1)$ for some r and $\bar{o}_1 \subseteq O_1$. In order to prove refinement, we obtain $\bar{o}_1 \cup \bar{o}_2 \rightarrow_{O_2} \bar{o}_2$ because of $\bar{o}_2 \subseteq \bar{o}_1 \cup \bar{o}_2$. This further implies $\bar{o}_1 \cup \bar{o}_2 \rightarrow_{O_1 \cup O_2} \bar{o}_2$. Because of $\bar{o}_2 \rightarrow_{O_2} \bar{o}_2$ we obtain $\bar{o}_1 \cup \bar{o}_2 \rightarrow_{O_1 \cup O_2} \bar{o}_2 \rightarrow_{O_2} \bar{o}_2$, which proves refinement since $\bar{o}_2 \in O_2 = (O_1 \cup O_2) \cap O_2$. ■

The composition operators fulfill certain laws:

Lemma 5 (Laws for Policy Composition). *Direct composition is commutative and ordered composition is associative, i.e., for all privacy policies Pol_i for $i = 1, 2, 3$ with pairwise compatible vocabularies, we have*

$$Pol_1 \bigcup Pol_2 \equiv Pol_2 \bigcup Pol_1;$$

$$\left(Pol_1 \uplus^{<} Pol_2\right) \uplus^{<} Pol_3 \equiv Pol_1 \uplus^{<} \left(Pol_2 \uplus^{<} Pol_3\right). \qquad \square$$

Proof. (Sketch) The commutativity part is obvious. We now show the associativity part. The composition of vocabularies is associative. For the rulesets, $(Pol_1 \uplus^{<} Pol_2) \uplus^{<} Pol_3$ yields a ruleset where 0 is the lowest precedence of Pol_3 and higher than the highest precedence of Pol_2, while $Pol_1 \uplus^{<} (Pol_2 \uplus^{<} Pol_3)$ yields a ruleset where 0 is the lowest precedence of Pol_2 and higher than the highest precedence of Pol_1. By shifting the second ruleset by $max(Pol_2) - min(Pol_2) + 1$ one obtains a ruleset that is identical, and thus clearly equivalent, to the first set. Since this precedence shift retains equivalence, the second ruleset is equivalent to the first ruleset. ■

5 Two-Layered Privacy Policies

An enterprise must abide by the law. In addition, the consent that an individual has granted when submitting data to an enterprise is mandatory and should not be changed. In contrast, unregulated and non-promised issues can be freely decided by the enterprise, i.e., enterprise privacy practices can be changed by the CPO or the administrators of the enterprise. In order to reflect this requirement, we now introduce a distinction between mandatory and discretionary parts of a policy. This represents a modal view of a policy semantics [18]: The mandatory part *must* be adhered to under any circumstances, the remaining part *may* be adhered to. We capture this view by introducing *two-layered policies*.

The real value of this notion lies in new possibilities for composition that cannot be captured by just taking the ordered composition of the discretionary part under the mandatory part.

5.1 Syntax and Semantics of Two-Layered Privacy Policies

Syntactically, a two-layered policy is simply a pair of (usual) privacy policies. The first element is the mandatory part and the second element the discretionary part.

Definition 23. *A pair $Pol = (Pol_1, Pol_2)$ of privacy policies with compatible vocabularies is called a* two-layered policy. *The policy Pol_1 is called the* mandatory part *of Pol, and Pol_2 the* discretionary part. $\diamond$

The semantics of such a two-layered policy is described as an algorithm, given an authorization request $q \in Req$ and an assignment $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$:

- *Evaluate mandatory policy.* Evaluate the request q under Pol_1, yielding $(r_1, \bar{o}_1)$.
- *First policy dominates.* If $r_1 \notin \{\circ, scope\_error\}$, output $(r_1, \bar{o}_1)$.
- *Evaluate second policy.* If $r_1 \in \{\circ, scope\_error\}$, evaluate the request q under Pol_2, yielding $(r_2, \bar{o}_2)$. If $r_2 = scope\_error$ and $r_1 = \circ$, output $(r_1, \bar{o}_1)$, else output $(r_2, \bar{o}_1 \cup \bar{o}_2)$.

This captures the intuition that Pol_1 is mandatory: Only if Pol_1 does not care about a request, or if it does not capture the request, the discretionary part Pol_2 is executed. Note that the mandatory part can still be used to accumulate obligations, e.g., for strictly requiring that every employee has to send a notification to his or her manager before a specific action.

We now show that the resulting semantics is the same as that of ordered composition, as one would expect. Recall, however, that the composition operators make use of the fact that a two-layered policy retains the information which parts were mandatory.

Lemma 6. *For $Pol := (Pol_1, Pol_2)$, the two-layered semantics is equivalent to the ordinary semantics of an ordered composition: $Pol \equiv Pol_2 \biguplus Pol_1$.* $\square$

Proof. We have to show that evaluation of Pol and $PolC := Pol_2 \biguplus Pol_1$ always return the same ruling and equivalent obligations.. Let us first assume that the evaluation of Pol outputs $(r_1, \bar{o}_1)$ with $r_1 \in \{+, -\}$ (first policy dominates), i.e., the request was in the scope of Pol_1 and produced a ruling $+$ or $-$. Hence when evaluating $PolC$, only higher precedence rules of Pol_1 are used and $PolC$ also outputs $(r_1, \bar{o}_1)$.

Let us now assume that $r_1 = scope\_error$ in Pol. Then the result is determined by Pol_2. The same holds for $PolC$.

Let us finally assume that $(r_1, \bar{o}_1)$ was output by Pol_1 with $r_1 = \circ$. If the request is in the scope of Pol_2, then Pol and $PolC$ will output the ruling of Pol_2 with a union of both obligations. If the request is out of the scope of Pol_2, the pair $(r_1, \bar{o}_1)$ is output in both cases. $\blacksquare$

This lemma and Lemma 4 imply that $(Pol_1, Pol_2) \prec Pol_1$, i.e., that every two-layered policy refines its mandatory part.

5.2 Refinement and Composition of Two-Layered Policies

For two-layered privacy policies, we define the following notion of refinement.

Definition 24 (Two-Layered Refinement). *Let two-layered policies $Pol = (Pol_1, Pol_2)$ and $Q = (Q_1, Q_2)$ be given where Pol_1 and Q_1 as well as Pol_2 and Q_2 have compatible vocabularies. Then Pol is a* refinement *of Q, written $Pol \prec Q$, iff $Pol_1 \prec Q_1$ and $Pol_2 \tilde{\prec} Q_2$.* $\diamond$

The distinction between mandatory and discretionary parts enabled us to use the notion of weak refinement: We require that the mandatory part is a normal refinement. This reflects that access rights as well as denials must be preserved. The discretionary part may be weakly refined. This reflects the fact that this part can be modified at the discretion of the enterprise.

Coming up with a meaningful definition of composing two-layered policies is more difficult. The main goal is to never violate any mandatory part. Composing the mandatory parts either according to Definition 21 or 22 would typically overrule some mandatory rules, which would defeat this goal. Therefore, we only allow to compose two-layered policies if the rulesets of their respective mandatory parts are *collision-free*, which means that for all requests and all assignments, it is never the case that one mandatory part accepts the request (i.e., it rules $+$), whereas the other one denies the request (i.e., it rules $-$).

Definition 25 (Collision-Free). *Two privacy policies Pol_1 and Pol_2 with compatible condition vocabularies are called* collision-free *if for all requests $q \in Req$ and all assignments $\chi \in \mathfrak{Ass}_{\subseteq}(Var_1 \cup Var_2)$ the following holds: If Pol_i rules $(r_i, \bar{o}_i)$ for $i = 1, 2$, then $\{r_1, r_2\} \neq \{+, -\}$.* $\diamond$

Two-layered policies with conflicting mandatory parts cannot be composed. For two-layered policies with collision-free mandatory parts, we define composition as follows:

Definition 26 (Two-Layered Composition). *Let two-layered policies $Pol = (Pol_1, Pol_2)$ and $Q = (Q_1, Q_2)$ be given where Pol_1 and Q_1 as well as Pol_2 and Q_2 have compatible vocabularies, and Pol_1 and Q_1 are collision-free. Then the* composition *of Pol and Q is defined as $Pol \bigcup Q := (Pol_1 \bigcup Q_1, Pol_2 \bigcup Q_2)$. Similarly, the* ordered composition *of Pol under Q is defined as $Pol \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q := (Pol_1 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_1, Pol_2 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_2)$.* $\diamond$

The following lemma shows the main property that this composition wants to preserve, i.e., that the resulting composed policy refines both mandatory parts.

Lemma 7. *Let two-layered policies $Pol = (Pol_1, Pol_2)$ and $Q = (Q_1, Q_2)$ be given such that Pol_1 and Q_1 as well as Pol_2 and Q_2 have compatible vocabularies. Moreover, Pol_1 and Q_1 are collision-free. Then $Pol \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q$ is a refinement of Q. Furthermore, $PolC := Pol \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q$ is a refinement of Pol_1 and Q_1.* $\square$

Proof. The first claim follows directly from Lemma 4. Lemma 6 implies that $PolC \equiv (Pol_2 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_2) \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, (Pol_1 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_1)$. The fact that $PolC$ is a refinement of Q_1 follows from Lemma 4.

Let us now assume that $PolC$ does not refine Pol_1. Since $PolC$ refines $(Pol_1 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_1)$, this implies that $(Pol_1 \,\underline{\cup}\!\!\!\!\!\!\;\,\,\,\lhd\, Q_1)$ does not refine Pol_1. This implies that there exists a request within the scope of Pol_1 and Q_1 such that Q_1 does not rule $\circ$ since otherwise the ruling of Pol_1 would be output. For this request, the rulings of Pol_1 and Q_1 differ, which contradicts our assumption that Pol_1 and Q_1 are collision-free. $\blacksquare$

6 Conclusion

Privacy policies are a core component for enterprise privacy technologies. The current proposals for privacy policies required policy authors to create one single overall policy for the complete enterprise(s) that are covered. We therefore described a toolkit to handle multiple policies. This includes refinement for auditing and policy validation and composition for multi-domain or delegated-authorship policies. In addition, we introduced a new notion of two-layered policies to track mandatory and discretionary parts. This enables privacy administrators to detect and resolve conflicts between mandatory policies, e.g., if a customer promise or another contract would violate a law. These tools enable the privacy officers of an enterprise to create and manage the complex privacy policy of an enterprise more efficiently while retaining the semantic rigor that is required for trustworthy privacy management.

Acknowledgments

We thank Günter Karjoth, Calvin Powers, and Michael Waidner for valuable discussions.

References

1. P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL). Research Report 3485, IBM Research, 2003. `http://www.zurich.ibm.com/security/enterprise-privacy/epal/specification`.
2. P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-P3P privacy policies and privacy authorization. In *Proc. 1st ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–109, 2002.
3. A. Belokosztolszki and K. Moody. Meta-policies for distributed role-based access control systems. In *Proc. 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 106–115, 2002.
4. C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekerat. Obligation monitoring in policy management. In *Proc. 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 2–12, 2002.
5. P. A. Bonatti, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. A component-based architecture for secure data publication. In *Proc. 17th Annual Computer Security Applications Conference*, pages 309–318, 2001.
6. P. A. Bonatti, S. De Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 164–173, 2000.
7. P. A. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1):1–35, 2002.
8. A. Cavoukian and T. J. Hamilton. *The Privacy Payoff: How successful businesses build customer trust*. McGraw-Hill/Ryerson, 2002.
9. S. De Capitani di Vimercati and P. Samarati. An authorization model for federated systems. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 1996.

10. S. Fischer-Hübner. *IT-security and privacy: Design and use of privacy-enhancing security mechanisms*, volume 1958 of *Lecture Notes in Computer Science*. Springer, 2002.
11. Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu. IPSec/VPN security policy: Correctness, conflict detection and resolution. In *Proc. 2nd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, volume 1995 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2001.
12. V. Gligor, H. Khurana, R. Koleva, V. Bharadwaj, and J. Baras. On the negotiation of access control policies. In *Proc. 9th International Workshop on Security Protocols*, 2002.
13. H. Hosmer. The multipolicy paradigm. In *Proc. 15th National Computer Security Conference*, pages 409–422, 1993.
14. S. Jajodia, M. Kudo, and V. S. Subrahmanian. Provisional authorization. In *Proc. E-commerce Security and Privacy*, pages 133–159. Kluwer Academic Publishers, 2001.
15. S. Jajodia, P. Samarati, M. L. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(4):216–260, 2001.
16. G. Karjoth and M. Schunter. A privacy policy model for enterprises. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 271–281, 2002.
17. G. Karjoth, M. Schunter, and M. Waidner. The platform for enterprise privacy practices – privacy-enabled management of customer data. In *Proc. Privacy Enhancing Technologies Conference*, volume 2482 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2002.
18. J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE JSAC Special Issue on Network Management*, 11(9):1404–31414, 1993.
19. Platform for Privacy Preferences (P3P). W3C Recommendation, Apr. 2002. `http://www.w3.org/TR/2002/REC-P3P-20020416/`.
20. C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies with complex constraints. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2001.
21. TRUSTe. Privacy Certification. Available at `www.truste.com`.
22. eXtensible Access Control Markup Language (XACML). OASIS Committee Specification 1.0, Dec. 2002. `www.oasis-open.org/committees/xacml`.

Authenticating Mandatory Access Controls and Preserving Privacy for a High-Assurance Smart Card

Helmut Scherzer[1], Ran Canetti[2], Paul A. Karger[2],
Hugo Krawczyk[2,3], Tal Rabin[2], and David C. Toll[2]

[1] IBM Deutschland GmbH, Secure Systems and Smart Cards,
Schönaicher Str. 220, D-71032 Böblingen, Germany
scherzer@de.ibm.com
[2] IBM Research Division, T. J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598, USA
{canetti,talr,toll}@us.ibm.com, {karger,hugo}@watson.ibm.com
[3] Department of Electrical Engineering, Technion, Haifa, 32000, Israel
hugo@ee.technion.ac.il

Abstract. This paper presents an authentication protocol for high-assurance smart card operating systems that support download of mutually suspicious applications. Such a protocol is required to be part of the operating system, rather than the traditional smart card approach of allowing applications to do authentication, because strong authentication is essential for the operating system to protect one application from another. The protocol itself is based on the existing IKE protocol [13], used for authentication in IPSEC. What is new is the integration of an IKE-like protocol with authentication of mandatory secrecy and integrity access controls, the recognition that a single PKI-hierarchy cannot certify identity and all possible mandatory access rights, and the use of IKE to resolve privacy problems found in existing smart card authentication protocols.

1 Caernarvon – A High-Assurance Smart Card OS

IBM® has been developing a secure smart card operating system, called Caernarvon, that is intended to be evaluated at the highest assurance levels of the Common Criteria [17]. The Caernarvon system is written for, and requires, processors containing hardware security features that provide separate supervisor and user modes, and hardware memory protection. The security of the system is then enforced by the hardware, rather than being entirely dependent on software. The initial version is being written for the Philips Smart*XA2* processor, which is a 16-bit smart card processor containing the required hardware security features.

The operating system is designed to permit applications developers to solve security-related problems that could never be addressed before, either in smart cards or in larger computer systems. In particular, the security model and the operating system are designed to permit in-the-field downloading of applications written either in native languages (such as C or assembler) or in interpreted languages (such as Java Card^{TM}). These downloaded applications could be mutually hostile, yet the operating system will prevent unauthorized interference between the applications, yet still allow controlled sharing of information between selected applications, subject to the constraints of the new security model. The applications themselves do not need to be evaluated or certified. The only obvious exception to this is the special *guard* applications that are privileged to change the security markings of files.

The Caernarvon system contains a Mandatory Security Policy; this consists of a modified Bell and LaPadula [3] lattice secrecy model, together with a modified Biba [4] integrity model. Space does not permit discussion of the Caernarvon Mandatory Security Policy in detail, but policy is described in more detail in [22,23,24,29]. Girard [12] has also suggested the use of mandatory security policies in smart cards.

2 High Assurance and Authentication

One of the fundamental requirements for a high assurance operating system is authentication. This authentication must be performed by the operating system itself, not by an application, so that the operating system both is guaranteed and can guarantee to others that the authentication has been completed. The operating system then knows, with high assurance, the identity of the *user*, in this case the outside world, namely the system behind the smart card reader. The system can use this knowledge to safely grant the user access to files and other system objects; conversely, it can also use this knowledge to ensure, with high assurance, that a user is denied access to anything he is not authorized to see or use.

If the operating system were to follow the current smart card practice of delegating all authentication to the application, then true protection of applications from one another is impossible. For example, if authentication were delegated to a poorly written vending machine application, it might claim mistakenly to have properly authenticated the card issuer's primary administrator. Without strong authentication, such as that provided by the Caernarvon operating system, the vending machine application could mistakenly grant access to the card issuer's administrative files stored on the card.

Furthermore, even operating systems that have been successfully evaluated and certified to the highest levels of the Common Criteria might not be perfect. Authentication provides the first line of defense in any operating system, and should an attacker get past an application-provided authentication scheme, even if only improperly authenticated as a legitimate user of that application, then that attacker has gained additional access to software running on the card that could then be targeted. While a successful high-assurance evaluation gives a very high level of confidence that attack is not possible, there will always remain the small possibility of a vulnerability that escaped detection. The U.S. National Computer Security Center recognized this possibility in its so-called, "Yellow Book" [31] that provided guidance on when to use operating systems that have been evaluated to different levels. The “Yellow Book” recognized that even an A1-evaluated operating system, the highest level possible under the “Orange Book” [10], was not sufficiently strong to protect the very highest levels of compartmented intelligence information against totally uncleared users.

3 Authenticating Multi-organizational Mandatory Access Controls

Caernarvon is designed to support mandatory access controls in a commercial setting. This means that the lattice security models must contain entries from multiple organizations. This is very different from most military applications of mandatory access controls where the access controls have a common definition. It is beyond the scope of this paper to discuss the implications of mandatory access controls from different organizations. An early discussion of some of these issues can be found in [20], and a preliminary design for multi-organizational mandatory access controls can be found in [21], although this design has already undergone a number of changes. For purposes of this paper, it is sufficient to say that the smart card must be able to handle mandatory access controls from multiple organizations.

Both the Bell and LaPadula secrecy model [3] and the Biba integrity model [4] provide a lattice structure of non-hierarchic access classes. Each object in the system is assigned an access class, and each user is assigned a security clearance that is also an access class. Access control decisions are made by comparing the access class of an object with the access class of the referencing user or process. The details of access classes are unimportant to this paper. What is important is that both the security and integrity lattices may contain access classes from different mutually suspicious organizations, and that possession of these access classes must be authenticated. Note that this type of multi-organizational access class is much more general than the access classes typically used in the US Department of Defense, such as those defined in FIPS PUB 188 [30] or the DoD Common Security Label [8].

Proving that either a server or a card actually holds a particular organizational access class is not easy. The conventional approach of a Public Key Infrastructure (PKI) with a trusted third-party Certification Authority (CA) is not likely to be acceptable, because of the extreme sensitivity of mandatory access classes in some applications. For example, despite the fact that the NATO countries are all close allies, there is not likely to be a single CA that all of the NATO countries would accept to prove that someone holds a particular national security clearance.

Instead, a organizational Security Authority (SA) is defined, which serves as the agency to digitally sign certificates that prove that a particular organizational access class is held. An SA is very similar to a CA, except that there is no hierarchical tree above the various SAs. Each SA stands alone.

Therefore, both servers and smart cards must hold digital signatures from multiple organizations’ SAs and must be prepared to verify those digital signatures. Each SA with an access class on the card or on a server must sign a hash of that organization's access class, cryptographically bound to the public key of the card or server.

IBM has developed a protocol for defining access classes and SAs on a smart car. However, that protocol is beyond the scope and page limitations of this paper, and will be the subject of a future paper. This paper will simply assume that SA public keys for the appropriate organizations are available on the card.

The mandatory security policy requires that an effective authentication of its users (in this case, the outside world) be performed before allowing access to objects under the control of the operating system. The security policy in particular, applies to application download, application selection, and inter-application information sharing and exchange.

The authentication scheme must be application independent, and enforced by the Caernarvon kernel. The authentication protocol as described here has been submitted to the ESIGN-K [2] working group that is designing a specification for the European signature application on smart cards and to the Global Platform organization (http://www.globalplatform.org) that is developing standards for multi-application smart cards based on earlier work by Visa.

The Caernarvon authentication combines four mechanisms:

– a device verifies the existence of a certified secret key on the other party.
– the devices negotiate or exchange information to establish a common session key for subsequent operations.
– the devices negotiate or exchange information to establish a common access class for subsequent operations.
– the authentication is mutual, that is it is two-way, and binds all of the above elements.

The session encryption uses a symmetric algorithm, for performance reasons. Therefore this document describes the derivation of symmetric keys, and does not consider an option for asymmetric keys. Once the session key is established, a trusted channel is available to protect or conceal the information transmitted over the interface. The application of Secure Messaging (SM) is mandatory for subsequent operations to ensure the provision of a trusted channel.

There are current smart card standards for digital signature applications [6,7] that use a two-way authenticated Diffie-Hellman key agreement scheme in order to create a triple-DES session key for the current session. This key agreement scheme is described in ISO/IEC 11770-3 [19], section 6.7 “Key agreement mechanism 7”; this in turn is based on the three-pass authentication mechanism of ISO/IEC 9798-3 [18]. However, we show in section 5 below, that these existing smart card standards have privacy problems. Therefore, Caernarvon uses a Diffie-Hellman key agreement scheme based on the SIGMA design [25], standardized by the Internet Key Exchange Protocol (IKE) [13]. This was chosen so that authentication for Caernarvon could be based on existing standards, and because the security and privacy properties of SIGMA protocols have been formally proven in [5].

The cryptographic principles behind Caernarvon authentication are not new. What is new is the application of IKE to smart cards to resolve the privacy problems in existing smart card standards and the combination of IKE authentication with multi-organizational mandatory access controls in a high-assurance operating system. The Caernarvon protocol also solves a potential problem of a man-in-the-middle modifying the Diffie-Hellman public parameters. In normal usage, IKE does not suffer from this problem, but it can arise in other contexts. See section 6 for details.

4 Diffie-Hellman Key Exchange

Diffie-Hellman was the first public-key algorithm openly published in 1976 [9]. The Diffie-Hellman algorithm was first developed by M. J. Williamson at the Communications-Electronics Security Group (CESG) in the UK and published internally somewhat later in [33], but that work remained classified until much later [11]. It gets its security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of performing exponentiation calculations in the same field.

4.1 Simple Diffie-Hellman Key Exchange, No Authentication

The general form of an unauthenticated Diffie-Hellman key exchange is as follows: in this, A is the card reader and B is the Caernarvon smart card.

Both A (the card reader) and B (Caernarvon) share the public quantities p, q and g where:

- p is the modulus, a prime number; for security, this number should be of the order of 1024 bits or more.
- q is a prime number in the range of 159-160 bits
- g is a generator of order q, that is $g^q = 1 \bmod p$ and for all i where $i < q$, $g^i \neq 1 \bmod p$

Then the protocol proceeds as follows:

1. A chooses random number a with $1 \leq a \leq q-1$, and B chooses random number b with $1 \leq b \leq q-1$
2. A computes $K_A = g^a \bmod p$ and sends it to B. B computes $K_B = g^b \bmod p$ and sends it to A
3. A computes $K_{AB} = (K_B)^a \bmod p$. B computes $K_{BA} = (K_A)^b \bmod p$.
4. A deletes the random number a. B deletes the random number b.

A and B have derived the same number (the key) because:

$$K_{AB} = (K_B)^a = (g^b)^a = (g^a)^b = (K_A)^b = K_{BA}$$

4.2 Authenticated Diffie Hellman

The algorithm described above in section 4.1 produces an agreed secret key; however, because there is no authentication included, it is vulnerable to man-in-the-middle attacks. ISO/IEC 11770-3 [19] section 6.7 “Key agreement mechanism 7” uses a Diffie-Hellman style of key exchange, and also involves mutual authentication. The following shows the general flow of such a scheme.

Stage 1. The reader, A, calculates K_A as above, and concatenates it with a certificate containing A's identity A and public key PK_A. A transmits this to B:

$$\begin{tikzcd}\mathbf{A} \arrow[r, "Cert(A\|PK_A)_{CA}\|K_A"] & \mathbf{B}\end{tikzcd}$$

Fig. 1. Key Agreement Mechanism 7 of ISO 11770-3, Part 1

Stage 2. The card, B, checks the certificate received from A. B then calculates K_B and K_{BA} as above, and generates a message for A consisting of the following items concatenated together:

1. K_B
2. a certificate containing B's identity B and public key PK_B
3. a signature of the concatenation of A's identity A, K_A and K_B, signed using B's secret key SK_B corresponding to the public key PK_B
   Note that this signature consists of just the signature value - it is not necessary to include the quantities being signed, since A already has them (they were sent to B in Stage 1 above), or are included as part of this message.
4. a cryptographic check, using a keyed hash function **f** as a message authentication code with the key K_{BA}, of A's identity A, K_A and K_B.

B then transmits this to A:

$$\begin{tikzcd}\mathbf{A} & \mathbf{B} \arrow[l, "K_B\|Cert(B\|PK_B)_{CA}\|Sig_{SK_B}(A\|K_A\|K_B)\|\mathbf{f}_{K_{BA}}(A\|K_A\|K_B)"']\end{tikzcd}$$

Fig. 2. Key Agreement Mechanism 7 of ISO 11770-3, Part 2

Stage 3. A now checks the certificate received from B, verifies the signature (using B's public key PK_B contained in the certificate), and calculates K_{AB}, and uses this to verify the cryptographic check using the function **f** and the key K_{AB}.

A then generates a message consisting of the following items concatenated together:

– a signature of the concatenation of K_A, K_B and B's identity B, signed using A's secret key SK_A corresponding to the public key PK_A
  Note that this signature consists of just the signature value - it need not include the quantities being signed, since B already has them.
– a cryptographic check, using the crypto function **f** with the key K_{AB}, of K_A, K_B and B's identity B.

A then sends this to B:

$$\begin{tikzcd}\mathbf{A} \arrow[r, "Sig_{SK_A}(B\|K_A\|K_B)\|\mathbf{f}_{K_{AB}}(B\|K_A\|K_B)"] & \mathbf{B}\end{tikzcd}$$

Fig. 3. Key Agreement Mechanism 7 of ISO 11770-3, Part 3

Stage 4. Finally, B verifies the signature (using A's public key PK_A transmitted in the certification in Stage 1), and verifies the cryptographic check using the function $\mathbf{f}$ and the key K_{BA}. If the two verifications succeed, B signifies his acceptance to A:

$$\begin{tikzcd}\mathbf{A} & \mathbf{B} \arrow[l, "\text{OK}"']\end{tikzcd}$$

Fig. 4. Key Agreement Mechanism 7 of ISO 11770-3, Part 4

5 Privacy-Preserving Protocol

The protocol based on ISO 11770-3, discussed in section 4.2 above, has the disadvantage that the card ("B" in the discussion) reveals his identity and certificate before he has verified the credentials of the reader "A". This could be viewed as a violation of the privacy of the card holder - the identity and certificate of the card B are revealed, not just to the reader A, but also to anyone eavesdropping on the communications between the reader and the card. The reader A might not be physically co-located with the card, but actually connected via a network of some sort. The DIN standards [6,7] for digital signature cards suffer from this potential privacy problem.

The protocol based on ISO 11770-3, discussed in section 4.2 above, also has the disadvantage that the number of bits transmitted in all the stages is somewhat larger than necessary. Minimizing the total number of bits transmitted is important, because some smart card readers will only communicate at 9600bps, and even ignoring the cost of computing the cryptographic operations, the time needed to transmit all the bits could become a serious problem in response time to the card holder.

To resolve both the privacy problems and to reduce the number of bits to be transmitted, Caernarvon bases its authentication on the SIGMA design [25] and the Internet Key Exchange (IKE) standard [13]. This protocol offers several significant advantages:

1. The session key parameters are exchanged very early in the protocol, even before the authentication has been completed. In this way, the information exchanged in the protocol, including the peers' identities can be protected from third-party eavesdropping.
2. A discloses its identity and credentials to B first; B reveals its identity and credentials only after verifying those of A. This prevents revealing the card holder's identity to a reader that cannot be authenticated or that cannot prove that it is authorized for a particular mandatory access classes. Therefore, the card's identity is protected not only against eavesdropping, but also against an active (man-in-the-middle) attacker. The reader's identity is not protected against an active attacker, but presumably the reader has fewer privacy concerns than the card holder. Note that in all authentication protocols, one party must reveal its identity first, and that party's privacy will always be subject to active attacks of this kind.
3. IKE transmits fewer bits in total. This will improve performance on slow readers.
4. The SIGMA and IKE protocols followed here have been rigorously analyzed and proven correct [5], which is a major benefit in any system planning to be evaluated at the highest levels of the Common Criteria. In particular, see [25] for more details on the cryptographic rationale of these protocols and the subtle cryptographic attacks they prevent.

This section contains a cryptographic description of the authentication protocol used by Caernarvon. Note that in contrast to the protocol described in section 4.2, the Caernarvon protocol starts as in unauthenticated Diffie-Hellman, and then authenticates A before B exposes his identity. The crucial technical difference between these protocols is that in the case of the Caernarvon protocol, A can authenticate itself to B without having to know B's identity, while in the ISO protocol of section 4.2, A authenticates to B by signing B's identity (thus requiring the knowledge of B's identity by A before A can authenticate to B)[1].

As discussed in section 4.1, A (the reader) and B (Caernarvon) share the public quantities p, q, and g. Section 6 will discuss why these public quantities must themselves be authenticated.

Stage 1. A chooses a random number a with $1 \leq a \leq q-1$, computes a key token $K_A = g^a \bmod p$, and transmits it to B.

$$\begin{tikzcd} \mathbf{A} \arrow[rrr, "K_A"] & & & \mathbf{B} \end{tikzcd}$$

Fig. 5. Authentication Stage 1: A sends a key token to B

[1] The Caernarvon protocol described here assumes that only one authentication is in progress between a reader and a smart card at any one time. This follows current practice in the smart card industry. If support for multiple authentication sessions were desired, the protocol would have to be modified to include a cryptographically bound session identifier.

Stage 2. B chooses a random number b with $1 \leq b \leq q-1$, computes a key token $K_B = g^b \bmod p$, and transmits it to A.

$$\begin{tikzcd}\mathbf{A} & & & \mathbf{B} \arrow[lll, "K_B"']\end{tikzcd}$$

Fig. 6. Authentication Stage 2: B sends a key token to A

At this point, neither A nor B has revealed his identity. However, they now can compute a mutual key K_{AB} as in section 4.1. Using the mutual key K_{AB}, they can derive additional keys K_{ENC}, for encrypting messages and K_{MAC}, for computing message authentication codes (MACs) as specified in section 7.2.

Stage 3. A now sends its certificate to B by encrypting it with K_{ENC}. A now computes E_{01} as shown below:

$$E_{01} = 3DES\_Encrypt_{K_{ENC}}(Cert(A))$$

A now transmits E_{01} together with its MAC to B.

$$\begin{tikzcd}\mathbf{A} \arrow[rrr, "E_{01}\|MAC_{K_{MAC}}(E_{01})"] & & & \mathbf{B}\end{tikzcd}$$

Fig. 7. Authentication Stage 3: A sends certificate to B

Stage 4. B responds with a challenge. From a strictly cryptographic perspective, stage 4 could be combined with stage 2, reducing the total number of message flows. However, this is a protocol for smart cards, and it must fit into the existing standard for smart card commands [15] and use the GET CHALLENGE and EXTERNAL AUTHENTICATE commands.

$$\begin{tikzcd}\mathbf{A} & & & \mathbf{B} \arrow[lll, "\text{RND.B}"']\end{tikzcd}$$

Fig. 8. Authentication Stage 4: B sends challenge to A

Stage 5. A now computes E_1 as shown below:

$$E_1 = 3DES\_Encrypt_{K_{ENC}}(A\|Sig_{SK_A}[K_A\|A\|RND.B\|K_B\|DH(g\|p\|q)])$$

A now transmits E_1 and a MAC of E_1 to B. The signature is a signature with message recovery, so all parameters in the signature can considered to be recoverable. The Diffie-Hellman key parameters are part of the signature in order to provide authenticity of the parameters. See section 6 for details.

$$\begin{tikzcd}\mathbf{A} \arrow[r, "E_1\|MAC_{K_{MAC}}(E_1)"] & \mathbf{B}\end{tikzcd}$$

Fig. 9. Authentication Stage 5: Authenticate A

At the conclusion of stage 5, B has authenticated A. It is at this point that the Caernarvon authentication protocol mandatory access control checks, as described below in section 7.5, steps 11 and 12. The mandatory access checks are done here, so that B can verify mandatory access rights, before revealing any privacy-sensitive information to A.

Stage 6: B now verifies the MAC, decrypts E_1, and verifies the signature using A's public key PK_A. B has now authenticated A and knows that K_A and K_B are fresh and authentic. However at this point, while B knows there is no man-in-the-middle because B checked the signature from A, A does not know who he is talking to, and hence is unsure if there may be a man-in-the-middle attack. B computes E_{02} (its encrypted certificate) and sends it to A.

$$E_{02} = 3DES\_Encrypt_{K_{ENC}}(Cert(B))$$

$$\begin{tikzcd}\mathbf{A} & \mathbf{B} \arrow[l, "E_{02}\|MAC_{K_{MAC}}(E_{02})"'] \end{tikzcd}$$

Fig. 10. Authentication Stage 6: B sends certificate to A

Stage 7. A sends a challenge to B. Just as for stage 4, strict cryptographic requirements could reduce the total number of message flows. However, once again, it is desirable to use the ISO standard [15] GET CHALLENGE and EXTERNAL AUTHENTICATE commands.

$$\begin{tikzcd}\mathbf{A} \arrow[r, "\text{RND.A}"] & \mathbf{B}\end{tikzcd}$$

Fig. 11. Authentication Stage 7: A sends challenge to B

Stage 8. B now computes E_2 as shown below:

$$E_2 = 3DES\_Encrypt_{K_{ENC}}(B\|Sig_{SK_B}[K_B\|B\|RND.A\|K_A])$$

The signature is a signature with message recovery, so all parameters in the signature can considered to be recoverable.

B now transmits E_2 and a MAC of the value E_2 to A.

$$\begin{tikzcd}\mathbf{A} & & & \mathbf{B} \arrow[lll, "E_2\|MAC_{K_{MAC}}(E_2)"']\end{tikzcd}$$

Fig. 12. Authentication Stage 8: authenticate B

A can now verify the MAC and decrypt E_2. Using the chain of certificates back to the root CA, A can verify the certificate from the IC manufacturer for B, which contains B's identify B and public key PK_B. Thus A knows, and can trust, B's public key PK_B. Hence A can now authenticate B by verification of the signature:

$$Sig_{SK_A}[K_A\|A\|RND.B\|K_B\|DH(g\|p\|q)]$$

6 The Authenticity of the Public DH Parameters

The proof given in [5] assumes that the public DH parameters are authentically known. This assumption does not necessarily hold for signature cards, as the public DH parameters may be retrieved from a file on the card prior to the device authentication.

As a consequence, the authenticity of these public key parameters is not given implicitly. Therefore the public key parameters need to be signed by the reader, which allows the card to verify that the reader used its correct parameters.

The following scenario demonstrates how this weakness could be used for an attack, given that the public DH parameters were not signed by the reader.

This attack scenario was contributed to the E-Sign committee by Andreas Wiemers [32]. Related attacks have been described by Lim and Lee [27] and Antipa, et. al. [1].

Step	**READER(A)**		**CARD (B)**
1		$\longleftarrow$	p, q, g
2	$K_A = g^a \bmod p$	$\longrightarrow$	
3		$\longleftarrow$	$K_B = g^b \bmod p$
4			$K_{AB} = {K_A}^b \bmod p$
5		$\longleftarrow$	RND.B
6	$\tilde{K}_{AB} = {K_B}^a \bmod \tilde{p}$		
7	$SIG_A(K_A\|RND.B\|K_B)$	$\longrightarrow$	
8	$RND.A$	$\longrightarrow$	
9		$\longleftarrow$	$SIG_B(K_B\|RND.A\|K_A)$

As soon as K_{AB} is available it is used to secure the communication between the reader and the card (after session key generation).

We suggest that an attacker changes the transmission such, that he replaces p by $\tilde{p}$. We obtain:

Step	READER(A)		CARD (B)
1		$\longleftarrow$	$\tilde{p}, q, g$
2	$\tilde{K}_A = g^a \bmod \tilde{p}$	$\longrightarrow$	
3		$\longleftarrow$	$K_B = g^b \bmod p$
4			$\tilde{K}_{AB} = \tilde{K}_A^b \bmod p$
5		$\longleftarrow$	RND.B
6	$\hat{K}_{AB} = {K_B}^a \bmod \tilde{p}$		

In general $\hat{K}_{AB}$ and $\tilde{K}_{AB}$ are different. However, if the attacker chooses $\tilde{p} = c \cdot p$ with a small number c (e.g. $c = 2$), an attacker observes the case $K_{AB} = 1 \bmod c$ with an expected probability of approximately $\frac{1}{c}$. In this case, it is easy to prove that the equation $\tilde{K}_{AB} = \hat{K}_{AB} = K_{AB}$ is valid exactly if $K_{AB} \equiv 1 \bmod c$.

If the reader and the card may communicate with secure messaging without error indication, the attacker obtains information about the negotiated key. For example, if c is 2, then the attacker would know that K_{AB} is odd, thus leaking one bit of the key. Choosing c larger makes it less probable to hit a valid negotiation between the reader and the card. However if this hit occurs, more bits are leaked from K_{AB}. As K_A and K_B are computed from a and b, which are chosen randomly, the attack cannot be used to accumulate information about the negotiated keys. However in the sense of provable security this is a valid example to demonstrate the general observation that the authenticity of public parameters is important for proper security.

Reason: For $\tilde{p} = c \cdot p$, it is always valid

$$\tilde{K}_{AB} = (g^a \bmod pc)^{r_B} \bmod p = g^{a \cdot b} \bmod p = K_{AB}$$

As $0 \leq K_{AB} \leq c \cdot p$ and $0 \leq K_{AB} < p$, then $\hat{K}_{AB} = K_{AB}$ is true if this equality is mod p and mod c. As the equality mod p is trivial and $\hat{K}_{AB} = {K_B}^a \bmod c \equiv 1$, the assumption is confirmed.

7 Caernarvon Authentication Flow

This section describes the full flow of the authentication protocol used by the Caernarvon kernel, including implementation details omitted from the cryptographic description above in section 5 and the authentication of mandatory access controls of section 3. It will also include smart-card specific details. Each stage from section 5 will be discussed in a separate sub-section. Each stage is implemented as one or more steps. While the numbering of sub-sections, stages, and steps is complex, it helps cross-referencing back to the pure cryptographic protocol of section 5 and ultimately to the implementation source code (not included in this paper.)

The reader is authenticated first; this is to provide the highest achievable protection of the ICC's (i.e. the card's) identity and data until the IFD (the reader) has been authenticated.

7.1 Stage 1 – Reader Sends Key Token to the Card

The five steps that make up stage one are explained below.

Step 1. Power the card
When the smart card is first inserted into the reader, it receives power and must respond following the Answer to Reset protocol specified in [14].

Step 2. Read Cryptographic Token Information
Prior to authentication the IFD might require parameters on how to proceed with, and where to find resources relevant for, the authentication. The IFD may issue commands to retrieve information for the authentication parameters in a format specified by ISO 7816-15 [16]. In particular, it may retrieve information on the authentication algorithm, the format of certificates, the presence of specific certificates, and key related information (key ID's, key length etc.)

Step 3. Read key exchange parameters
If the reader does not have the required key exchange information available, it may read them from the card. These may include public algorithm quantities which depend on the authentication algorithm. For instance, in a Diffie Hellman Key exchange scheme the public key quantities would be the public parameters p, q and g. The public key quantities reveal information about the authentication mechanism. As long as these quantities are used in many cards, the identity of a card is not revealed by the leakage of this information prior to authentication of the reader.

Step 4. Reader Computes its Key Token and Sends it to the Card
The reader chooses a random number a, and computes its key token $K_A = g^a \bmod p$ (see section 4). It then sends the key token to the card.

7.2 Stage 2 – Card Sends Key Token to the Reader

The two steps that make up stage two are explained below.

Step 5. Card Computes its Key Token and Sends it to the Reader
Caernarvon chooses a random number b, and computes its key token $K_B = g^b \bmod p$. It then sends the key token to the reader.

Step 6. Derive Keys for Use During the Remainder of Authentication
At this point, the reader and the card have completed simple unauthenticated Diffie-Hellman key agreement, as described in section 4.1. Neither side has been authenticated yet, but using the common secret K_{AB}, they now derive an encryption key K_{ENC} and a MACing key K_{MAC} that will

be used to protect the remainder of the authentication protocol from casual eavesdroppers. Both keys are 112-bit 3DES keys. Key derivation from the common secret is according to ANSI X9.63 [28] using HMAC [26] as the pseudo-random function. Note that there could still be a man-in-the-middle at this point in the protocol.

Both the reader and the card calculate:

$HASH1 = HMAC[K_{AB}](1)$
and
$HASH2 = HMAC[K_{AB}](HASH1\|2)$

112 bits are selected from $HASH1$ to produce K_{ENC}, and 112 bits are selected from $HASH2$ to produce K_{MAC}.

7.3 Stage 3 – Sending the Reader's Certificate

The two steps that make up stage three are explained below:

Step 7. Selection of the CA Public Verification Key
This step may be a single stage, or two stages, depending on whether the CA public key is held in the card. If the key is present in the card, then the reader need only select the key. If the key is not present in the card then the reader has to send the CA's public key with the appropriate certificate which must be verified by the card before it can be used. Care must be used in implementing this step, because a malicious reader could attempt a denial of service or a privacy attack against the card. If the card stores the CA's public key and certificate on a long-term basis, the reader could run the card out of memory, simply by sending lots of public keys and certificates. Since authentication has not yet been completed, the card cannot use memory quotas to protect itself, because it does not yet know whose memory quota to charge. The same approach could be used to violate privacy by using the public keys and certificates as the equivalent of web browser cookies. Fortunately, there is an easy fix. The card simply needs to not store these intermediate keys and certificates after either authentication has completed or the card has been reset. That simple fix eliminates the possibility of either denial of service or privacy attacks.

Step 8. Verify Reader's Certificate
The reader sends a computer verifiable (CV) certificate containing its public key. The IFD executes the VERIFY CERTIFICATE command, which causes Caernarvon to verify the certificate using the public key of the certification authority selected or verified in step 5. The public key of the reader cannot be trusted until it is confirmed by an appropriate certificate. The signature and the use of HMAC ensure that the operation is fresh and is cryptographically tied to the values of K_A and K_B.

7.4 Stage 4 – Card Sends a Challenge

Step 9. Get Challenge:

In order to prove its authenticity dynamically, the reader requests a challenge from the ICC. The challenge consists of a simple random number. Cryptologically, K_A and K_B have sufficient random quality. The additional request for $RND.B$ is used to initialize certain cryptographic checksum counters used in the secure messaging protocol of [14].

7.5 Stage 5 – Authenticate the Reader

Step 10. External authentication:

The reader computes a signature on the concatenation of K_A, A's identity, the challenge, K_B, and the public Diffie-Hellman parameters. Including the public parameters avoids the attack described in section 6.

After Step 10, the reader's public key can be trusted, and the card knows that there is no man-in-the-middle.

Step 11. Choose Secrecy Access Class

This step is to verify the secrecy access class to be requested, which may be less than the maximum secrecy access class of either the reader or the card. Note that this step is entirely optional; if it is omitted then a secrecy level of System Low will be used by default. This procedure is performed at this point in the authentication because:

- the session key has been agreed, so this negotiation can be performed using encrypted transmissions, to prevent leakage of information as to what access authorizations permitted to each participant. Thus this step cannot be performed earlier in the authentication.
- it is completed now rather than later for privacy reasons, that is so that the card does not reveal any personal information until it has been verified that the reader is indeed authorized to see such information. Revealing the card's identity before authenticating the reader provides for the possibility of unauthorized tracking the movements of the card holder.

Initially, the reader (IFD) must prove its secrecy access class to the card. The data field passed is an access class, signed and cryptographically tied to the public key of the IFD. Caernarvon verifies the IFD's access to the IFD (by verification of the appropriate signatures from the SA and that the public keys match), and then, in the response message, returns proof that the card is authorized to use the same access class. This proof consists of the card's access class signed by the SA and cryptographically tied to the card's public key.

There is no privacy problem revealing this information, because at this point it has been verified that the IFD holds the proper access classes. The IFD cannot complete its verification of the card's access classes until after step 16 below, because the IFD cannot be sure of the card's public key and the absence of a man-in-the-middle attack until step 16 has been completed. However, Caernarvon returns the access classes here because the card does know that it is safe to do so, and it avoids the necessity of implementing an additional step and APDU in the sequence after step 16.

At this point, the card now knows the maximal access class that can be used. The IFD must next specify exactly what access class (less than or equal to the maximal) that is to be used for this session. It is possible to reduce this access class, for example if the access class contains several categories, to select a subset of these categories, or to select a lower secrecy level than permitted by the verified access class. (In principle, the IFD could authenticate at precisely the access class that it wished to use. However, this would require that the secrecy authority for the access class to have signed all possible combinations of access classes less than or equal to the maximal access class. Storing all those signatures would be quite impractical, particularly when the IFD's memory capacity may be as limited as the smart card itself.) This selection of the subset could be done at a later stage in the authentication; however, it seems sensible to do it at the same time as the verification.

Step 12. Choose Integrity Access Class

This step is to verify the integrity access class to be requested. This procedure is performed at this point in the authentication for the same reasons as for the secrecy access class. Note that this step is entirely optional; if it is omitted then an integrity level of System Low will be used by default. Initially, the reader (IFD) must now prove its integrity access class to the card. The data field passed is signed and cryptographically tied to the public key of the IFD. This has now set a maximal access class (i.e. integrity level) that can be used. It is possible to reduce this access class, to select a lower integrity level than permitted by the verified access class. This selection of the subset could be done at a later stage in the authentication; however, it seems sensible to do it at the same time as the verification.

7.6 Stage 6 – Sending the Certificate to the Reader

Step 13. Read Card's Certificate

The reader now needs to read the certificate of the card. In the E-Sign protocol, this certificate may be signed by an external CA and there may be a need to chain back through one or more certificates. However, for a high-assurance system, simply chaining back through certificates to some public CA does not give the reader any assurance that the smart card is actually running the genuine high-assurance Caernarvon operating system. Instead, the public keys of each Caernarvon card are signed by the card manufacturer who is responsible for ensuring that the correct high-assurance ROM image has been burned into the card and that the card has been properly and securely initialized. No other authority can provide better assurance, because ultimately it is the card manufacturer who controls the ROM image.

Step 14. Read Card Manufacturer's Certificate

If the reader does not already have the card manufacturer's certificate, it reads in from the card.

Step 15. Key selection

Before the reader can process the INTERNAL AUTHENTICATE command, the card's private authentication key must be selected, using a Manage Security Environment command.

7.7 Stage 7 – Reader Requests Card to Send Challenge to Reader

Step 16. Internal Authentication

The reader issues an INTERNAL AUTHENTICATE command. The causes the card to send its challenge to the reader. The card then computes the signature over the challenge and the key token K_A, K_B and returns it to the reader encrypted with secure messaging.

7.8 Stage 8 – Reader Authenticates Card

Step 17. Verifying the signatures

The reader now verifies the response with the trusted key from the card's certificate and gets evidence that the signer (holder of the certificate) is identical with the entity that made the key negotiation.

After Step 17, both sides are authenticated and mandatory access classes have been selected and verified.

7.9 Post-authentication Phase

A successful authentication selects the desired access class, negotiated from that provided in the certificate of the IFD, for the current session. The session keys for the session are available to both parties, as described in section 7.2. All further communication is done under Secure Messaging either through protection (MAC) or encryption of the data being transmitted at the interface. The decision to send subsequent blocks in encrypted form depends on the selected secrecy access class. In general, secrecy access classes higher than certain specified values will require session encryption. The session ends when the card is RESET or powered off. There is no way to start a fresh authentication without RESETting the card.

8 Conclusion

We have shown how a high-assurance smart card operating system that supports download of mutually suspicious applications must enforce its own high security authentication protocol, rather than allowing the traditional smart card approach of allowing individual applications do perform their own authentication. Strong operating system-based authentication is essential so that the operating system can reliably protect one application from another, yet still permit controlled sharing of information. The protocol is designed to support mandatory access controls for both secrecy and for integrity. We have also shown potential privacy problems with existing smart card authentication protocols and how our new protocol helps to preserve the privacy of the smart card holder. However, the protocol is based on existing authentication standards that have been formally proven, and is being submitted for possible standardization.

Acknowledgements

We acknowledge the useful comments and suggestions of Jonathan Edwards, Rosario Gennaro, and Elaine Palmer of IBM, and Helmut Kurth of atsec information security. Andreas Wiemers of the BSI provided assistance in writing section 6. We also thank Philips Semiconductors for their continuing support. Finally, we thank the anonymous reviewers of the program committee and Yves Deswarte for their suggested improvements to the paper.

References

1. Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone. Validation of elliptic curve public keys. In Yvo G. Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567, pages 211–223, Miami, FL, 2003. Springer Verlag.
2. Application interface for smartcards used as secure signature creation devices: Part 1 - basic requirements. Technical Report CEN/ISSS WS/E-Sign Draft CWA Group K Version 1.05, Secretariat: DIN Deutsches Institut für Normung e.V, Berlin, 7 May 2003.
3. David E. Bell and Leonard J. LaPadula. Computer security model: Unified exposition and multics interpretation. Technical Report ESD–TR–75–306, The MITRE Corporation, Bedford, MA, USA, HQ Electronic Systems Division, Hanscom AFB, MA, USA, June 1975. `http://csrc.nist.gov/publications/history/bell76.pdf`.
4. Kenneth J. Biba. Integrity considerations for secure computer systems. Technical Report ESD–TR–76–372, The MITRE Corporation, Bedford, MA, USA, HQ Electronic Systems Division, Hanscom AFB, MA, USA, April 1977.
5. Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology - Crypto 2002*, volume 2045 of *Lecture Notes in Computer Science*, pages 143–161, Santa Barbara, CA, 2002. Springer-Verlag.
6. Chipcards with digital signature application/function according to SigG and SigV - part 1: Application interface. Technical Report DIN V66291-1, Secretariat: DIN Deutsches Institut für Normung e.V, Berlin, 15 December 1998.
7. Chipcards with digital signature application/function according to SigG and SigV - part 4: Basic security services. Technical Report DIN V66291-4, Secretariat: DIN Deutsches Institut für Normung e.V, Berlin, 17 October 2000.
8. Common security label (CSL). Technical Report MIL-STD-2045-48501, Joint Interoperability and Engineering Organization (JIEO), Fort Monmouth, NJ, 25 January 1995.
9. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

10. DOD 5200.28-STD, Department of Defense, Washington, DC, USA. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985. `http://csrc.nist.gov/publications/history/dod85.pdf`.
11. J. H. Ellis. The story of non-secret encryption. Technical report, Communications-Electronics Security Group (CESG), Cheltenham, UK, 1987. `http://www.cesg.gov.uk/publications/media/nsecret/ellis.pdf`.
12. Pierre Girard. Which security policy for multiapplication smart cards? In *Proceedings of the USENIX Workshop on Smartcard Technology*, pages 21–28, Chicago, IL, 1999. The USENIX Association.
13. D. Harkins and D. Carrel. The internet key exchange (IKE). Technical Report RFC2409, November 1998. `ftp://ftp.rfc-editor.org/in-notes/rfc2409.txt`.
14. Information technology - identification cards - integrated circuit(s) cards with contacts - part 3: Electronic signals and transmission protocols. Technical Report ISO/IEC 7816-3:1997(E), International Organization for Standardization, Genève, 18 September 1997.
15. Information technology - identification cards - integrated circuit(s) cards with contacts - part 4: Inter-industry commands for interchange. Technical Report ISO/IEC 7816-4, International Standards Organization, Genève, 1995.
16. Information technology - identification cards - integrated circuit(s) cards with contacts - part 15: Cryptographic information application. Technical Report ISO/IEC CD 7816-15, draft edition, International Organization for Standardization, Genève, 2001.
17. Information technology - security techniques – evaluation criteria for it security – parts 1, 2, and 3. Technical Report ISO/IEC 15408-1, -2, and -3, International Organization for Standardization, Genève, 1999.
18. Information technology - security techniques - entity authentication - part 3: Mechanisms using digital signature techniques. Technical Report ISO/IEC 9798-3, International Organization for Standardization, Genève, 15 October 1998.
19. Information technology - security techniques - key management - part 3: Mechanisms using asymetric techniques. Technical Report ISO/IEC 11770-3, International Organization for Standardization, Genève, 1 November 1999.
20. Paul A. Karger. The lattice security model in a public computing network. In *ACM 78: Proceedings 1978 Annual Conference*, volume 1, pages 453–459, Washington, DC, USA, 4–6 December 1978. Association for Computing Machinery.
21. Paul A. Karger. Multi-organizational mandatory access controls for commercial applications. Technical Report RC 21673 (97655), IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, 22 February 2000. `http://domino.watson.ibm.com/library/CyberDig.nsf/home`.
22. Paul A. Karger, Vernon R. Austel, and David C. Toll. A new mandatory security policy combining secrecy and integrity. Technical Report RC 21717 (97406), IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, 15 March 2000. `http://domino.watson.ibm.com/library/CyberDig.nsf/home`.
23. Paul A. Karger, Vernon R. Austel, and David C. Toll. Using a mandatory secrecy and integrity policy on smart cards and mobile devices. In *EUROSMART Security Conference*, pages 134–148, Marseilles, France, 13–15 June 2000.
24. Paul A. Karger, Vernon R. Austel, and David C. Toll. Using mandatory secrecy and integrity for business to business applications on mobile devices. In *Workshop on Innovations in Strong Access Control*, Naval Postgraduate School, Monterey, CA, 25-27 September 2000. published on CD-ROM. `http://www.acsac.org/sac-tac/wisac00/wed0830.karger.pdf`.

25. Hugo Krawczyk. SIGMA: the 'SIGn-and-MAc' approach to authenticated diffie-hellman and its use in the IKE protocols. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003 Proceesings*, volume 2729 of *Lecture Notes in Computer Science*, pages 399–424, Santa Barbara, CA, 17-21 August 2003. Springer–Verlag.
26. Hugo Krawczyk, M. Bellare, and Ran Canetti. HMAC: keyed-hashing for message authentication. Technical Report RFC-2104, February 1997. `http://www.faqs.org/ftp/rfc/rfc2104.txt`.
27. Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Burton S. Kaliski, editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263, Santa Barbara, CA, 1997. Springer Verlag.
28. Public key cryptography for the financial services industry, key agreement and key transport using elliptic curve cryptography. Technical Report X9.63-2001, American National Standards Institute (ANSI), 2001.
29. Gerhard Schellhorn, Wolfgang Reif, Axel Schairer, Paul Karger, Vernon Austel, and David Toll. Verification of a formal security model for multiapplicative smart cards. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *6th European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes in Computer Science*, pages 17–36, Toulouse, France, 2000. Springer-Verlag.
30. Standard security label for information transfer. Technical Report FIPS PUB 188, National Institute of Standards and Technology, Gaithersburg, MD, 6 September 1994.
31. Technical rationale behind CSC-STD-003-85: Computer security requirements – guidance for applying the department of defense trusted computer system evaluation criteria in specific environments. Technical Report CSC-STD-004-85, DoD Computer Security Center, Fort George G. Meade, MD, 25 June 1985.
32. Andreas Wiemers. Kommentare zu application interface for smart cards used as secure signature creation devices, part 1 - basic requirements version 0.14 28th february 2003 (in German). Technical report, Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany, 14 March 2003.
33. M. J. Williamson. Thoughts on cheaper non-secret encryption. Technical report, Communications-Electronics Security Group (CESG), Cheltenham, UK, 10 August 1976. `http://www.cesg.gov.uk/publications/media/nsecret/cheapnse.pdf`.

Hardware Encapsulation of Security Services

Adrian Baldwin and Simon Shiu

Hewlett Packard Labs, Bristol, UK
{adrian.baldwin,simon.shiu}@hp.com

Abstract. Hardware security modules can be used to encapsulate simple security services that bind security functions such as decryption with authorisation and authentication. Such hardware secured services provide a functional root of trust that can be placed within context of a wider IT solution hence enabling strong separations of control and duty. This paper describes an approach to using such hardware-encapsulated services to create virtual trust domains within a deployed solution. This trust domain is defined by the hardware protection regime, the service code and the policies under which it is managed. An example is given, showing how a TLS session within a web service environment can be protected and how this service can extend the secure communications into the backend systems.

1 Introduction

Security should be thought of as a holistic property that emerges from an IT solution and it is therefore useful to look at how particular security mechanisms affect the overall solution [1]. This paper examines how secure hardware can be used to deliver simple security solutions and the resultant effects on the wider solution. Services operating over a range of levels from protocol level services through to third party services are examined. A common theme throughout the paper is the way these services are encapsulated in secure hardware consequentially gaining a degree of separation from the IT infrastructure to which they are connected and which they help secure.

The next section describes in general terms how simple security services can be delivered on secure hardware and the resultant security properties. This is followed by the main example that looks at the security of a web service infrastructure communicating with clients via TLS or SSL. A service is demonstrated that firstly aims to encapsulate just the protocol and subsequently this is built upon to bring additional authorisation and audit properties to the wider web service systems.

The last discussion sections reiterate the general service delivery approach and the properties that emerge relating the general properties back to the example.

2 Secure Hardware and Service Delivery

A fundamental requirement in computer system security is ensuring a separation of duty between certain tasks. Secure operation systems take the approach of trying to distribute privileges over several operators [2] but these are hard to manage and hence not commonly used. Other approaches may involve using multiple machines in different locations for example, keeping the primary domain controller within a separate area from the administration of other enterprise machines. An extreme example of using separation of duty is to involve third parties [3][4] particularly for tasks relating to audit and non-repudiation. A good example here is the use of a time-stamp service [5] that provides an independent view on events specifying when things happened and that the events have not changed.

In effect, this separation of control is creating separate domains where certain tasks can be performed. For example, a secure web server platform will create one domain where web pages can be generated and a second where they can just be served. This paper describes an approach to using secure hardware to run simple security services and in doing so the creation of an alternative trust domain within the normal IT infrastructure. Running a security service within a secure hardware device with limited functional and management APIs allows policies [6] to be specified defining who can play (or delegate) roles – hence creating a separation of control.

2.1 Traditional Secure Hardware Usage

Traditionally hardware security modules (HSM) are used to create a small-scale separation between those who have access to cryptographic keys and the administrators of the systems that use them. The HSM will often be enabled by a security officer whilst the servers to which it is connected are run by system administrators.

Examining this model more closely however, there is an obvious weakness as shown in figure 1. The HSM keeps keys secure by protecting them in hardware but it offers a simple cryptographic interface [7] where the application uses the HSM simply to decrypt, encrypt and sign. An administrator configures the device such that it can only be used by the appropriate applications. However, a skilful administrator can misuse the device and whilst they are not able to obtain the keys they can gain access to use the keys. These keys could simply be used to access data to which the administrator should not have access. More seriously, they could be used to sign contracts, or initiate transactions say to transfer money in a bank's IT systems. Servers have complex software (how many millions of lines of code are in windows 2000) consequentially there are many bugs and these can lead to vulnerabilities. These vulnerabilities allow attackers to gain control of applications using the keys or even gain access to the cryptographic operations via gaining root control of the system. The latest Samba flaw (one out of many published recently) shows that it is a real possibility that the attackers can completely subvert a machine and gain such access.

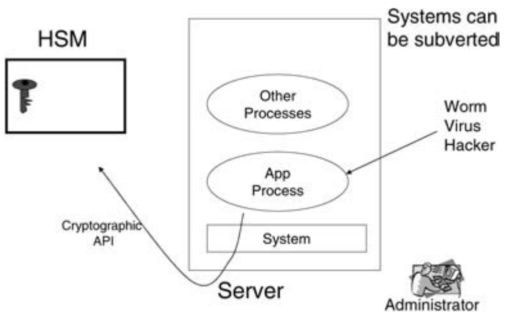


Fig. 1. Typical HSM usage.

The HSM does provide a high level of protection on what is often quite a general computing device; albeit with very little storage capacity. It will have some processing capability often with enhanced cryptographic acceleration. More importantly, it is a very safe environment. Once running the HSM protects against attacks [8] trying to gain access to the running code, or the keys stored in protective memory. Typically, there is active protection circuitry that, on detecting an attack, will destroy cryptographic material and blank memory so that no trace of the process state can be obtained.

2.2 Range of Usage Models

As well as the HSM usage model, there are other proposals for using cryptographic processors that encapsulate more code. For example, Smith et al [9] have developed a more powerful crypto-co-processor and have looked at pushing various applications, e.g. Kerberos [10] and file searching [11] inside the protected boundary. This approach of placing an application within the secure boundary has obvious security benefits but care needs to be taken in placing the too much of the application and underlying OS within the protected boundary where bugs can make the system vulnerable.

The LOCK program [12] took an alternative approach of building a secured reference monitor within a secure OS. The reference monitor is run within secure hardware such that it enforces the appropriate policies along with its crypto capacities. This can lead to considerable improvements in security but it does not naturally lend itself to securing complex (service based) distributed solutions within a heterogeneous environment.

Other hardware-based solutions such as TCPA [13] build trust up from the ground upwards. The trusted platform module within the TCPA specification offers three features allowing trust in the overall system to be built. Firstly, it gives each machine an identity; secondly, the TPM can be used to protect cryptographic keys. Lastly, the TPM acts as a root of measurements that can be used to ensure the correct versions of systems have booted and are running. The approach can help solidify the locked down platforms but they are still complex.

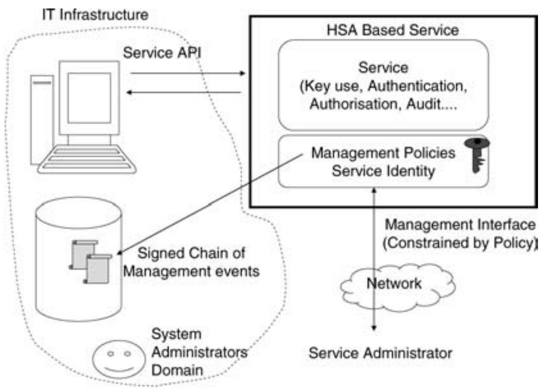


Fig. 2. The domain of control within an HSA Service.

This paper seeks to view a secure hardware device as a service delivery mechanism where a service is strongly encapsulated within the security device hence gaining its own security domain. These simple security services sit along side traditional web services helping to secure the overall solution. Of key importance, here, is to consider not only the functional stages of the service but the initialisation and hence the management policies [14] [15] [16] and therefore the way the service is run and administered. This paper shows how a strong separation of control and duty emerges from this approach.

2.3 HSA Approach

Simple security services can be created by taking the authentication, authorisation and audit elements of an application and combining these with the use of keys. For example, a simple service to decrypt data will now combine the authentication and authorisation with the decryption binding them together in the HSM and thereby stopping administrators or hackers getting in between these critical security processes. There are many applications ranging from secure storage [17] through controlling the signing of contracts when outsourcing infrastructure [18] to providing accountability services [19] [20] where simple security services encapsulating these elements can be extracted and made atomic by placing them into secure hardware.

It is not simply enough to place pieces of an application within the secure hardware; it is only when thought of as a separate service with its own identity and its own well defined operational and management structures that a virtual trust domain is created. The HSM device then becomes a method of delivering relatively small security services into a larger IT infrastructure but where the administration of such services are outside of the normal administrative boundaries. This turns the secure hardware into an appliance for delivering secure services (Figure 2) and therefore can be described as a hardware security appliance (HSA) rather than an HSM.

The HSA itself is given an identity as it is manufactured and has its firmware loaded. The identity consists of a RSA key pair and a certificate specifying the level of trust appropriate for the device and firmware. Rather than having a complex OS within the HSA it would contain a simple kernel and hardware abstraction layer along with a set of cryptographic, key management, communication and policy enforcement libraries on which services can be built. The service code within the HSA will thus be relatively small. The raw HSA then supports loading of a service.

A service running within an HSA exists with in its own right – it controls its own identity. As part of the configuration stage, the service is given an identity, a lifetime and configuration policies which include who can manage the service. This identity is given as an instance of a particular service and but is also signed with the HSA device identity. This allows a relying party to understand the trust characteristics of the service and the environment in which it runs. The HSA device ensures that a particular service runs, and a hash of the service code is included in the services identity certificate. The code could be published so that users of the service can check to see that the services' operational and management APIs are correctly enforced.

Consider a simple file decryption service that incorporates authentication and authorisation checks before decrypting a file. In itself, this binds together these operations but leaves open the question of who associated the authorisation checks with the file. This is specified at the initial configuration time, or more accurately, who specifies the authorisation policies is defined.

Now a set of service administrators control a service instead of the IT administrators. Further, aspects they control can be set by initial management policies. Lastly, to keep the HSA service administrators honest the service can produce secure audit log (by signing each event accompanied with the hash of a previous event) so it is clear who has made changes.

In the past, such secure hardware devices have tended to be relatively low power with optimisation for specialist cryptographic processing. It is believed that this is not a necessity and as processor power grows and memory remains, relatively cheap it is possible to build secure hardware devices with the power to execute authorisation and authentication policies as well as perform the cryptographic tasks.

3 Securing Web Services

The above discussion presents an approach of using security services running on secure hardware as a method of gaining wider system security through the creation of a service specific trust domain hence achieving separations of duties and concerns. This section describes how this approach can be applied at various levels within a web service solution and in doing so looking at a range of properties that can be derived.

A typical web service requiring security, for example, on-line banking or, e-commerce, will have a web server at the front acting as a presentation layer for

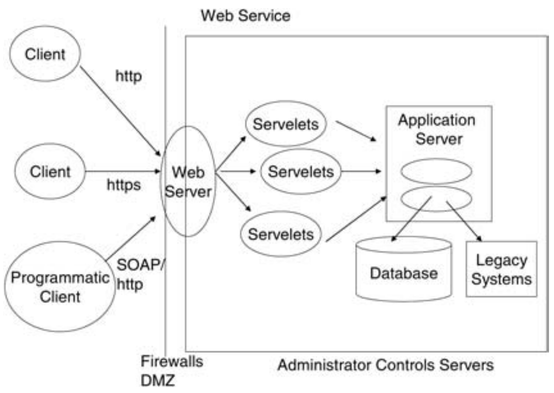


Fig. 3. A (java like) web service architecture.

backend processes that would perhaps be within an application server [21] or services within a .net infrastructure [22]. These services themselves will heavily rely on a database (Figure 3). Typically, the client will talk to the service using SSL or TLS, at least for the security sensitive parts of their interaction. The example in this section examines the client interactions using TLS and shows how an HSA service manages the running of this protocol and helps in extending its security properties further into the backend systems.

The service provider will have a number of security problems that range from simply keeping their TLS keys (that is their public identity) secure through to securing their backend and ensuring transactions cannot be inserted or information lost. Commonly, security designs concentrate on securing the network architecture, ensuring that there is a DMZ between the customer, hackers and the backend processes. Less concern is placed on security in the backend, here the administrators often retain much control and could easily create or change transactions. Gaining a degree of independence from the administrator is critical for a number of reasons including the heavy reliance on contractors (many attacks are from insiders) and the dangers that exist if the administrative accounts become subverted (say by a virus attack). Independence of control within various elements of the web service is critical for security and limiting the effect of attacks. Similar properties are also a tremendous advantage when outsourcing IT functions.

The rest of this section follows through this web service example, firstly simply looking at the security of the TLS protocol with the client – addressing questions pertinent to control of the keys on the server side. Using an HSA based service, elements of the TLS protocol are bound together so that they cannot be influenced by the normal IT staff. Now keys will only be used for this protocol and since the service understands and manages the sequence of messages, it becomes very hard even to misuse the elements from which the protocol is composed.

Security services at this protocol level see very little of the application semantics thereby making it hard for them to have a wider security influence of the overall system. Therefore, ways of extending these protocol level services to gain more influence in the wider backend systems are demonstrated. The extreme of this is the use of third party control as described in section 3.3.

3.1 Protocol Level Services

One of the key security features relied upon in such a web service infrastructure is the TLS or SSL session held with the client or at least it is what the client sees as security. It confirms to whom the client is talking and ensures their interactions are kept confidential and have integrity. The security of the TLS session is therefore significant to the security of transactions with the web server; in turn, the security of the TLS session is largely dependant on the security of the keys involved in the session. This includes protecting the private key used in the TLS handshake as well as ensuring the master key is protected (particularly where session re-establishment is enabled).

The TLS protocol (figure 4) [23][24] starts with a handshake initiated by the client leading to the client defining a pre-master secret and sending it encrypted with the servers public key (alternative options may involve Diffie Hellman exchanges). The server decrypts this pre-master secret using it to generate a master secret that in turn is used to generate the six values for HMACs, encryption and IVs in both directions. These secrets then form the basis of the secure channels both encrypting and ensuring the integrity of the data. The master secret and the cipher state are cached along with a session identity value – this value can be used in re-establishing or duplicating a session on the client's request.

TLS and SSL accelerators (or HSMs) are available to offload some or most of the cryptographic functions – particularly the expensive asymmetric operations. Secure processors can also be used to ensure that the server's private key is not disclosed. The discussion here is based on the type and level of the services exposed by such devices. Devices with PKCS11 [7] interfaces expose some of the basic crypto functions necessary for SSL and TLS for example to generate the master secret and the cipher state. Whist offloading the crypto processing and protecting the encryption keys this type of interface does not strongly bind all the protocol operations together around the session id.

The HSA approach (at least for the server side) would be to have a slightly higher level API that encapsulates the protocol level interactions rather than the cryptographic functions. The previous section talks about combining authentication, authorisation and audit operations along with the use of cryptographic keys in forming simple security services; however, when looking simply at the protocol level it is hard to integrate in these elements. This discussion examines raising the API level to represent and bind together the various stages in the protocol. In itself, this adds further control as to how the system operates the TLS protocol and therefore adds some additional protection when processes are subverted. More importantly, it forms the basis for the next couple of sections that add additional control elements into the secure service.

TLS Services. Instead of having the simple cryptographic functions in the secure hardware here, consideration is given to running much more of the protocol within the secure hardware. In doing so there is now a service (or set of services) which more fully controls the TLS interaction thereby creating an alternative trust domain controlling the protocol. The simplest option is for the TLS handshake to be run in the secure hardware this removes gaps between pieces of the handshake particularly around computing a digest of the handshake message required in the final finish stage.

The initial client hello message contains information concerning the protocol versions and cipher suites that are understood as well as the client's public random value and proposed session id. This initial message would be forwarded to the TLS service within the secure hardware. Typically, this would lead to the initiation of a new session with a session context within the bounds of the secure hardware with the reply being an acceptable cipher suite a new session ID, the server's random number and where required a certificate request. An alternative reply where the client specified a session ID would enable the session to be re-established.

It is worth noting here that the secure hardware is building up session state in its internal memory. This session state will include partially computed hashes of messages, random values and eventually the cipher state. The secure hardware therefore needs sufficient memory to hold this state (for each session being managed). The session state is not permanent; a loss of power will lead to a loss in the state and all sessions will be dropped – the same as if power is dropped from the server. The session state information and keys will not leak.

Assuming it is a new session being established the server will get responses back including the client key exchange, and if requested the client certificate and certificate verify message. The client key exchanges contain the pre-master secret encrypted for the server's public key. The TLS service holds the private key and therefore gets the pre-master secret that is used to generate the master secret. Where the TLS service has been set up to request a client authentication, it would also expect to get the certificate verification message. The clients message set ends with the finish message which is encrypted with the new cipher spec (the key set generated from the master secret). The TLS service can generate its own finish message based on the cipher spec and the digest of all the messages that it has computed.

The TLS service now has the appropriate cipher spec that it can use in decrypting and validating client messages and encrypting and generating the HMAC for the outgoing messages. In this way, the TLS service would receive blocks of data ready for encryption and return the encrypted and MACed message to the server for forwarding to the client. The client's messages will initially be passed through the TLS service for validation and decryption with the data being returned to the server. Each request should be associated by the server with a connection and session Id so that the TLS service can associate the messages with the cipher context.

TLS specifies data compression algorithms as well as encryption algorithms and a simple form of this TLS service would take the compressed data and pass back compressed data. This however, leaves little possibility of extending the

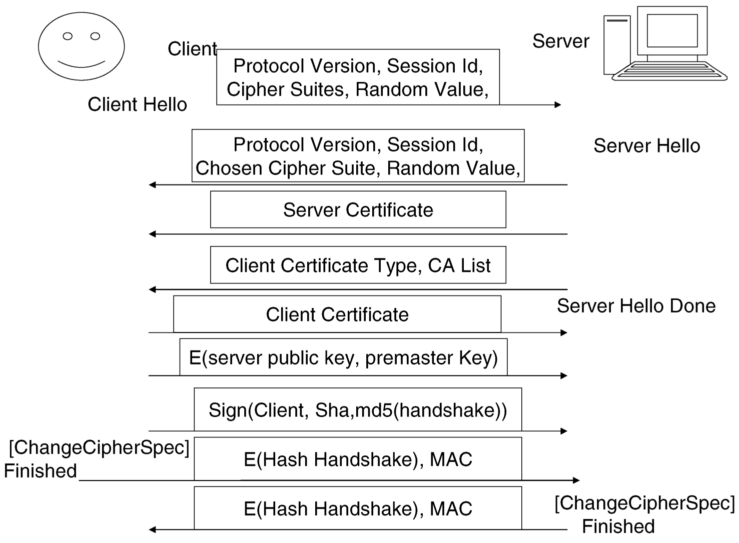

Client
Server
Client Hello
Protocol Version, Session Id, Cipher Suites, Random Value,
Protocol Version, Session Id, Chosen Cipher Suite, Random Value,
Server Hello
Server Certificate
Client Certificate Type, CA List
Server Hello Done
Client Certificate
E(server public key, premaster Key)
Sign(Client, Sha,md5(handshake))
[ChangeCipherSpec] Finished
E(Hash Handshake), MAC
E(Hash Handshake), MAC
[ChangeCipherSpec] Finished

Fig. 4. The TLS Handshake.

service with some form of application syntax or semantics as described later. To do this the service needs some view on the syntax of the messages and this is hard when they are compressed. Therefore, either de-compression should also be carried out in the TLS service or compression should not be used.

Control. Having described the basic protocol level service it is worth reflecting the level of control that can be gained by wrapping the TLS protocol handling as a separately identified service. The basic service takes control of the server's private keys and hence the identity – this can be generated in the service and its use can be constrained as part of this generation and service initialisation protocol. However, the protocol does not allow for many constraints to be specified. Policies can be used to ensure that only acceptable cipher suites are used and to specify if client authentication is required and the acceptable CA trust list. The service itself only runs TLS (or perhaps SSL) therefore; a subverted server will not be able to use the key for any other protocols. They could not use the key for arbitrary tasks such as signing contracts, or certificates – although certificate extensions aim to block alternative uses, validation algorithms are often complex and not fully implemented. Equally, a subverted server will not be able to control the choice of cipher suites (e.g. to drop encryption) or drop the need for client authentication.

3.2 Application Security

Controlling communications at the protocol level can be helpful but when considering the wider web service including the backend it is clear that the security

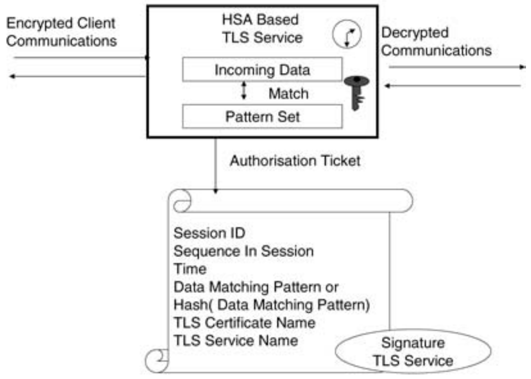


Fig. 5. The TLS Service as an Authorisation Ticket Generator.

ends at the web server boundary. New security methods can then be used in the backend but clearly there is a potential disconnect and reliance on the security of the web server platform. This section is concerned with how the simple protocol service can be expanded – whilst keeping it as a relatively small and simple service. One key question being addressed here is how a simple service encapsulated in secure hardware can have a wider influence on the overall solution.

A typical three-tier web service uses SSL or TLS sessions to provide secure communications with the client but then there is this disconnect with the backend. Take the example of internet banking, a user can connect, authenticate themselves with their password and authorise transactions. Communications will be secure to the level of the web server, past this there is a reliance on the security of the backend systems. This reliance is based both on the trust of the administrators (i.e. that they will not initiate transactions; use the web server to subvert user passwords etc) as well as a reliance that the service has not been subverted. In the example, how would the backend distinguish between a transfer initiated through an administrator (or subverted web server) as opposed to one initiated correctly by a client.

Extending the TLS Service. This section is concerned with extending the influence of the TLS session that the user has into the backend to achieve a greater level of security in the overall system. Towards this goal, the basic service needs to be extended so that it has some view onto the underlying applications and the messaging structure between the client and server. However, it is not desirable to try to push application semantics into what is intended to be a relatively simple security service.

The TLS service can be extended to recognise simple syntactic patterns within the message structure and then to issue tickets containing the information associated with such patterns or in the case of passwords a simple function on these patterns. Such a ticket would take the form shown in figure 5 containing information concerning the TLS session and the information matching the pattern. The ticket will also include a time and sequence number added by the TLS service, and where client authentication is used, a reference of the certificate is included. Tickets relating to a particular session will be numbered sequentially so that sequences of commands can be recognised – several may be needed to authorise a transaction. Following the banking example, a ticket could be generated when the TLS service sees a particular web form (with a very well defined structure) that relates to the transfer of money to other accounts.

The ticket is then signed which in a busy server environment can present a considerable cost therefore; a ticket batching mechanism can be used where the signature is on a ticket batch with the batch containing the hash of individual tickets. Such ticket batches could then be issued say 4 times a second. This would introduce delays into the system but they are probably small and acceptable when compared to user's perceptions and the normal network delays.

An assumption in the above discussion is that a web interaction is represented by a single TLS session that is re-initiated for each http get and send. Whilst possible with TLS, this was not possible with early versions of SSL and the browser may not use this functionality. Having tickets showing that individual requests occurred can be important but it will often be necessary to link sequences of tickets for example the authentication with the transaction. Such sessions can be created across the http requests over different SSL sessions using cookies that should be created and validated by the TLS service (within the secure hardware).

The TLS service is now configured to generate certain tickets that are triggered from specified patterns. For example, a pattern may be set up such that once an http send response containing a particular web form is seen a ticket is generated. In general, the patterns are regular expressions working on the clients messages and once matched the data (or the hash of the data) is placed into a ticket. This ticket is passed to the host server and can then be forwarded to the web server so that it can accompany the normal backend requests.

Two particular aspects of such tickets are expanded here:

Password Security. Web services often wish to authenticate users (e.g. an internet bank) but the complexity of PKI deployments means that they do not use two way TLS; instead, they have a password database. The user will login providing their user name and password by filling out a particular HTML form at the start of their interactions. Typically, this password is protected during transmission by the TLS session and will be processed on receipt by the backend server, hopefully, using standard password techniques, but possibly by just comparing it with the password value in the database. In either case, the password will be in the clear on the web server and therefore becomes vulnerable.

The inclusion of a regular expression parsing mechanism within the TLS service can be used to specify the template that results from the login form and includes the user login name and their password. Once recognised within the TLS service the password field can be removed from the normal data stream and substituted with a function (e.g. salted hash, or encryption) of the password. Additionally, a ticket can be generated by the service. In this case, the ticket should include the mangled password along with the user name. This data can be validated against the password database and passwords or other authentication information is not exposed.

Transaction Security. The generation of a ticket specifying the password is just one example of the types of tickets that can be generated. Using regular expression matching within the service, any part of a web form or even a SOAP request can be matched and a ticket representing this part of the session can be created.

The ticket, as described earlier, can then be presented to the backend showing that the transaction has been authorised as part of the clients TLS session. The main issue is then associating the particular ticket with the user log in and the rest of the session. This is where sequenced tickets can be useful where the backend can get a login and password ticket from the same session as the authorisation ticket.

Consider the on-line banking example. An online banking service may allow a user to transfer money to other accounts. For such an action, they may want proof that the user enacted the transaction both as an audit record and as part of the authorisation of the transaction. A combination of two linked tickets: the first containing the user name and their mangled password; the second describing the transfer amount, source and destination accounts; provide evidence to the bank. These tickets could even form the authorisation in other systems run elsewhere in the bank – particularly when the IT for the on-line service is outsourced.

The tickets provide two factors for the backend process – firstly, they provide a method of linking the TLS session with the transaction that can be verified before the transaction is enacted. Secondly, the sequences of tickets provide an audit trail that is directly linked to the user's interactions. This linkage is based on the TLS service being encapsulated and thereby having a trust domain that is independent from the normal IT infrastructure.

Extended Application Security. The examples have shown how authorisation tickets can be generated by a relatively simple extension to a service operating the protocol. These tickets have the effect of strengthening the end-to-end security within the backend in that they convert the client's requests into secured authorisation tickets. This prevents the web service administrator from inserting transactions at the web server level. The backend systems (or web service wrappers) can then be modified to check that a current ticket is presented (and is not being replayed) before processing the result. The administrator can still subvert the backend systems although traditional security mechanisms should still be deployed. Where the backed systems themselves are built using HSA based services, these services could be setup to consume the authorisation tickets and the service based roots of trust start to join up.

TLS Service Domain. It is interesting to examine the domain of the TLS service in more detail. There are three distinct stages: loading of the TLS service, initialisation of the TLS keys and its normal operation. The first two stages define the TLS service domain.

In first stage, the TLS service is loaded into the secure hardware during a service configuration stage. The software is loaded outside of the control of the normal IT infrastructure and as part of the loading process; the service generates a RSA key pair that is then certified. The private key is retained within the service – it enables data to be sent securely to the TLS service and it is used to sign tickets.

Part of the certification process should also involve the specification of the service controller. The service will offer a limited management API and the service controller (or those specified by the controller) will be the only people whose requests will be enacted.

The second stage happens once the service is running and positioned within the IT infrastructure, the service controller can get the service to create a RSA key pair for the TLS identity. This will result in a certificate request being generated and certified in the usual manner. Along with the certificate, the controller can set a bunch of policies concerning both protocol level controls and ticket generation patterns. The service controller can later change these configurations.

The service controller would probably be identified using their private key and associated certificate. On configuring or changing the configuration of the TLS service, they would first contact the service to initiate an update protocol. The TLS service responds with a nonce that the controller combines with the request before signing this structure. The TLS service then receives this configuration information, checks it corresponds to a recently generated configuration update nonce and matches the initially configured service policies as a valid configuration update. The controller need, therefore, not be local to the system – they could be elsewhere in the company (e.g. in the audit department) or they could even be within a third party.

The important message from this discussion is that a service encapsulated within secure hardware and offering a limited API can be used to create an alternative trust domain. During the configuration stage, the service provider can define who controls the service and the policies under which it is run. This allows the creation of strong separations of duty and control.

3.3 Service as a Third Party

The above discussion introduces the idea that once the service is encapsulated it can be remotely controlled and this control can even be from a third party. Third party services [3] can be very important acting as independent witnesses in a variety of transactions. The use of tickets as an evidence trail demonstrating that a client really made a request can be important but becomes far more compelling if this ticket is generated outside the control of either the client or server.

As well as generating tickets that match particular patterns, end of session tickets can also be generated – either as a session end is detected or as a result of a timeout within the session cache. This session receipt would then contain a hash of all traffic, or hashes of all client traffic and all server traffic. The receipt would be signed along with a time (from an internal HSA clock) by the TLS service. It can now be given to the client and server and if the service is initiated as a third party (of good name); it is likely to be trusted by both.

Such a receipt would firstly record that a transaction took place and would contain information that links into the individual tickets. Secondly, either party could save the complete text of the interaction or enough to recreate it – this would then provide proof of what was said. A slight modification can be made in that the receipt could also include removed or separate sections identifying the bytes removed and the hash of these bytes. This ensures that passwords can be removed and that they do not remain in the audit log.

This now becomes equivalent to a service provider, say a bank, recording a transaction with a call centre. It opens up the possibility of the user of an internet bank to prove that they did authorise a transaction or change payment instructions. As opposed to having to rely on good IT practices as currently happens today.

Having described the TLS service as a third party service, which is completely under the control of the third party may not be the most desirable separation of duty. The TLS service provider needs to be in overall control but they are not necessarily the ones who should be controlling the ticket generation policies. For example, the TLS service provider would provide the service context for a bank to operate a TLS session with the necessary constraints so they can issue tickets and receipts. The Bank would have control of the RSA keys and the ability to set policies over what is ticketed for sessions involving these keys. A set of service level policies configured by the overall service controller can be used to specify how control is delegated.

4 HSA Solutions

This paper proposes the use of secure hardware based services as a mechanism for delivering security into solutions. Because the hardware can be trusted, the services become roots of trust in the wider solution. The example given above starts by rooting trust in the way a secure protocol is run but the scope is then widened allowing certified information to be disseminated throughout backend IT systems. In doing so, the service is providing trust in authorisation tickets even to the extent that they can be used to form a strong audit or evidence trail for dispute resolution.

The example given is one of a set of such services that can be used to have a wider effect on the overall system. These range from simple time-stamping services that tightly integrate into an IT infrastructure through secure storage solutions to accountability services say for electronic patient records [17], [19]. At the heart of all these solutions are services that combine elements of authorisation, authentication and audit along with key usage in a service run in secure hardware so that the service becomes atomic from the point of view of the rest of the infrastructure.

One of the key features of such services is the way an alternative trust domain is created and that this trust domain remains strong even when deployed. The discussion of the TLS service in section 3.2.5 illustrated how such domains are created:

- A service is placed on the secure hardware device.
- The service configuration is initiated. Here a range of service policies are set. These define a service controller, tasks they can perform and perhaps the lifetime of the service.
- The service instance with its configuration is then given a strong identity.

The service code along with configuration policies defines what can be done with the service. These policies may allow a service controller to delegate certain rights but the domain of control is very much centred on that service controller and the limited functionality they are provided by the service API.

5 Conclusion

This paper has described the use of an instance of an HSA based service that helps in securely managing a TLS session and more importantly extends the control of the session out into the backend – even providing mechanism to share receipts with the clients.

This example demonstrates how simple security services running within secure hardware can be used to deliver elements of control and separation of duty within a typical IT installation. The paper has demonstrated how this service delivery mechanism allows a service to create its own virtual trust domain outside of the control of administrators or subverted IT systems.

References

1. Monahan, B.: From security protocols to systems security: A case for systems security modeling tools. In: Proceedings of the 11th Cambridge International Workshop on Security Protocols. LNCS, Springer-Verlag (2003)
2. Dalton, C.I., Griffin, J.: Applying military grade security to the internet. In: Joint European Networking Conference, Edinburgh. (1997)
3. Baldwin, A., Beres, Y., Casassa Mont, M., Shiu, S.: Trust services: Reducing risk in e-commerce. In: Proceedings of the International Conference on E-Commerce Research. (2001)
4. Baldwin, A., Shiu, S., Casassa Mont, M.: Trust services: A framework for service based solutions. In: 26th IEEE Computer Software and Applications Conference (COMPSAC), Oxford UK (2002)
5. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology **3** (1991) 99–111
6. Goh, C., Baldwin, A.: Towards a more complete model of roles. In: 3rd ACM Workshop on Role-Based Access. (1998) 55–61
7. RSA: PKCS#11 v2.11 cryptographic token interface standard (2001)

8. FIPS: Security requirements for cryptographic modules. Fips 140-2 (2001)
9. Smith, S., Palmer, E., Weingart, S.: Using a high performance programmable secure coprocessor. In: The second international conference on financial cryptography. LNCS, Springer-Verlag (1998)
10. Itoi, N.: Secure coprocessor integration with Kerberos V5. In: Usenix Security Symposium. (2000) 113–128
11. Smith, S., Safford, D.: Practical private information retrieval with secure coprocessors. Technical report, IBM Research T.J. Watson Research Centre (2000)
12. Smith, R.: Cost profile of a highly assured, secure operating system. ACM Transactions on Information Systems Security (2000)
13. Pearson, S., ed.: Trusted Computing Platforms: TCPA technology in context. HP Books, Prentice Hall (2002)
14. Sloman, M., Lupu, E.: Policies for distributed systems and networks. In: Proceedings of the 2nd International Policy Workshop. Volume 1995 of LNCS., Springer Verlag (2001)
15. Grandison, T., Sloman, M.: Sultan - a language of trust specification and analysis. In: Proceedings of the 8th Workshop of the HP Openview University association. (2001)
16. Casassa Mont, M., Baldwin, A., Goh, C.: Power prototype: Towards integrated policy based management. In J. Hong, R.W., ed.: Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS). (2000) 789–802
17. Baldwin, A., Shiu, S.: Encryption and key management in a SAN. In: IEEE Security In Storage Workshop, SISW02. (2002)
18. Baldwin, A., Shiu, S.: Hardware security appliances for trust. In: First International Conference on Trust Management. LNCS, Springer Verlag (2003)
19. Ferreira, A., Shiu, S., Baldwin, A.: Towards accountability for electronic patient records. In: The 16th IEEE Symposium on Computer-Based Medical Systems. (2003)
20. Baldwin, A., Shiu, S.: Enabling shared audit data. In: Proceedings of the 6th Information Security Conference. LNCS, Springer Verlag (2003)
21. Kurniawan, B.: Java for the Web with Servlets, JSP and EJB. A developer's guide to J2EE solutions. Que (2002)
22. Richter, J.: Applied Microsoft .Net Framework Programming. Microsoft Press (2002)
23. Dierks, T., Allen, C.: The TLS protocol version 1.0. IETF RFC 2246 (1999)
24. Freier, A., Karlton, P., Kocher, P.: The SSL protocol version 3.0. IETF Internet Draft (1996)

A Formal Security Model of the Infineon SLE 88 Smart Card Memory Management

David von Oheimb[1], Georg Walter[2], and Volkmar Lotz[1]

[1] Siemens AG, Corporate Technology, D-81730 Munich
{David.von.Oheimb,Volkmar.Lotz}@siemens.com
[2] Infineon AG, Secure Mobile Solutions, D-81609 Munich
Georg.Walter@infineon.com

Abstract. The Infineon SLE 88 is a smart card processor that offers strong protection mechanisms. One of them is a memory management system, typically used for sandboxing application programs dynamically loaded on the chip. High-level (EAL5+) evaluation of the chip requires a formal security model.
We formally model the memory management system as an Interacting State Machine and prove, using Isabelle/HOL, that the associated security requirements are met. We demonstrate that our approach enables an adequate level of abstraction, which results in an efficient analysis, and points out potential pitfalls like non-injective address translation.

Keywords: Security, formal analysis, smart cards, memory management, Interacting State Machines, Isabelle/HOL.

1 Introduction

Since smart cards are becoming widely spread and are typically used for security-critical applications, smart card vendors face the demand for validating the security functionality of their cards wrt. adequacy and correctness. Third-party evaluation and certification is accepted as the appropriate approach, making it quite an active field. Certification of smart card processor products according to the Common Criteria [CC99] typically refers to the Smartcard IC Platform Protection Profile [AHIP01] and its augmentations like [AHIP02]. Based on these documents, the security target [WN03] for the Infineon SLE 88 smart card chip demands assurance level EAL5 to be achieved, in particular requiring formal reasoning on the requirements level, viz. a formal security model.

Infineon could make use of an extension of the established LKW model [LKW99] which already covers most aspects of security. Yet the SLE 88 offers a new security feature that requires special attention: a sophisticated memory management. For its evaluation we have developed a formal model which we describe in detail in the present article. The upcoming field of multi application smart cards motivate protection of applications from each other. The model shows that this can be effectively achieved with classical hardware based separation of memory areas. This feature may be used in particular within Java

Virtual Machine implementations, yielding major progress in the area of dynamically loadable applications for smart cards.

The memory management security model is given in terms of Interacting State Machines (ISMs) introduced in [Ohe02,OL03]. ISMs are state-transition automata that communicate asynchronously on multiple input and output ports and thus can be seen as high-level Input/Output Automata [LT89]. They have turned out to be appropriate for the task of security modeling, for instance the full formalization of the LKW model with the Isabelle theorem prover in [OL02].

Most of related work on high EAL evaluation for smart cards, done e.g. at Trusted Logic and Philips, is unpublished. There are publications dealing with the Java Card runtime system [MT00] and with smart card operating systems in general [SRS$^+$00]. Note that these focus on software, while we focus on hardware.

2 SLE 88 Memory Management

In this section we introduce the virtual memory system of the SLE 88 with their associated security objectives and protection mechanisms.

2.1 Memory Organization

The physical memory of the SLE88 family is handled via 22 bit *physical effective addresses (PEAs)*. Virtual memory is addressed via 32 bit *virtual effective addresses (VEAs)*. The atomic units of the translation from virtual to physical addresses are *pages* of 64 bytes, which results in 6 bit wide *displacements*, i.e., address offsets. Peripheral hardware is memory mapped and thus can be accessed — and protected — in the same way as ordinary memory cells.

Typically, there are several independent software modules of different origin. Therefore, virtual memory is logically divided into 256 *packages* of equal size, such that the *package address (PAD)* makes up the upper 8 bits of the VEA. Packages 0 to 2 are *privileged* because they control security-critical entities. Package 0 contains the *security layer (SL)*, package 1 contains the *platform support layer (PSL)*, also known as *hardware abstraction layer (HAL)*, and package 2 contains the *operating system (OS)*. Of the remaining *regular* packages, those with numbers 3 to 15 are reserved, while those with numbers 16 to 255 are available for (third-party) application software to be uploaded on demand.

2.2 Security Requirements

The security objective relevant here is O.Mem-Access: "Area based Memory Access Control", defined in [WN03, §4.1]:

The TOE must provide the Smartcard Embedded Software with the capability to define restricted access memory areas. The TOE must then enforce the partitioning of such memory areas so that access of software to memory areas is controlled as required, for example, in a multi-application environment.

This means in particular that inter-package access to code and data should be restricted and that the corresponding protection attributes should be controlled by (specially protected) privileged packages only. Detail on the associated *Security Functional Requirements (SFRs)* may be found in [WN03, §5.1.1.2].

2.3 Protection Mechanisms

Virtual memory is associated with *effective access rights (EARs)* determining the read, write, and execute access of packages. Their granularity is 256 bytes, corresponding to the lower 8 bits of the VEAs. Moreover, each physical *page block* of 16 bytes, corresponding to the lower 4 bits of the PEA, is associated with additional security attributes referred to as *block protection field (BPF)*. The only information we will need in the model is a predicate called *PASL* specifying whether a page block should be accessible by SL only.

An EAR is given by a two-letter code where each of the letters may be `W`, `R`, `X`, or `-`, which specify read/write access to data, read-only access to data, executing access to code, and no access, respectively. The first letter refers to access within a package, while the second letter refers to access of one package (the source) to some other package (the target). The only allowed combinations are `WW`, `WR`, `RR`, `W-`, `R-`, and `X-`. Note that the EAR gives an implicit classification of memory sections as code or data. Code can be marked only with `X-`, which indicates that inter-package code fetch is generally prohibited. Regardless of the EAR, privileged packages have free data access to all other packages except SL.

Apart from the restrictions on (linear) code fetch, inter-package control transfer is allowed only if the target holds a special PORT instruction sequence that defines the set of packages allowed to enter.

3 Formalism and Tools

For modeling (and partially verifying) the SLE 88 memory management, we take the ISM [OL02] approach. This means that we formally define and analyze its security model as an Interacting State Machine (ISM) [Ohe02,OL03] within the theorem prover Isabelle/HOL [Pau94].

Interacting State Machines (ISMs) are automata whose state transitions may involve multiple input and output simultaneously on any number of ports. For the SLE 88 security model we need only a single basic ISM with one input port accepting machine (micro-)instructions and one output port issuing the reaction of the memory management system. What we do make use of is single state transitions as well as transition sequences which result from system runs.

Isabelle is a generic theorem prover that has been instantiated to many logics, in particular the very practical *Higher-Order Logic (HOL)*. HOL [PNW$^+$] is a predicate logic based on the simply-typed λ-calculus and thus in a sense combines logical and functional programming. Proofs are conducted primarily in an interactive fashion assisted by automatic and semi-automatic tactics.

4 System Model

In order to provide an comprehensive instructive presentation of the formal model, we reproduce all the definitions, lemmas and theorems (essentially leaving out proof scripts and a few other parts needed for technical reasons only), augmented with comments, just as they appear in the Isabelle theory sources[1]. By employing the automatic LaTeX typesetting facility of Isabelle, we achieve on the one hand maximal accuracy of the presentation, retaining the mathematical rigor and the "flavor"[2] of the machine-checked specifications, and on the other hand good readability by using standard logical notation as far as possible and interspersing textual explanations and motivation.

A very important design principle is to keep a high level of abstraction, which improves readability and simplifies the proofs. Therefore, we model only those features that are strictly relevant for security, abstracting away unnecessary detail caused e.g. by efficiency optimizations. For the same reason we often use a modeling technique called underspecification, i.e. for part of the logical types and constants we do not give full definitions but only declarations of their names.

4.1 Overview

Following the standard approach to security analysis, we provide a system model describing the abstract behavior of the memory management and formalize the security objectives as properties of the system model. The state-based ISM approach fits well with modeling both the page table and the physical memory as state components of the system, mapping virtual addresses to physical addresses and physical addresses to values. Our specification of (both virtual and physical) addresses represents the structure of the memory organization as described in §2.1. Values are only of interest in case of PORT instructions; we leave other values unspecified. The model further includes mappings describing the assignments of BPFs to page blocks and EARs to sections of the virtual address space.

To complete the system model, state transitions represent the different kinds of memory access that may occur. For each of them there is a corresponding input message for the ISM triggering a transition. Each transition produces an output stating whether the access is granted or denied. In case of denial, we have different output messages representing the different traps or alarms. The computation of the output refers to our formalization of the protection rules stated in §2.3. A transition may also result in modifications of state components, for instance, write access to the main memory or page table updates.

4.2 Addressing

First we have to define several aspects of the SLE 88 address space introduced in §2.1. These include the type of package addresses, *PAD*, defined as the disjoint sum of privileged and regular PADs, where we enumerate all three possibilities for privileged packages but do not specify the actual range of regular PADs:

[1] Isabelle/HOL adopts the notational standards of functional programming, writing for instance (multi-argument) function application as `f x y z` instead of $f(x, y, z)$.

[2] For example, the order of definitions is strictly bottom-up.

datatype *pri_PAD = SL | PSL | OS* — package addresses 0 - 2
typedecl *reg_PAD* — package addresses 3 - 255
datatype *PAD = Pri pri_PAD | Reg reg_PAD* — package addresses, priv. or regular

Next, we define a predicate distinguishing privileged from regular packages.

```
consts    is_Pri :: "PAD ⇒ bool"
primrec  "is_Pri (Pri p) = True"
         "is_Pri (Reg r) = False"
```

While PADs form the upper (i.e., most significant) part of virtual addresses, displacements *DP* form the lower sections used for addressing individual bytes of memory within a page. We need to split them further because there are four page blocks within a page that are associated with their own BPFs. Note that despite the names that contain numbers giving bit positions, we do not actually specify the concrete ranges of the types declared but just state that *DP* is the Cartesian product of the two other types:

typedecl *DP_lo* — 4-bit offset within page block (with same BPF)
typedecl *DP_hi* — 2-bit page block address within page
types *DP* — 6-bit displacement within *VEA*s and *PEA*s
= *"DP_hi × DP_lo"*

A virtual effective address consists of the package address, a middle part that we call *VEA_mid*, and the displacement. We have to further split the middle part because only the upper 16 bits of it are used to determine the EAR associated with the address. We also define the type *VP* of virtual page pointers which will be mapped to physical page pointers.

typedecl *VEA_mid_lo* — 2-bit part of *VEA_mid* with identical EARs
typedecl *VEA_mid_hi* — 16-bit part of *VEA_mid* with different EARs
types *VEA_mid* — 18-bit middle part of *VEA*
= *"VEA_mid_hi × VEA_mid_lo"*
VEA_dEAR — 24-bit upper part of *VEA* determining EARs
= *"PAD × VEA_mid_hi"*
VEA — 32-bit virtual effective address
= *"PAD × VEA_mid × DP"*
VP — 26-bit virtual page pointer
= *"PAD × VEA_mid"*

Physical page pointers *PP* are combined with displacements to form physical effective addresses. The part of PEAs determining the BPF is called *PEA_dBPF*.

typedecl *PP* — 16-bit physical page pointer
types *PEA_dBPF* — 18-bit page block address determining the BPF
= *"PP × DP_hi"*
PEA — 22-bit physical effective address
= *"PP × DP"*

We define an auxiliary function *PAD* extracting the package information from any address containing a PAD as its uppermost part, simply by projecting on this first part of the tuple: `PAD (pad,x) = pad`

4.3 Effective Access Rights

We enumerate all allowed EARs as defined in §2.3 and relate them with the access that they grant by functions for intra-package and inter-package access.

```
datatype EAR = WW | WR | RR | Wn ("W-") | Rn ("R-") | Code ("X-")
datatype access_mode = Read | Write | Execute
consts   — access modes for read/write operations
  RWX_own   :: "EAR ⇒ access_mode set"
  RWX_other :: "EAR ⇒ access_mode set"
primrec  — intra-package access
 "RWX_own   WW   = {Read, Write}"
 "RWX_own   WR   = {Read, Write}"
 "RWX_own   RR   = {Read}"
 "RWX_own   W-   = {Read, Write}"
 "RWX_own   R-   = {Read}"
 "RWX_own   X-   = {Execute}"
primrec  — inter-package access
 "RWX_other WW   = {Read, Write}"
 "RWX_other WR   = {Read}"
 "RWX_other RR   = {Read}"
 "RWX_other W-   = {}"
 "RWX_other R-   = {}"
 "RWX_other X-   = {}"
```

4.4 State

Our abstract model of the SLE 88 memory management state consists of three aspects that are crucial for the security analysis:

- the physical memory contents (where the only sort of value we are interested in is a PORT instruction sequence with its associated set of packages) and the PASL predicate associated with page blocks
- the essentials of the page table entries, i.e. page mapping and EARs – there is no need for us to model complex structures like translation lookaside buffers and multi-level page tables required merely for optimization
- the package information contained in the current program counter and in the return address stack

For simplicity, we model PORT instructions as atomic values. We define them as one of the alternatives in a (free) datatype, which implies that they can be distinguished from all other instructions. This is adequate because the SLE 88 instruction layout ensures that PORT instructions are uniquely determined.

```
typedecl value'
datatype value = PORT "PAD set"  — specifying the packages permitted to enter
               | Other_value value'
```

The abstract state itself is defined as a record. Each of the field names induces a corresponding selector function whose first argument is a value, typically called *s*, of type *state*.

```
record state =
— abstraction of physical memory:
  memory    :: "PEA       ⇒ value"  — including peripherals
  BPF_PASL  :: "PEA_dBPF  ⇒ bool"   — BPF stating SL-only access to page blocks
— abstraction of page table (package descriptions and translation lookaside buffers):
  PT_map    :: "VP        ⇀ PP"     — page mapping, relative to packages
  PT_EAR    :: "VEA_dEAR  ⇒ EAR"    — EARs for 256-byte sections
— abstraction of execution state:
  curr_PAD  :: "PAD"                — currently executing package
  stack     :: "PAD list"           — package part of return addresses
consts s0 :: state  — the initial state
```

The state components *BPF_PASL*, *PT_map*, and *PT_EAR* each define a mapping only for the relevant sections of physical and virtual addresses, which helps to avoid redundancies in particular for update operations. Yet it is often convenient to perform the lookup operation with a full PEA or VEA, respectively. The auxiliary functions *BP_PASL*, *PEA*, and *EAR*, respectively, provide these liftings.

4.5 Assumed Initial State Properties

The security target [WN03] requires that all EARs should be initialized with a reasonable value. Since the exact value is immaterial for our analysis, we apply the standard technique, viz. to declare a constant giving the default EAR of memory sections without actually defining its value.

```
consts default_EAR :: "EAR"  — underspecified
```

The functional specification requires that only the PSL package may call the SL package, which restricts the sets of packages within PORT instructions of SL. We specify this for the initial state *s0* with the following axiom:

```
axioms  — checks by PORT instructions of SL
 init_PORT_SL: "PEA s0 (Pri SL, la) = Some pa ⟹
   memory s0 pa = PORT PADs ⟹ PADs ⊆ {Pri SL,Pri PSL}"
```

The axiom can be read as follows. For any VEA that belongs to SL (i.e., has the form *(Pri SL, la)* for some *la*), if in the initial state it is mapped to any PEA *pa* and a PORT instruction is stored at that address, then the associated set *PADs* of allowed packages may contain only SL and PSL.

A further important requirement is that the BPFs are reasonably set: for any physical pointer *pp* and page block address, PASL should be true iff the page block is owned by SL, i.e. *pp* is associated with some VEA belonging to SL:

```
axioms init_BPF_PASL:
  "BPF_PASL s0 (pp,pb) = (∃la. PT_map s0 (Pri SL,la) = Some pp)"
axioms init_PT_EAR: "PT_EAR s0 ea = default_EAR"
```

It is evident that the processor should start executing in the SL package with an empty return stack. Though we do not actually need these properties in our proofs, we state them for symmetry:

```
axioms
  init_PAD:   "curr_PAD s0 = Pri SL"  — unused
  init_stack: "stack    s0 = []"      — unused
```

4.6 Aliasing via Page Table

By the construction of the page table mapping, there is the possibility that the mapping is non-injective, i.e., that multiple VEAs refer to the same PEA. The MMU device driver in the PSL package avoids such aliasing, but the page table may be manipulated directly by privileged packages in order to meet extraordinary needs for inter-package sharing. Our model is general enough to handle also such forms of aliasing. Naturally, in such cases the guarantees that can be made are weaker. In particular, conflicting EARs may arise, for example if a certain memory page is mapped for two different packages where one package sets the EAR such that all others should not be able to write to that memory page, while the other package claims to have write access by setting its EAR accordingly. We have identified a predicate on the page table contents that specifies conditions as weak as possible but still guaranteeing inter-package consistency of EARs: if two different packages `p` and `p'` happen to map the same memory page then the EARs associated with that page should be both `WW` or both `RR`.

```
constdefs
  EARs_consistent :: "state ⇒ bool"
 "EARs_consistent s ≡ ∀p p' vea_mid_hi vea_mid_hi' lo lo'.
        PT_map s (p,vea_mid_hi,lo) = PT_map s (p',vea_mid_hi',lo') ⟶
        PT_map s (p,vea_mid_hi,lo) = None ∨ p = p' ∨
        PT_EAR s (p,vea_mid_hi) = WW ∧ PT_EAR s (p',vea_mid_hi') = WW ∨
        PT_EAR s (p,vea_mid_hi) = RR ∧ PT_EAR s (p',vea_mid_hi') = RR"
```

4.7 Interface

We define the SLE 88 memory management system as an Interacting State Machine (ISM) with a rather trivial interface: it has one input port named `In` and one output port named `Out`.

```
datatype interface = In | Out
```

The messages exchanged with the environment are either instructions given to the system or results sent by the system. The instructions are abstractions of the usual CPU (micro-)instructions where we focus on code fetch (which is the first step of each instruction execution), memory read and write, various forms of branches, and write operations to various special registers including the page table. The chip may respond with positive or negative acknowledge or various traps (which will be explained where appropriate) in case of denied access.

```
datatype message =
  Code_Fetch  VEA  — is meant to precede each other type of instruction
— read/write operations:
| Read_Mem    VEA
| Write_Mem   VEA value
— control transfer operations:
| Jump VEA
| Call VEA
| Return
| Write_RetAddr   VEA
— operations for setting security attributes and page table entries:
| Write_BPF_PASL  PEA_dBPF  bool
| Write_PT_EAR    VEA_dEAR  EAR
| Write_PT_map    VP        "PP option"
— outcome of operations:
| Ok | No  — access granted or denied without generating a trap
| MPA | MPSF | RLCP | MPBF | PRIV | MCR  — traps
```

4.8 Auxiliary Access Functions

For modeling the access control checks performed when executing access operations, it is beneficial to factor out common behavior and to reduce the complexity of the associated system transitions by defining dedicated auxiliary functions.

The function `mem_access` takes as its arguments the access mode, the current system state `s`, and the virtual address `va` to be accessed. It determines whether the current package, which is the subject (called *source*) of the operation at hand, is allowed to access — in the given mode — the package given by the PAD of `va`, which is the object of the operation (called *target*). In particular, it checks whether

– the virtual address is mapped to some existing PEA `pa` (and otherwise causes a Memory Protection Package Boundary Fault trap)
– the source is privileged and performs a read or write access where the target is some other package except SL[3], or the EAR associated with `va` allows access with the given mode, making the distinction if the access is local or to some other package (and otherwise causes a Memory Protection Access Violation or MPBF trap)
– PASL is true for `pa` iff the target is SL (which checks consistency of the PASL setting), or SL accesses data – for testing purposes – in a page block not belonging to SL where PASL is true (and otherwise causes a Memory Protection Security Field trap).

```
constdefs  — read/write access restrictions to main memory
 mem_access :: "access_mode ⇒ state ⇒ VEA ⇒ message"
```

[3] This operation is typical for e.g. the operating system loading a package.

```
"mem_access mode s va ≡ case PEA s va of None ⇒ MPBF | Some pa ⇒
  let source = curr_PAD s; target = PAD va in
  (if is_Pri source ∧ mode≠Execute ∧ source≠target ∧ target≠Pri SL ∨
      mode ∈ (if source=target then RWX_own else RWX_other) (EAR s va)
   then (if ((BP_PASL s pa = (target = Pri SL)) ∨ BP_PASL s pa ∧
                   source = Pri SL ∧ mode ≠ Execute ∧ target ≠ Pri SL)
         then Ok else MPSF)
   else (if mode ≠ Execute then MPA else MPBF))"
```

The function `Call_access` takes as its arguments the current system state `s` and the virtual address `va` to be called. It grants intra-package calls (i.e., the PAD of the target `va` equals the current PAD), and otherwise checks whether

- `va` is mapped to some PEA `pa` (and otherwise causes a Memory Protection Package Boundary Fault trap)
- the value stored at `pa` is a PORT instruction (and otherwise causes a Privileged Instruction trap)
- the PORT instruction allows the current package to enter (and otherwise typically just returns from the call without causing a trap).

constdefs — restrictions for procedure calls

```
 Call_access :: "state ⇒ VEA ⇒ message"
"Call_access s va ≡ if PAD va = curr_PAD s then Ok else
     case PEA s va of None ⇒ MPBF | Some pa ⇒
    (case memory s pa of PORT PADs ⇒
     if curr_PAD s ∈ PADs then Ok else No | Other_value v ⇒ PRIV)"
```

The function `Write_PT_access` takes as its arguments the current system state `s` and the package to be affected. Writing to the page table is granted only if the current package is privileged. It must be even SL if the target package is SL. Otherwise a Memory Protection Core Register Address trap is generated.

constdefs — restrictions for writing page table information

```
 Write_PT_access :: "state ⇒ PAD ⇒ message"
"Write_PT_access s target ≡ if (target=Pri SL ⟶ curr_PAD s=Pri SL) ∧
                                 is_Pri (curr_PAD s) then Ok else MCR"
```

4.9 Transitions

The core of our security model is the definition of the ISM that specifies the overall memory management system of the SLE 88. For each kind of instruction that may be issued (by sending it to the ISM) there is one transition rule. Transitions are atomic and instruction execution is meant to be sequential. The system reacts by outputting a value that indicates granted or denied access, where the latter typically leads to a trap. In our abstract model there is no need to specify trap handling. A couple of the transition rules have preconditions, and most of them have postconditions specifying changes to (part of) the system state. Since conditional changes to mappings are very common, we define the syntactic abbreviation "`c ? f(x:=y)`" ⇀ "`if c then f(x := y) else f`".

```
ism SLE88_MM =
  ports interface
    inputs  "{In}"
    outputs "{Out}"
  messages message  — instructions received or indications of success sent
  states data state init "s0" name "s"  — the initial state is s0
  transitions
Code_Fetch:  — Okay if the PAD of va equals the current PAD and has the EAR X-
and va is mapped to some page block where PASL is true iff the current PAD is SL.
  in  In  "[Code_Fetch va]"
  out Out "[mem_access Execute s va]"
Read_Mem:
  in  In  "[Read_Mem va]"
  out Out "[mem_access Read s va]"
Write_Mem:  — Sets the memory cell at address va to the value v by the value v if
access is granted. If the target package is SL and PASL is false for the affected page
block, it may non-deterministically – as specified using the free variable belated_MPSF
– write the value even though the access is denied, namely if the MPSF trap is delayed.
  in  In  "[Write_Mem va v]"
  out Out "[mem_access Write s va]"
  post memory := "(mem_access Write s va = Ok ∨
                   mem_access Write s va = MPSF ∧ belated_MPSF ∧
                   PAD va = Pri SL ∧ ¬BP_PASL s (the (PEA s va))) ?
                  (memory s)(the (PEA s va) := v)"
Jump:  — Only intra-package jumps are permitted.
  in  In  "[Jump va]"
  out Out "[if PAD va = curr_PAD s then Ok else MPA]"
Call:  — If the call is allowed then the current PAD is updated and its old value is
pushed on the abstract return stack.
  in  In  "[Call va]"
  out Out "[Call_access s va]"
  post curr_PAD:="if Call_access s va = Ok then PAD va else curr_PAD s",
       stack   :="if Call_access s va = Ok then curr_PAD s#stack s else
                                                                stack s"
Return:  — The first precondition states that the stack is non-empty with top element
r while the second precondition just gives an abbreviation. A return into SL is not
allowed, causing a Return Leave Current Package Mistake trap, otherwise r is popped
from the stack and becomes the new current PAD.
  pre      "stack s = r#rs", "ok = (r = Pri SL ⟶ curr_PAD s = Pri SL)"
  in  In  "[Return]"
  out Out "[if ok then Ok else RLCP]"
  post curr_PAD := "if ok then r  else curr_PAD s",
       stack    := "if ok then rs else stack s"
```

Write_RetAddr: — Setting the return address, i.e. the stack top, to an address whose PAD is different from the current one is possible only for privileged packages.

```
  pre "stack s = r#rs", "ok = (PAD va=curr_PAD s ∨ is_Pri (curr_PAD s))"
  in  In  "[Write_RetAddr va]"
  out Out "[if ok then Ok else No]"
  post stack := "if ok then (PAD va)#rs else stack s"
```

Write_BPF_PASL: — Only SL is allowed to change the block protection field.

```
  in  In  "[Write_BPF_PASL ba b]"
  out Out "[if curr_PAD s = Pri SL then Ok else MCR]"
  post BPF_PASL := "curr_PAD s = Pri SL ? (BPF_PASL s)(ba:=b)"
```

Write_PT_EAR:

```
  in  In  "[Write_PT_EAR ea e]"
  out Out "[Write_PT_access s (PAD ea)]"
  post PT_EAR := "Write_PT_access s (PAD ea) = Ok ? (PT_EAR s)(ea:=e)"
```

Write_PT_map:

```
  in  In  "[Write_PT_map vp ppo]"
  out Out "[Write_PT_access s (PAD vp)]"
  post PT_map := "Write_PT_access s (PAD vp) = Ok ? (PT_map s)(vp:=ppo)"
```

Having given all the above definitions, we use them for stating and proving security properties. Many of these require additional assumptions on reasonable behavior of the SL package, which we will give as additional axioms restricting the transitions of the ISM.

5 Security Properties

5.1 Overview

Given the system model in the form of an ISM, we are ready to formalize the security requirements of §2.2 as properties of (sequences of) ISM state transitions. Since the security requirements are formulated on a very high level, expressing the properties and arguing for their completeness has been appropriately done by discussing them with the requirement engineers, taking into account the SLE 88 specifications and the justifications given in the security target, which define details like access modes, EARs, the PASL attribute, and their intended effect.

The main concern of the security requirements is separation of applications, i.e., suitable restriction of inter-package access, which we address by the theorems

– *interpackage_Read_Mem_respects_EAR* and *interpackage_Write_Mem_respects_EAR*, addressing inter-package read/write protection, described in §5.2, and
– *interpackage_transfer_only_via_Call_to_suitable_PORT_or_Return*, addressing inter-package control transfer, described in §5.4.

Another critical issue is the special protection of the SL package because it manages the security attributes, onto which access control is based. By stating the series of theorems given in §5.3 culminating in *only_SL_changes_SL_memory* and *only_SL_reads_SL_memory*, and in §5.4 culminating in *only_PSL_enters_SL*, we have covered all properties implied by the security requirements.

Proving that the theorems hold for the given system model completes the formal security analysis. The proofs show some inherent complexity, for instance by having to consider layered protection mechanisms and effects of aliasing, i.e., non-injective page table mappings. Still, due to adequate modeling and the powerful Isabelle proof system, developing the machine-checked proofs has been a matter of just a few days.

The act of conducting proofs identifies necessary assumptions concerning the initial state and the access control attribute settings for the SL package. In particular, we introduce a notion of consistency of EAR assignments that is useful in case of aliasing.

5.2 Inter-package Read/Write Protection

Our first two theorems state basic properties of inter-package read and write access. If in any state *s*, a read instruction for some virtual address *va* not belonging to the current package is successful, then this has been done by a privileged package accessing a package other than SL, or read (or read/write) access is granted by the EAR associated with *va*. Note that the access rights are determined at the virtual (not: physical) address level, which opens up the possibility of inconsistencies incurred by aliasing, i.e. different access paths to the same physical memory area. In effect, the accessibility of a memory area is determined by the minimum protection of all related EARs. Only if inter-package consistency of the EARs is ensured, we can guarantee that for any other virtual address *va'* belonging to a different package and mapped to the same physical address, the associated EAR is the same (and cannot be *WR* because this EAR is not symmetric) and thus no unwanted access is possible.

```
theorem interpackage_Read_Mem_respects_EAR: "⋀va va'.
⟦((p,s),c,(p',s')) ∈ Trans; hd (p In) = Read_Mem va; hd (p' Out) = Ok;
 PAD va ≠ curr_PAD s⟧ ⟹ is_Pri (curr_PAD s) ∧ PAD va ≠ Pri SL ∨
   (EAR s va = WW ∨ EAR s va = WR ∨ EAR s va = RR) ∧
   (EARs_consistent s ⟶ PEA s va' = PEA s va ⟶ PAD va ≠ PAD va' ⟶
                          EAR s va' = EAR s va ∧ EAR s va ≠ WR)"
```

Some notational remarks are advisable here: in Isabelle formulas, '⋀' is a universal quantifier; multiple premises are bracketed using '⟦' and '⟧' and separated using '`;`'. The term *hd (p In)* refers to the input and *hd (p' Out)* to the output of the transition *((p,s),c,(p',s'))* which takes the state *s* to *s'*.
The proof of this theorem proceeds by case distinction on the transition rules. The only non-trivial case is the one of *Read_Mem* where we unfold the definitions of *mem_access*, *RWX_other*, and *EARs_consistent* and perform standard predicate-logical reasoning and term rewriting.

The analogous theorem concerning the write instruction is a bit simpler because there are less cases that allow write access:

```
theorem interpackage_Write_Mem_respects_EAR: "⋀va va'.
⟦((p,s),c,(p',s')) ∈ Trans; hd (p In)=Write_Mem va v; hd (p' Out)=Ok;
 PAD va ≠ curr_PAD s⟧ ⟹ is_Pri (curr_PAD s) ∧ PAD va ≠ Pri SL ∨
```

```
EAR s va = WW ∧ (EARs_consistent s ⟶ PEA s va' = PEA s va ⟶
                                 PAD va ≠ PAD va' ⟶ EAR s va' = WW)"
```

5.3 Read/Write Protection for SL Memory

The next bunch of lemmas and theorems focus on the protection of the memory areas of the SL package.

Only SL may change the mapping of PEAs belonging to SL. More precisely, for any sequence of transitions `ts` that may result from a system run and any state transition from `s` to `s'` within it, unless the current package is SL, the page table mapping concerning SL is the same for `s` and `s'`. This is a simple consequence of the definition of `Write_PT_access` used in the rule `Write_PT_map`.

theorem *only_SL_changes_PT_map_of_SL:*
```
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts; curr_PAD s ≠ Pri SL⟧ ⟹
  PT_map s (Pri SL, lvp) = PT_map s' (Pri SL, lvp)"
```

The analogous property holds for the EARs associated with SL memory:

theorem *only_SL_changes_EAR_of_SL:*
```
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts; curr_PAD s ≠ Pri SL⟧ ⟹
  EAR s (Pri SL, lva) = EAR s' (Pri SL, lva)"
```

Similarly, only privileged packages may change EARs:

theorem *only_Pri_change_EAR:*
```
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts; ¬is_Pri (curr_PAD s)⟧ ⟹
  EAR s va = EAR s' va"
```

Later we will need an invariant stating that the EARs associated with SL deny any access by other packages. In order to establish this property, we have to assume that the default EAR denies such access as well and that SL sticks to this policy when writing EARs:

axioms *default_EAR_denies_RWX_other:* `"RWX_other default_EAR = {}"`

axioms *Write_PT_EAR_consistent_with_denial_of_RWX_other_for_SL_memory:*
```
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s = Pri SL;
  hd (p In) = Write_PT_EAR (Pri SL, lva) e; hd (p' Out) = Ok⟧ ⟹
  RWX_other e = {}"
```

The necessity of such axioms makes explicit some important assumptions on the initialization of security attributes and the behavior of SL and therefore gives valuable feedback for system software development.

With the help of the two axioms just given and the axiom `init_PT_EAR` given in §4.5, we can prove the invariant easily by induction on the length of transition sequences. In terms of the Isabelle/HOL implementation of ISMs, this invariant can be expressed in the following compact way:

lemma *SL_pages_deny_RWX_other:*
```
  "Inv (λs. ∀lva. RWX_other (EAR s (Pri SL, lva)) = {})"
```

It reads as follows. For any reachable state *s* and any virtual address within the SL package, the associated EARs for other packages is the empty set.
Similar comments apply to the invariant that PASL is true for all memory belonging to SL. It requires the additional assumptions that SL writes the block protection fields and the page table entries for its memory only in a way such that PASL remains true:

```
axioms Write_BPF_PASL_consistent_for_SL_memory:
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s = Pri SL;
  hd (p In) = Write_BPF_PASL (pp,dp') b; hd (p' Out) = Ok;
  PT_map s (Pri SL, lvp) = Some pp⟧ ⟹ b = True"
```

```
axioms Write_PT_map_consistent_with_BP_PASL_for_SL_memory:
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s = Pri SL;
  hd (p In) = Write_PT_map (Pri SL, lvp) (Some pp); hd (p' Out) = Ok⟧ ⟹
  BP_PASL s (pp,dp)"
```

Together with the axiom *init_BPF_PASL* also given in §4.5, we can prove the invariant in an analogous way.

```
lemma SL_memory_has_PASL:
 "Inv (λs. ∀lva pa dp. PEA s (Pri SL, lva) = Some pa ⟶ BP_PASL s pa)"
```

Taking advantage of the two invariance lemmas just given, we prove that only SL can change memory allocated to SL. The proof uses the invariant *SL_memory_has_PASL* concerning PASL three times, where in all these cases there is aliasing in the page table such that the same physical memory area is allocated to both SL and some non-SL package. Thus we can conclude that PASL plays an important role for detecting such (unwanted) aliasing wrt. SL memory.

```
theorem only_SL_changes_SL_memory:
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts; curr_PAD s ≠ Pri SL;
  PEA s (Pri SL, lva) = Some pa⟧ ⟹ memory s pa = memory s' pa"
```

The theorem stating that only SL can read memory allocated to SL requires only the invariant *SL_pages_deny_RWX_other* concerning EARs of SL:

```
theorem only_SL_reads_SL_memory:
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts;
  hd (p In) = Read_Mem (Pri SL, lva);  hd (p' Out) = Ok⟧ ⟹
  curr_PAD s = Pri SL"
```

5.4 Inter-package Control Transfer and PORT Instructions

As can be derived easily from the transition rule for the `Code_Fetch` operation and the definition of `mem_access`, code may be executed only from memory that belongs to the current package and that is marked with the EAR `X-`:

```
theorem Code_Fetch_only_local_X:
"⟦( (p,s),c,(p',s')) ∈ Trans; hd (p In) = Code_Fetch va; hd (p' Out) = Ok⟧
 ⟹ PAD va = curr_PAD s ∧ EAR s va = X-"
```

Thus, the only form of inter-package code access to be further addressed is transfer of control where the current package changes.

Our next theorem states that the only possibilities for such control transfer is a legal procedure call or return; in more detail: if there is a transition from state *s* to *s'* where the current package changes, then either it has been caused by a call whose target is a virtual address *va* mapped to a physical address *pa* containing a PORT instruction that explicitly allows the calling package to enter, or it has been caused by a return to a package other than SL:

```
theorem interpackage_transfer_only_via_Call_to_suitable_PORT_or_Return:
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s' ≠ curr_PAD s⟧ ⟹
  (∃va pa PADs. hd (p In) = Call va ∧ PEA s va = Some pa ∧
                memory s pa = PORT PADs ∧ curr_PAD s ∈ PADs) ∨
  (∃r rs.       hd (p In) = Return ∧ stack s = r#rs ∧ r ≠ Pri SL)"
```

The proof of this theorem is straightforward by case distinction on all instructions available and unfolding the definition of *Call_access*.

Much more involved is the proof of our final theorem stating that only PSL can enter SL: we need an invariant that all PORT instructions contained in memory allocated to SL allow only calls by SL itself and by PSL. This in turn requires two assumptions that SL writes memory allocated to itself and the page table entries for its memory only in a way such that the invariant is maintained:

```
axioms Write_Mem_PORT_to_SL_only_SL_PSL:
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s = Pri SL;
  hd (p In) = Write_Mem va (PORT PADs); hd (p' Out) = Ok;
  PEA s va = PEA s (Pri SL, lva)⟧ ⟹ PADs ⊆ {Pri SL, Pri PSL}"
```

```
axioms Write_PT_map_pointing_to_PORT_only_SL_PSL:
"⟦((p,s),c,(p',s')) ∈ Trans; curr_PAD s = Pri SL;
  hd (p In) = Write_PT_map (Pri SL, lvp) (Some pp); hd (p' Out) = Ok;
  memory s (pp,dp) = PORT PADs⟧ ⟹ PADs ⊆ {Pri SL, Pri PSL}"
```

Note the essential role of aliasing in the first of these axioms: the instruction intended to write a PORT instruction at virtual address *va* might affect SL memory even if *va* does not belong to SL, namely if there is some other virtual address *(Pri SL, lva)* that happens to be mapped to the same physical address.

Apart from the two axioms, the proof of the invariant requires the axiom *init_PORT_SL* given in §4.5 as well as the theorems *only_SL_changes_PT_map_of_SL* and *only_SL_changes_SL_memory*.

```
lemma SL_PORT_SL_PSL:
"Inv (λs. ∀lva pa PADs. PEA s (Pri SL, lva) = Some pa ⟶
          memory s pa = PORT PADs ⟶ PADs ⊆ {Pri SL, Pri PSL})"
```

Exploiting the invariant, the theorem can be proven in just a few steps. It reads as follows: for any sequence of transitions *ts* that may result from a system run and any state transition from *s* to *s'* within it, if SL becomes the current package in *s'*, the current package of the pre-state *s* must have been PSL.

theorem *only_PSL_enters_SL:*

```
"⟦ts ∈ TRuns; ((p,s),c,(p',s')) ∈ set ts; curr_PAD s ≠ Pri SL;
  curr_PAD s' = Pri SL⟧ ⟹ curr_PAD s = Pri PSL"
```

This finishes our abstract formal analysis of the SLE88 memory management.

6 Conclusion

We have introduced a security model for the memory management of the SLE88 smart card processor chip. Memory management contributes to security by providing access control mechanisms on the levels of both virtual and physical memory addresses, allowing to separate applications and privileged SW packages (e.g., the operating system and the security layer SL) as well as applications from each other. Access control is guided by a policy comprising both discretionary (by effective access rights EAR) and mandatory (wrt. SL and privileged packages) rules. Enforcing the policy is non-trivial: the ability to change EARs and address mappings, the interacting levels of protection, aliasing in address translation, inter-package calls, and the peculiarities of SL have to be considered.

The model gives an abstract view of the SLE88 by concentrating on memory access and its protection only, leaving out details of the system and application functionality. Abstraction is achieved by reductions on the data structure and interface design and by underspecification. For instance, many data types used in the model are declared but not actually defined.

Carrying out the formal modeling work turned out to be worthwhile, because it provided new insights and lead to clarification of so far fuzzy concepts. Formal reasoning resulted in a minimal set of requirements on non-injective address mappings that guarantee the maintenance of the security properties. These requirements are given by restrictions on admissible combinations of EAR settings. The derived notion of EAR consistency is the least restrictive one preserving security and offers much more flexibility compared to simply forbidding aliasing. Formal analysis showed that security depends on assumptions on the initial state (e.g., initial EAR and PASL settings) as well as on benign behavior of SL. The assumptions can be interpreted as requirements on configuration upon delivery and on the software development of privileged packages. They clearly indicate the distribution of responsibility between the Target of Evaluation and its environment. Last, but not least, formal arguments lead to a clarification of the role of PASL: the PASL mechanism does not provide additional protection in case of weak EARs for SL, but protects against effects resulting from undesired mapping of both SL and non-SL virtual addresses to the same physical address.

To summarize, results of the modeling and proving process are the identification of relevant assumptions on the system environment and the derivation of new insights in the memory management and its security properties. The cost-benefit ratio is adequate: the whole work required no more than a six weeks effort, largely due to the availability of adequate tool support through Isabelle/HOL and the ISM approach. Thus, the SLE88 memory management security model is an excellent example for the value of formal security modeling in practical industrial-scale applications.

References

[AHIP01] Atmel, Hitachi Europe, Infineon Technologies, and Philips Semiconductors. Smartcard IC Platform Protection Profile, Version 1.0, July 2001. `http://www.bsi.de/cc/pplist/ssvgpp01.pdf`.

[AHIP02] Atmel, Hitachi Europe, Infineon Technologies, and Philips Semiconductors. Smartcard Integrated Circuit Platform Augmentations, Version 1.0, March 2002. `http://www.bsi.de/cc/pplist/augpp002.pdf`.

[CC99] Common Criteria for Information Technology Security Evaluation (CC), Version 2.1, 1999. ISO/IEC 15408.

[LKW99] Volkmar Lotz, Volker Kessler, and Georg Walter. A Formal Security Model for Microprocessor Hardware. In *Proc. of FM'99 World Congress on Formal Methods*, volume 1708 of *LNCS*, pages 718–737. Springer-Verlag, 1999.

[LT89] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. `http://theory.lcs.mit.edu/tds/papers/Lynch/CWI89.html`.

[MT00] Stephanie Motre and Corinne Teri. Using B method to formalize the Java Card runtime security policy for a Common Criteria evaluation. In 23rd *National Information Systems Security Conference*, 2000. `http://csrc.nist.gov/nissc/2000/proceedings/toc.html`.

[Ohe02] David von Oheimb. Interacting State Machines: *a stateful approach to proving security.* In Ali Abdallah, Peter Ryan, and Steve Schneider, editors, *Proceedings from the BCS-FACS International Conference on Formal Aspects of Security 2002*, volume 2629 of *LNCS*. Springer-Verlag, 2002. `http://ddvo.net/papers/ISMs.html`.

[OL02] David von Oheimb and Volkmar Lotz. Formal Security Analysis with Interacting State Machines. In Dieter Gollmann, Günter Karjoth, and Michael Waidner, editors, *Proc. of the* 7th *European Symposium on Research in Computer Security (ESORICS)*, volume 2502, pages 212–228. Spinger, 2002. `http://ddvo.net/papers/FSA_ISM.html`. A more detailed journal version is submitted for publication.

[OL03] David von Oheimb and Volkmar Lotz. Generic Interacting State Machines and their instantiation with dynamic features. In *Proc. of the* 5th *International Conference on Formal Engineering Methods (ICFEM).* Spinger, November 2003. `http://ddvo.net/papers/GenISMs.html`, to appear.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994. For an up-to-date documentation, see `http://isabelle.in.tum.de/`.

[PNW$^+$] Lawrence C. Paulson, Tobias Nipkow, Markus Wenzel, et al. The Isabelle/HOL library. `http://isabelle.in.tum.de/library/HOL/`.

[SRS$^+$00] G. Schellhorn, W. Reif, A. Schairer, P. Karger, V Austel, and D. Toll. Verification of a formal security model for multiapplicative smart cards. In Frédéric Cuppens, Yves Deswarte, Dieter Gollmann, and Michael Waidner, editors, *Proc. of the* 6th *European Symposium on Research in Computer Security (ESORICS)*, volume 1895. Spinger, 2000.

[WN03] Georg Walter and Jürgen Noller, Infineon. SLE88CX720P / m1491 Security Target. `http://www.bsi.de/???0215???`, Version 1.00, March 2003.

Bridging Model-Based and Language-Based Security

Rogardt Heldal and Fredrik Hultin

Chalmers University of Technology, SE-412 96 Göteborg, Sweden
heldal@cs.chalmers.se, fredrik@hultin.info

Abstract. We present a way to support the development of software applications that takes into account confidentiality issues, and how the developed code can be automatically verified. We use the Unified Modelling Language (UML) together with annotations to permit confidentiality to be considered during the whole development process from requirements to code. We have provided support for software development using UML diagrams so that the code produced can be be validated by a language-based checker, in our case Jif (Java information flow). We demonstrate that the combination of model-based and language-based security is compelling.

1 Introduction

Our philosophy is that it should be convenient to consider security during the system development process, and that security should be automatically verifiable at code level. Addressing both of these aspects of system development is important to make a secure software system. We will show the benefits of combining a modelling language with a language-based security checker.

The development of software systems using UML [RJB99,OMG] has become the *de facto* standard for modelling object-oriented software systems in industry. There are several reasons for this: it is relatively easy to understand and learn, it permits several views of software systems, and it gives a good overview of the software's architecture. We aim to make it possible to consider security during a development process used in industry today, so UML is the obvious starting point.

One of the main problems with UML is that there has been a focus on functionality and less on constraints such as security. We want to use UML together with security annotations in such a way that developing secure programs becomes a seamless part of a project. This might seem like a difficult task since security requires rigorous treatment. Here, language-based checkers play an important role. In this approach, security information is derived from a program written in a high-level language during the compilation process and is included in the compiled object. This extra security information can take several forms including a formal proof or a type annotation. There have been several overview papers in this area [Koz99,SMH01,SM03]. Our combination of UML used together with annotation is intended to be used as a specification language to support building secure software systems, and a language-based checker should validate that the code really satisfies the security constraints. Therefore, our extended UML supports development of secure programs, and permits mistakes in the specification to be caught by the language-based checker. This is similar to modelling types in UML where the developer only needs to specify the types, and a type checker validates the types.

When we started to look for language-based security checkers, the Java information flow (Jif) system [Mye99a,Mye99b] was a natural choice since it handles a large subset of the object-oriented language Java [GJS96]. Object-orientation is important because UML is well suited to developing object-oriented systems. Jif is based on the Java language with some extra language constructs to control the release of data. The Jif system contains a type checker which guarantees that confidential data cannot leak. But, to make the system useful in practice, it permits data to be leaked in a controlled manner. This is acceptable provided that the system does not leak so much data that meaningful information can be derived. Technically, Jif deals with this problem in a simple way by giving part of the program authority to leak information[1].

In the standard security models, like the Bell-LaPadual model [BL73] and the Biba model [K.J77], the security policy is separated from the code. In this respect Jif differs in that the policy is incorporated into the code in the form of *labels*. Data are annotated with labels that specify the ownership and read permissions. The Jif type system checks whether the policies declared in the labels are satisfied. Jif is built on the decentralized label model [ML97,Mye99a]. In section 2 we will consider the labels of this model in more depth.

Java is not adequate for making programs which require tight control of confidentiality. Similarly, UML is not good for developing such programs. Therefore, we have created an extended version of UML, UMLS (Unified Modelling Language for Security). Our choice to support the development of Jif code had a large impact on how we extended UML. We did not extend UML in the standard way using UML's extension mechanisms (*stereotypes* and *tags* [OMG]) when modelling confidentiality. This was because we wanted more freedom in the choice of annotation in the current work. Furthermore, at present we do not automatically produce Jif code since we do not support any tools.

Several benefits follow from our work. Some of the diagrams — domain models, use-case diagrams, and activity diagrams — are so simple that most software system customers can be involved in the process of discussing confidentially issues. Customers are often the domain experts and they know best what information should be confidential. Therefore, involving the customer enhances the likelihood that confidentiality issues are handled correctly from the start. Furthermore, we have considered interaction diagrams and class diagrams where more detailed confidentiality issues can be considered by software designers. This permits confidentiality to be considered in greater depth during the development process.

[1] The Jif system has a way of dealing with information leaks, but no solution for deciding how much information can be leaked without causing problems. To solve this, information theory or complexity could be considered [VS00].

Domain models, use-case diagrams, activity diagrams, and interaction diagrams can be used to support the creation of the class diagrams[2]. Code skeletons can be created from class diagrams. There is still a lot of work to be done by the programmer, but there are confidentiality constraints on the attributes, operations, and classes which will guide/restrict the way the programmer will construct the code. This is a much simpler problem than writing the code without any support. The Jif compiler validates the code. If all confidentiality constraints are satisfied then the process is finished, otherwise the design/code has to be modified.

It is important to notice that UML diagrams cannot be validated on the same level as code. The code is needed to consider for example indirect information flow [DD77] and covert channels [Lam73]. Furthermore, the semantics of UML is still an open problem which makes it hard to prove things about UML. A further problem with validating the UML diagrams is that the transformation into code also has to be proven correct. We do not suggest our technique as an alternative for proving security on UML diagrams. It is often beneficial to prove security properties as early as possible. So, a combination of our technique and proving properties of the UML diagrams would be preferable.

In this paper we will first consider security labels similar to the ones used in Jif. Thereafter we will look at how we extended UML to consider confidentiality using labels. Then we will show a case study of how the extended UML diagrams can be used in a small process for developing a program which requires confidentiality. Finally, we will look at related work, conclusion, and future work.

2 Label

Modelling confidentiality in UMLS is done by using labels. Labels are used to specify the ownership and the read permissions of the data. We have chosen to use the same labels as used in Jif [Mye99b,ML97,ML98,ML00]. Types will be augmented with labels in UMLS.

Before we can discuss labels we first have to look at *principals* which are the building blocks of labels. A principal can be a user, a group, or a role. Principals can be arranged in hierarchies where a principal can act for another principal ("A can act for B" means that B can do anything that A can do). Principals are not purely static entities; they may also be used as values. First-class values of the new primitive type *principal* represents principals. For more information on run-time principals see [Mye99b].

To guarantee confidentiality the data needs to be annotated with labels. A label consists of policies, where a policy has the syntax: *owner:reader-list*.
The *owner* is a principal which owns the confidential data. This owner permits the principals in the *reader-list* to read the data. The *reader-list* is a list of comma separated principals that are able to read the data. Since a label can contain several policies, $\{policy_1; \ldots ; policy_n\}$ the data can be owned by several

[2] We construct the code skeleton from the class diagram, which is one of the best understood diagrams in UML.

principals. A principal can only read the data if all the owners permit this — the reader is included in all the reader lists of a label. An owner is implicitly a reader and the label {} is the least restricted label. Here is an example of a label where *Bob* is the owner and *Lise* and *Lars* are readers: {*Bob:Lise,Lars*}.

Labels may exits as run-time entities as well, represented by the new type *label*. For more information on run-time labels and their use see [Mye99b].

3 UMLS

In this section we will augment UML with the labels introduced in the previous section. The UML diagrams augmented with labels are the class, the interaction, and the activity diagram. We will also give the syntax and informal semantics for the extension made to UML. We have chosen to give the syntax and semantics in similar fashion as in the *OMG Unified Modelling Language Specification 1.4*[3][OMG]. In this section we will also show the syntax of UMLS and to make our extensions clear they will be set in bold type. Let us start by looking at how we can use labels and principals to annotate class diagrams.

3.1 Class Diagrams

In a software development process, class diagrams are among the last diagrams to be considered before code is created. They usually contain information about the class name, the attributes and the operations. This makes it straightforward to construct a code skeleton directly from class diagrams.

In this section we will look at how to annotate classes, their attributes and operations in UMLS. We will also describe the parameterised class and some issues concerning authorities.

Class. The class is the central symbol in the class diagram. A class is modelled with attributes and operations. In UMLS as in UML, attributes and operations have specified compartments. We have also defined a new compartment for giving the authority of the class[4], which is needed to be able to declassify confidential data, see figure 1. The concept of authority in UMLS will be discussed further when we look at authority constraints later in this subsection.

Here we can see that besides giving the attribute *name* a type it can also be given a label. In other words, the *type-expression* is augmented with a *label*. If the *label* is omitted on an attribute, that means that there are no confidentiality constraints associated with it. The syntax of an attribute is:

visibility name: type-expression ***label*** = *initial-value*

[3] For the purpose of the presentation we have simplified the OMG syntax where it has no impact on the confidentiality extension.

[4] Due to Jif the authority list cannot be inherited, meaning that if a class C has a superclass C_s, any authority in C_s must also be in the authority clause of C. It is not possible to obtain authority through inheritance.

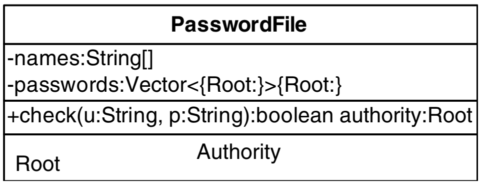

PasswordFile
-names:String[]
-passwords:Vector<{Root:}>{Root:}
+check(u:String, p:String):boolean authority:Root
Root
Authority

Fig. 1. Password file

The expression: *x: int*$\{Bob : Lise, Lars\}$ is an example of how to write an attribute with the type *int* augmented with a label where *Bob* is the owner and *Bob*, *Lise* and *Lars* can read the data. Now, when x is defined it can be used to restrict other variables, for example y : *int*$\{x\}$; meaning that the variable y should be as restricted as variable x.

The syntax of an operation is given by:

visibility name ***begin-label*** (*parameter-list*) ***end-label***
: *return-type-expression* ***return-label constraints***

where the *parameter-list* is a comma separated list of formal parameters, each given the syntax:

name : *type-expression* ***label***

Let us look at an example. Here is a public operation m with two arguments, x and y of type *int*, and a return value of type *String*. The two arguments are labelled with two different labels and in this case the return value is labelled with the joined label of the two arguments: +*m*(*x:int*{*Lise:*}, *y:int*{*Lars:*}) : *String*{*Lise:; Lars:*}

Labels may be omitted from an operation, signifying the use of implicit label polymorphism, e.g. the arguments of *check* in figure 1. When a formal argument's label is omitted, the operation is generic with respect to the label of the actual argument. We will come back to this when we consider interaction diagrams.

It is possible to specify the security context of operations with the *begin-label* and the *end-label*. The *begin-label* prevents a method from causing side effects that have lower security than the *begin-label*. The *end-label* specifies what information can be learned from the fact that the method terminates normally. For details the reader is referred to [Mye99a].

The default label for a return value is the *end-label*, joined with the labels of all the arguments. For example, for *check* in figure 1 the return label is $\{u; p\}$, so the return value could be written just as a *boolean*.

There are three types of constraints in UMLS; *authority*, *caller* and *actFor* [Mye99a]. In this paper we will only consider the authority constraint:

authority: principal-list This clause lists principals for which the operation is authorised to act. To be able to specify the authority of an operation the class needs to have at least this authority. For an example of how the operation

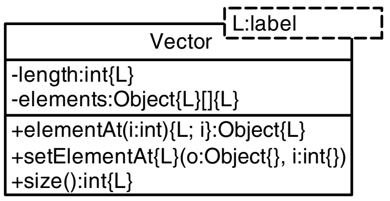

L:label
Vector
-length:int{L}
-elements:Object{L}[]{L}
+elementAt(i:int){L; i}:Object{L}
+setElementAt{L}(o:Object{}, i:int{})
+size():int{L}

Fig. 2. Vector class, showing attributes and operations

looks in a UMLS diagram see figure 1 where we have a class called *PasswordFile* with authority *root*. The authority is needed by operation *check* to be able to declassify information about whether the password is valid or not.

Parameterised Classes. Parameterised classes play an important role in UMLS for making reusable data structures with respect to labels and principals.

Let us look at an example. In figure 2 there is a class *Vector* parameterised on a label L (in the dotted box). This label is used to annotate attributes and operations of the class and makes it possible for *Vector* to be instantiated with different labels.

The attributes are annotated with the class's parameter label, L. From the figure 2 we can also see that *elements* has two labels. This is because an array needs special treatment. The first label is the label of the elements of the array. The second label is for the reference of the array.

The operation, *elementAt*, can be called with an index i as its argument. The *end-label* $\{L;i\}$ specifies what information can be learned by observing whether *elementAt* terminates normally. In this case the value returned will also have the same restrictions as the *end-label*.

In the operation *setElementAt* we need to prevent the method from causing side effect with a lower security level than $\{L\}$ by setting the *begin-label* to $\{L\}$. This is needed to be able to change any value in the array (which would fall into the category "causing side effects").

The specification of the *Vector* class put constraints on the Jif code written. The Jif code of *Vector* is given in Appendix A.

3.2 Relationships between Classes

There are many different types of relationships between classes. In this paper we need only to consider associations. They are modelled by drawing lines between the classes, see figure 7. Associations can contain *multiplicities* and *role names* [OMG] which we will see an example of in the case-study in section 4.

3.3 Interaction Diagrams

There are two types of diagrams for showing interactions between objects, the sequence diagram and the collaboration diagram. These two diagrams are similar

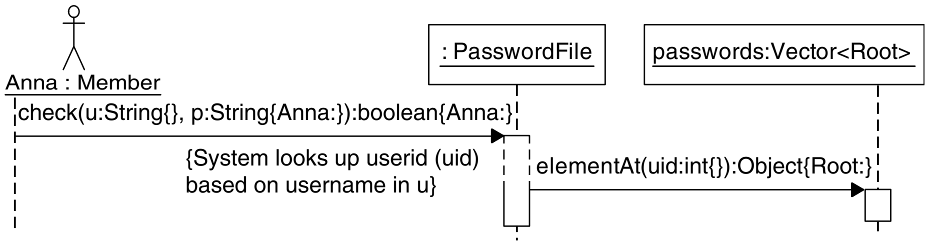

Anna : Member
: PasswordFile
passwords:Vector<Root>
check(u:String{}, p:String{Anna:}):boolean{Anna:}
{System looks up userid (uid) based on username in u}
elementAt(uid:int{}):Object{Root:}

Fig. 3. Password sequence

and for our perspective the distinction between them is not important. Interaction diagrams show dynamic properties of how objects work together to solve a larger problem, in contrast to class diagrams which shows static properties about classes, but these two diagrams are strongly related. For each message sent to an object in the interaction diagram there needs to be a matching operation in the class diagram[5].

Interaction diagrams are good for considering flow of data among objects. Data values flow to objects through arguments and back from objects via the result value. These values can be annotated with confidentiality constraints. Here is the syntax for the *sequence-expression*:

$$\begin{aligned}&\textit{return-value} := \ \textit{message-name}\ (\textit{argument-list}) : \\ &\qquad \textit{return-type}\ \boldsymbol{\textit{return-label}}\end{aligned}$$

where the syntax of *argument-list* is a comma separated list of arguments and labels: *argument* : *type-expression* ***label***.

Let us consider the sequence diagram in figure 3. First, *Anna* wants to check her password using *PasswordFile* from figure 1. The user name, *u*, is not confidential, but the password, *p*, is owned by principal *Anna*. Each user name is related to a number, *uid*, which is used to find the password in the vector from figure 2. By choosing the template parameter of the vector to be of principal *Root*, the data contained within the class will be owned by *Root*. Since this is the case the data returned from the vector must be at least as restrictive as *Root*. The password from the vector will be compared with the password from *Anna* producing a boolean value belonging to them both. Since *Anna* cannot read this value it has to be declassified. Here we have an interesting design question: how much authority should be given to *PasswordFile*? We decided to give *PasswordFile* the authority *Root* as this permits the method *check* to remove *Root* as owner of the boolean value returned to *Anna*. As we can see the interaction diagram helps to identify places where authority declarations must be considered.

In another scenario Bob might want to check his password using the principal *Bob*. This is no problem since the operation is defined as: *check*(u : *String*,

[5] There is one restriction on the use of interaction diagrams due to the fact that Jif cannot handle threads, so it makes no sense to talk about asynchronous communication. This is a limitation we hope will be removed in the future.

p : *String*) in *PasswordFile* which permits any label on u and p. The only change in the sequence diagram in figure 3 is that we change all principals *Anna* to *Bob*. Now, let us change the operation check belonging to *PasswordFile* to $check(u : String, p : String\{Anna :\})$. The sequence diagram in figure 3 will look the same, but now Bob cannot use the *check* operation any more. This distinction is shown in the class diagram, but not in the sequence diagram, because all principals are known in the sequence diagram.

3.4 Use Case Diagrams

Use case diagrams are used to describe the behaviour of the system in the form of use cases and the actors of the system — actors are the things which interact with the system through use cases.

The description of use cases is often done informally by description in running text. Confidentiality constraints might be considered as a part of use cases, but it is more natural to consider them as separate documents, which can be related to use cases. It is worth noting that interaction diagrams are often used to realise use cases. They are more formal than use cases and are therefore a better place to handle confidentiality constraints in a more formal way.

One benefit of using use case diagrams is that they identify the actors of the system. These actors can be used to define principal hierarchies.

3.5 Activity Diagrams

The last diagram type we will consider in this paper is the activity diagram. Activity diagrams can be used to model the flow of activities which happen in a system, a use case or a method. It is possible to show what kind of data are moved among activities within these diagrams. Furthermore, activity diagrams can contain *swimlanes* which can be used to separate the activities done by separate people, groups or organisations. This makes the activity diagrams perfect for showing how confidential data are moved among separate people, groups or organisations on an abstract level.

4 UMLS a Case Study

In this section we will show how UMLS can be used as part of a process, such as RUP[JBR99]. We will limit the discussion to the parts which are of interest when considering confidentiality. How to use UMLS in a process will vary depending on the project, in the same way as standard UML. Here we are going to look at stages where we found UMLS useful in a development process when considering confidentiality based on our case study.

4.1 The System

The example is that of a small medical application where patients can ask for information about diseases based on symptoms they provide. To obtain this information the patients also have to pay with a bank card. Since patients need to pay for the information, personal information that identifies the patients is also sent to the system. Since personal information needs to be sent to the system together with the symptoms the system could leak information about a particular patient's illness. We want to prevent this.

4.2 The Use Cases Diagram

Use-cases were developed to explore the behaviours of the system. Here we only consider the use case where a patient requires information about a disease. This use case contains interesting confidentiality issues. Due to the limited space we can only describe the *Casual Version* [Coc01] of the use case:

> **Use case: Obtain information about disease**
> The patient sends information about the symptoms and the payment to the medical-system. The medical-system validates the payment and charges the patient the specified amount. Based on the symptoms the medical-system looks up a matching disease, prepares a response and sends it to the patient.
> **Confidentiality constraints:** Any information sent to the system regarding the patient's symptoms are strictly confidential to the patient. Only the patient himself and the medical-system should be permitted to read the payment information. The medical-system should not leak more information than absolutely necessary to inform the patient about his illness[VS00].

As we can see from this description, interesting confidentiality issues can be considered at use-case level in an informal way. This description can easily be discussed with the customer.

4.3 The Activity Diagram

We use the activity diagram to better show the flow of confidential data in the medical system.

In figure 4 we have two swimlanes which separate the patient, here represented by *Lise*, and the medical system. It is interesting to consider what confidential data flows between the patient and the medical system. For example we can see that *Symptom* and *Payment* flow from the patient to the medical system. Look at *Payment*, this object contains *amount* and *cardNumber* which are owned by *Lise* and are readable by *Doctor*. In the case of *Symptom* we want the data to be owned by *Lise* but not readable by the medical system. This is because the patient do not trust the medical system and therefore want to prevent the system from sending the *Symptom* to an output channel such as a monitor or a printer. The medical system can still use the *Symptom* to find the correct treatment.

The interesting part is the activity *Lookup disease* which uses information from the patient, *Symptom*, and the doctor to find a disease. The result, *Disease*, contains data owned by both patient and doctor and therefore not readable by

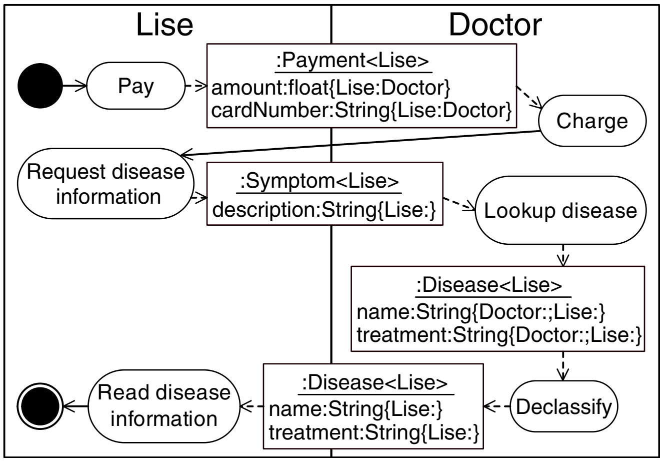

Lise
Doctor
Pay
:Payment<Lise>
amount:float{Lise:Doctor}
cardNumber:String{Lise:Doctor}
Charge
Request disease information
:Symptom<Lise>
description:String{Lise:}
Lookup disease
:Disease<Lise>
name:String{Doctor:;Lise:}
treatment:String{Doctor:;Lise:}
Read disease information
:Disease<Lise>
name:String{Lise:}
treatment:String{Lise:}
Declassify

Fig. 4. Medical system activity diagram

anyone. To make the data readable by the patient the data has to be declassified, which is done in the activity *Declassify*.

This diagram contains more information about the activities and flow of data than the use case diagram, but it is still quite informal. From this diagram we have obtained a better understanding of how the confidential data flows. Furthermore, we have started to consider principals, labels, and declassification. It is also natural to review this diagram together with the customer of the system.

4.4 The Domain

A domain model contains only concepts from the domain under consideration, and not software classes. A restricted form of class diagram is used for modelling domains, containing class names, attributes, and associations among classes. Since the domain model is a central part of the problem description this is an appropriate place to consider confidentiality.

Based on information in our use case and the activity diagram, we create a domain model[6], see figure 5. Notice that all the concepts in our domain model correspond to real world concepts: *Payment*, *Symptom*, *Patient*, *Doctor*, and *Disease*. A *Patient* is related to a *Payment*, *Symptom* and *Doctor* via associations. Furthermore, the multiplicity 1 on the association between *Patient* and *Payment* says that the *Patient* is associated with one *Payment* while the *Doctor* is associated with several *Diseases*, since the multiplicity is *. The names attached to one side of the associations are role names, here used to give the

[6] Some people prefer to model the system directly in the domain model, skipping the use cases. They believe that they obtain an object-oriented system of higher quality, which is easier to extend, reuse, and maintain.

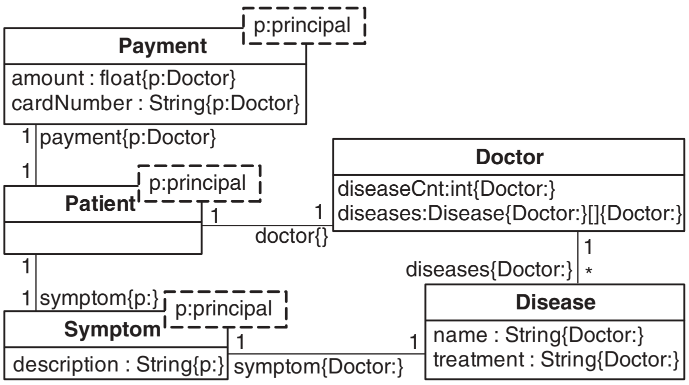


Fig. 5. Medical system domain mode

names of the attributes, for example *payment* is an attribute in *Patient* referring to *Payment*.

We will only consider a few of the confidentiality constraints considered in figure 5. The label {} on the role name *doctor*{} shows that the reference is not confidential. But, the attributes inside *Doctor* have confidentiality constraints on them. We have chosen principal-templates, to make *Patient*, *Symptom*, and *Payment* reusable, for example *Symptom*$\langle$*Lise*$\rangle$ would have an attribute *description* with the label {*Lise:*}.

In our case study the domain model provides deeper information about confidentiality issues than the use-case and activity diagram. But, it is still possible for a customer to consider the domain model. By using use-case diagrams, activity diagrams, and a domain model one can build up an understanding of the confidentiality issues with the system to be built.

4.5 The Interaction Diagram

Now we move from analysis to design. From the previous diagram we have obtained an informal understanding of how objects communicate confidential data. Here we will make this more precise with the help of a collaboration diagram.

In figure 6 we can see how *DataDoctor*, *Disease*, and *Symptom* collaborate to give information back to the patient, *Lise*, about her disease. From the numbers in front of the calls, we can see that the order of calls is: *getDisease*, *charge*, *match*, *equals*.

Let us consider *getDisease*. For each patient we want an instance of *DataDoctor* to be able to handle the call from the particular patient. This means that one instance of the *DataDoctor* only can handle one patient with a particular principal. As we can see from the diagram in figure 6, the call *getDisease* has the arguments $s : Symptom\langle\{Lise :; Doctor :\}\rangle$ and $p : Payment\langle Lise\rangle$. For the *DataDoctor* to handle this call it needs to know about the principal, *Lise*, which

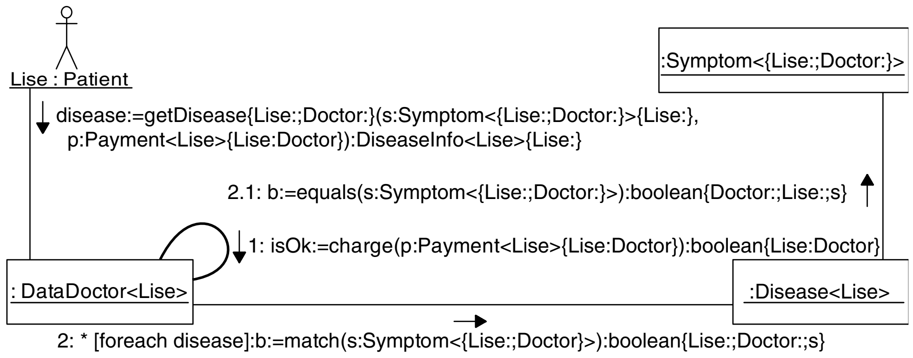

Lise : Patient
:Symptom<{Lise:;Doctor:}>
disease:=getDisease{Lise:;Doctor:}(s:Symptom<{Lise:;Doctor:}>{Lise:},
p:Payment<Lise>{Lise:Doctor}):DiseaseInfo<Lise>{Lise:}
2.1: b:=equals(s:Symptom<{Lise:;Doctor:}>):boolean{Doctor:;Lise:;s}
1: isOk:=charge(p:Payment<Lise>{Lise:Doctor}):boolean{Lise:Doctor}
: DataDoctor<Lise>
:Disease<Lise>
2: * [foreach disease]:b:=match(s:Symptom<{Lise:;Doctor}>):boolean{Lise:;Doctor:;s}

Fig. 6. Get disease collaboration diagra

is done through the template mechanism and therefore the $DataDoctor\langle Lise\rangle$ in the collaboration diagram. There are similar reasons for the $Disease\langle Lise\rangle$.

In our system we wanted to compare one *Symptom* owned by the principal *Doctor* and another owned by the patient, *Lise*. To simplify the comparison, due to Jif, we chose to make the *Symptom* owned by both principals — making it more confidential so it still satisfies the confidentiality constraint for the system. For this reason we changed *Symptom* into a class of template label instead of template principal, see figure 6. This is also the reason why the instance of the *Symptom* in *getDisease* has the label $\{Lise :; Doctor :\}$, but there is no need to make the reference more restrictive than $\{Lise:\}$

The instance disease of *DiseaseInfo* returned to *Lise* contains information about the treatment recommended for her illness. This data comes from information owned by both the patient, *Lise*, and the *Doctor*, the reason being that we need information from both the medical system and the patient to find the disease. This is done in the operation *match* which also uses *equals*. Therefore the *DataDoctor* needs to declassify the information so that the content can be read by *Lise*. In our case we found the need for declassification already in the activity diagram, see figure 4. We do not know if this is the case in general since the activity diagrams are often treated informal when used in the beginning of a process. In contrast to the the interaction diagram which by nature is more formal.

As we can see, collaboration diagrams are useful to show what confidential data flows from and to objects in the form of parameters and return types. These diagrams are often too detailed for a customer, but good for a designer moving one step closer to code.

4.6 The Class Diagram

We constructed the class diagram, figure 7, from the domain and the collaboration diagrams. As we can see from the collaboration diagram in figure 6 we need one more class, *DiseaseInfo*, and the template parameter of *Symptom* changed

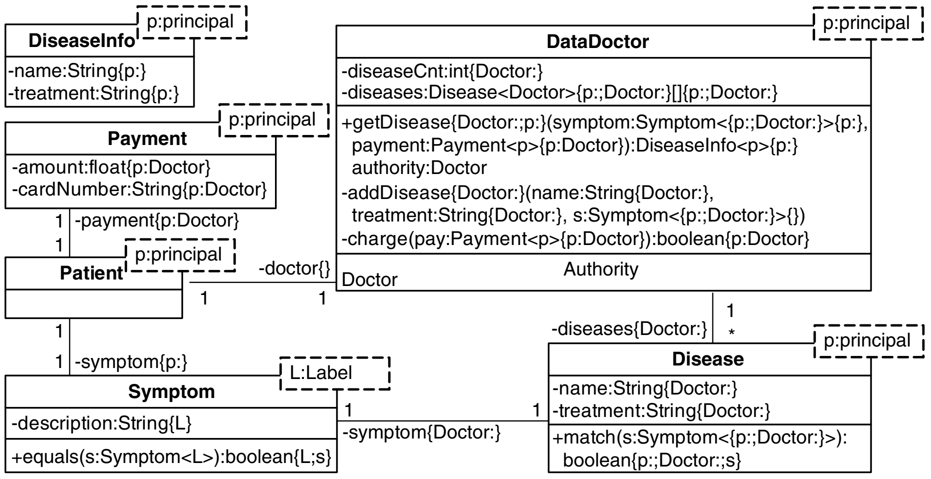

p:principal
DiseaseInfo
-name:String{p:}
-treatment:String{p:}
p:principal
Payment
-amount:float{p:Doctor}
-cardNumber:String{p:Doctor}
1
-payment{p:Doctor}
1
p:principal
Patient
-doctor{}
1
1
1
1
-symptom{p:}
L:Label
Symptom
-description:String{L}
+equals(s:Symptom<L>):boolean{L;s}
1
-symptom{Doctor:}
1
p:principal
DataDoctor
-diseaseCnt:int{Doctor:}
-diseases:Disease<Doctor>{p:;Doctor:}[]{p:;Doctor:}
+getDisease{Doctor:;p:}(symptom:Symptom<{p:;Doctor:}>{p:}, payment:Payment<p>{p:Doctor}):DiseaseInfo<p>{p:} authority:Doctor
-addDisease{Doctor:}(name:String{Doctor:}, treatment:String{Doctor:}, s:Symptom<{p:;Doctor:}>{})
-charge(pay:Payment<p>{p:Doctor}):boolean{p:Doctor}
Doctor
Authority
1
-diseases{Doctor:}
*
p:principal
Disease
-name:String{Doctor:}
-treatment:String{Doctor:}
+match(s:Symptom<{p:;Doctor:}>): boolean{p:;Doctor:;s}

Fig. 7. Class diagram

to a label. Furthermore, operations in *DataDoctor* and *Disease* require that we make them into template classes.

All the calls from the collaboration diagram are added as operations to the representing class in the class diagram plus one extra operation, *addDisease*. The reason we add the operation *addDisease* is that it shows one example of how to consider begin-labels during the creation of the class diagram. This operation has side-effects on the attribute *diseases* so it is natural to give it the begin-label $\{Doctor{:}\}$. The begin-label on *getDisease* was added for more technical reasons which came up during the implementation of the system.

From the collaboration diagram we know that the class *DataDoctor* needs to declassify information owned by *Doctor* and therefore needs the authority of the *Doctor*.

4.7 From UML to Jif

The diagram we translate into code is the class diagram. The information in the class diagram contains all the information needed to create the class skeleton containing the attributes and the method definitions (not the body). We have implemented the case study in Jif, see Appendix B for a code skeleton of *DataDoctor*. The translation from UMLS's class diagram to code skeleton is not hard, so we should be able to do this automatically.

After the code has been constructed the Jif compiler validates the confidentiality constraints. One of the major strengths of using UMLS for specification and Jif for validation is that the designer can consider security during the design phase, then gets an extra software validation step of the code which guarantees that indirect information flows [DD77] are not introduced.

4.8 Discussion

Each diagram is used to consider a different aspect of the confidentiality of the system. These diagrams are used to enhance the understanding of the system during the design phase much in the same way as the standard UML is used in the industry today.

The case study has been presented sequentially in this paper, however during the development we iterated several times through the different diagrams. The construction of the system was done by writing the diagrams on a white-board. This created a lot of discussion and several interesting confidentiality issues came up during this process. Coding confidential systems is a hard and error prone task. We feel that by using visual diagrams this task is simplified, and it is easy to invent a process to create code directly from the diagrams, making it easy for the developers to create large systems.

5 Related Work

To consider security in UML is a relatively new idea. Blobel, Pharow and Roger-France [BPRF99] used use cases to consider security in a very informal way in a medical setting. We find it very difficult to say anything about use cases since they are very informal and not very well understood semantically [GLQ02]. Furthermore, there has been work on developing a framework for model-based risk assessment of security-critical system using UML [HBLS02].

The connection between language-based security and security on the level of specifications has also been previously established by Mantel and Sabelfeld [MS01]. They have chosen a more theoretical approach than we have done. We hope that by choosing a more practical approach we will be able to reach more designers.

The research which is mostly related to ours is Jan Jürjens' work on modelling confidentiality in UML [Jür01b,Jür01a,Jür02]. Jürjens uses state-chart diagrams to handle confidentiality problems of a system. Being the first to consider confidentiality with UML it is only natural that his approach has several limitations. Firstly, the developer has to convince himself that the system is correct by examining the UML diagrams, which might be quite complex. Secondly, it is uncertain that the code created from these diagram is correct since that depends on how the code is created. Thirdly, the code is needed in order to find the covert channels. So, even if confidentiality properties are proved on the UML diagrams, which might be quite difficult in itself, there is no guarantee that the code correctly implements confidentiality constraints. All these problems are addressed by our approach.

One further problem is that Jürjens' work relies on a precise semantic definition of the state-chart diagram. Jürjens overcomes this problem by using a limited part of UML to which he gives his own semantics. Jürjens has moved towards using UML's extending mechanism for modelling confidentiality [Jür02], *stereotypes* and *tags* [OMG]. We have chosen not to do this, because we have not found any good way of expressing our extensions using stereotypes and tags that is as readable as our annotations.

There has been some work that considers role-based access control in a UML setting[ES99,LBD02]. Even though we have focused on information-flow, there are some interesting parallels to this research. UMLS/Jif permits declassification of data and this can perhaps be viewed as a form of access control.

6 Conclusion and Future Work

In this paper we have used a case study to demonstrate that our extensions to UML simplify the process of producing programs with confidentiality constraints/requirements. Furthermore, we have motivated the importance of using a language-based checker to validate the code. We believe that the combination of modelling confidentiality with a modelling language and validating the code with a language-based checker is crucial for building large applications that require a high degree of confidentiality.

The UML diagrams we have considered in this paper are, in our experience, often used in the development of object-oriented software which is the main reason behind our choice of diagram types. It would be interesting to look at state-chart diagrams as well, because state-chart diagrams can be used to generate additional code which considers confidentiality. Work done by Jürjens might be useful to consider here [Jür01b].

The main purpose of this paper is to show the powerful combination of a modelling language and a language-based checker. To take this research a step further requires more work on Jif, UMLS, and a tool to integrate them. Another interesting direction would be to see if there are other language-based checkers which also can be combined with UMLS or UML.

There is one area we have not addressed in this paper, but which is important for our work: secure environments. Here, the deployment diagram in UML might be very useful when specifying secure environments for Jif programs. This is also something we intend to study further.

Acknowledgement

Thanks to Andrew Myers, Stephen Chong, and Lantian Zheng at Cornell for enhancing our knowledge about the Jif environment, and for advising us in how to implement correct Jif code for our case-study. Furthermore, we want to thank Daniel Hedin for several important discussions about our research. We also want to thank colleagues at computing science department at Chalmers for comments on the paper, and the three anonymous referees.

References

BL73. D. Bell and L. LaPadula. Secure Computer Systems:Mathematical Foundations and Model. Technical Report MTR 2547 v2, The MITRE Corporation, Nov 1973.

BPRF99. B. Blobel, P. Pharow, and F. Roger-France. Security Analysis and Design Based on a General Conceptual Security Model and UML. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and B. Hertzberger, editors, *High-Performance Computing and Networking, 7th International Conference, HPCN Europe 1999, Amsterdam*, volume 1593 of *Lecture Notes in Computer Science*, pages 918–930. Springer, April 12-14 1999.

Coc01. Alistar Cockburn. *Writing Effective Use Cases.* Addison Wesley, 2001.

DD77. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 77.

ES99. P. Epstein and R. Sandhu. Towards A UML Based Approach to Role Engineering. In *RBAC '99, Proceedings of the Fourth ACM Workshop on Role-Based Access Control*, pages 135–143, October 28-29 1999.

GJS96. J. Gosling, B. Joy, and G. Steele. *The Java Language Specification.* Addison-Wesley, 1996.

GLQ02. G. Génova, J. Llorens, and V. Quintana. Digging into use case relationships. In J. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002*, volume 2460 of *LNCS*, pages 115–127. springer, September/October 2002.

HBLS02. S. H. Houmb, F. Braber, M. Soldal Lund, and K. Stolen. Towards a UML Profile for Model-Based Risk Assessment. In *Critical Systems Development with UML-Proceedings of of the UML'2 workshop*, pages 79–91, September 2002.

JBR99. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process.* Number ISBN 0-201-57169-2 in Object Technology. Addison-Wesley, 1999.

Jür01a. J. Jürjens. Secure Java Development with UMLsec. In B. De Decker, F. Piessens, J. Smits, and E. Van Herrenweghen, editors, *Advances in Network and Distributed Systems Security*, pages 107–124, Leuven (Belgium), November 26-27 2001. International Federation for Information Processing (IFIP) TC-11 WG 11.4, klu. Proceedings of the First Annual Working Conference on Network Security (I-NetSec '01).

Jür01b. J. Jürjens. Towards Development of Secure Systems using UMLsec. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering (FASE, 4th International Conference, Part of ETAPS)*, volume 2029, pages 187–200, 2001.

Jür02. J. Jürjens. UMLsec: Extending UML for Secure Systems Development. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 – The Unified Modeling Language*, volume 2460 of *lncs*, pages 412–425, Dresden, Sept. 30 – Oct. 4 2002. sv. 5th International Conference.

K.J77. K.J.Biba. Integrity consideration for secure computer system. Technical Report ESDTR-76-372,MTR-3153, The MITRE Corporation, Bedford,MA, April 1977.

Koz99. Dexter Kozen. Language-Based Security. In *Mathematical Foundations of Computer Science*, pages 284–298, 1999.

Lam73. Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, 1973.

LBD02. Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In Jean-Marc Jezequel, Heinrich Hussmann, and Stephen Cook, editors, *The unified modeling language: model engineering, concepts, and tools; 5th international*, volume 2460, pages 426–441. Springer, 2002.

ML97. Andrew C. Myers and Barbara Liskov. A Decentralized Model for Information Flow Control. In *Symposium on Operating Systems Principles*, pages 129–142, 1997.

ML98. Myers and Liskov. Complete, Safe Information Flow with Decentralized Labels. In *RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.

ML00. 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.

MS01. H. Mantel and A. Sabelfeld. A Generic Approach to the Security of Multi-Threaded Programs. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 126–142, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society Press.

Mye99a. A. Myers. Mostly-Static Decentralized Information Flow Control. Technical Report MIT/LCS/TR-783, MIT, 1999.

Mye99b. Andrew C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Symposium on Principles of Programming Languages*, pages 228–241, 1999.

OMG. OMG. *Unified Modeling Language Specification.*

RJB99. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Number ISBN 0-201-30998-X in Object Technology. Addison-Wesley, 1999.

SM03. A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

SMH01. Fred B. Schneider, Greg Morrisett, and Robert Harper. A Language-Based Approach to Security. *Lecture Notes in Computer Science*, 2000:86–101, August 2001.

VS00. Dennis M. Volpano and Geoffrey Smith. Verifying Secrets and Relative Secrecy. In *Symposium on Principles of Programming Languages*, pages 268–276, 2000.

A Jif Code of Vector

```
public class Vector[label L] {
    private int{L} length;
    private Object{L}[]{L} elements;

    public Vector{L}(){ resize(10); }

    public Object{L} elementAt(int i):{L;i}
        throws (ArrayIndexOutOfBoundsException){
            return elements[i];
    }

    public setElementAt{L}(Object{} o, int{} i) {
        if (i >= length)
            resize();       // make the array larger
        elements[i] = o;
    }

    public int{L} size(){ return length; }
    private void resize{L}(){...}
}
```

B Code Skeleton of DataDoctor

```
class DataDoctor[principal patient] authority(Doctor) {

    private Disease[patient]{Doctor:;patient:}[]{Doctor:;patient:} diseases;
    private int{Doctor:} diseaseCnt;

    public DiseaseInfo[patient]{patient:} getDisease{Doctor:;patient:}
     (Symptom[{patient:;Doctor:}] {patient:} s, Payment[patient]
     {patient:Doctor} payment) where authority(Doctor){...}

     public void addDisease{Doctor:} (String{Doctor:} name, String
      {Doctor:} treatment,Symptom[{Doctor:;patient:}] {} s) {...}

     private boolean{patient:Doctor}charge(Payment[patient]
      {patient:Doctor} pay){...}
}
```

An On-the-Fly Model-Checker for Security Protocol Analysis*

David Basin, Sebastian Mödersheim, and Luca Viganò

Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{basin,moedersheim,vigano}@inf.ethz.ch
www.infsec.ethz.ch/~{basin,moedersheim,vigano}

Abstract. We introduce the on-the-fly model-checker OFMC, a tool that combines two methods for analyzing security protocols. The first is the use of lazy data-types as a simple way of building an efficient on-the-fly model checker for protocols with infinite state spaces. The second is the integration of symbolic techniques for modeling a Dolev-Yao intruder, whose actions are generated in a demand-driven way. We present experiments that demonstrate that our tool is state-of-the-art, both in terms of coverage and performance, and that it scales well to industrial-strength protocols.

1 Introduction

A wide variety of model-checking approaches have recently been applied to analyzing security protocols, e.g. [1,7,13,22,23,25,26]. The key challenge they face is that the general security problem is undecidable [14], and even semi-algorithms, focused on falsification, must come to terms with the enormous branching factor in the search space resulting from using the standard Dolev-Yao intruder model, where the intruder can say infinitely many different things at any point.

In this paper, we show how to combine and extend different methods to build a highly effective security protocol model-checker. Our starting point is the approach of [4] of using *lazy data-types* to model the infinite state-space associated with a protocol. A lazy data-type is one where data-type constructors (e.g. *cons* for building lists, or *node* for building trees) build data-types without evaluating their arguments; this allows one to represent and compute with infinite data (e.g. streams or infinite trees), generating arbitrary prefixes of the data on demand. In [4], lazy data-types are used to build, and compute with, models of security protocols: a protocol and description of the powers of an intruder are formalized as an infinite tree. Lazy evaluation is used to decouple the model from search and heuristics, building the infinite tree on-the-fly, in a demand-driven fashion.

* This work was supported by the FET Open Project IST-2001-39252, "AVISPA: Automated Validation of Internet Security Protocols and Applications" [2].

This approach is conceptually and practically attractive as it cleanly separates model construction, search, and search reduction techniques. Unfortunately, it doesn't address the problem of the prolific Dolev-Yao intruder and hence scales poorly. We show how to incorporate the use of symbolic techniques to substantially reduce this problem. We formalize a technique that significantly reduces the search space without excluding any attacks. This technique, which we call the *lazy intruder technique*, uses a symbolic representation to avoid explicitly enumerating the possible messages the intruder can generate, by representing intruder messages using terms with variables, and storing and manipulating constraints about what must be generated and from which knowledge.

The lazy intruder is a general, technology-independent technique that can be effectively incorporated in different approaches to protocol analysis. Here, we show how to combine it with the lazy infinite-state approach to build a tool that scales well and has state-of-the-art coverage and performance. In doing so, we see our contributions as follows. First, we extend previous approaches, e.g. [17,7,1,22,16,8,10] to the symbolic representation of the intruder so that our lazy intruder technique is applicable to a larger class of protocols and properties. Second, despite the extensions, we simplify the technique, leading to a simpler proof of its correctness and completeness. Third, the lazy intruder introduces the need for constraint reduction and this introduces its own search space. We formalize the integration of the technique into the search procedure induced by the rewriting approach of our underlying protocol model, which provides an infinite-state transition system. On the practical side, we also investigate the question of an efficient implementation of the lazy intruder, i.e. how to organize state exploration and constraint reduction.

The result is OFMC, an on-the-fly model-checker for security protocol analysis. We have carried out a large number of experiments to validate our approach. For example, the OFMC tool finds all (but one) known attacks and discovers a new one (on the Yahalom protocol) in a test-suite of 36 protocols from the Clark/Jacob library [9] in under one minute of CPU time for the entire suite. Moreover, we have successfully applied OFMC to large-scale protocols including IKE, SET, and various other industrial protocols currently being standardized by the Internet Engineering Task Force IETF. As an example of industrial-scale problem, we describe in §5 our analysis of the H.530 protocol [18], a protocol invented by Siemens and proposed as an Internet standard for multimedia communications. We have modeled the protocol in its full complexity and have detected a replay attack in 1.6 seconds. The weakness is serious enough that Siemens has changed the protocol.

We proceed as follows. In §2 we give the formal model that we use for protocol analysis. In §3 we review the lazy protocol analysis approach. In §4 we formalize the lazy intruder and how constraints are reduced. We present experimental results in §5, and discuss related and future work in §6. Due to lack of space, examples and proofs have been shortened or omitted; details can be found in [5].

2 Protocol Specification Languages and Model

The formal model we use for protocol analysis with our tool OFMC is based on two specification languages, which we have been developing in the context of the AVISPA project [2]: a high-level protocol specification language (HLPSL) and a low-level one (the Intermediate Format IF). HLPSL allows the user to specify the protocols in an Alice & Bob style notation. A translator called HLPSL2IF automatically translates HLPSL specifications into the IF, which the OFMC tool takes as input. Due to space limitations and since the ideas behind our protocol specification languages are fairly standard, e.g. [11,19], we restrict ourselves here to the presentation of the IF; discussions and examples can be found in the technical report [5].

The Syntax of the IF. Let $\mathcal{C}$ and $\mathcal{V}$ be disjoint countable sets of *constants* (denoted by lower-case letters) and *variables* (denoted by upper-case letters). The *syntax* of the IF is defined by the following context-free grammar:

$$\begin{aligned}
\mathit{ProtocolDescr} &::= (\mathit{State}, \mathit{Rule}^*, \mathit{State}^*)\\
\mathit{Rule} &::= \mathit{State}\ \mathit{NegFacts} \Rightarrow \mathit{State}\\
\mathit{State} &::= \mathit{PosFact}\ (\ .\ \mathit{PosFact})^*\\
\mathit{NegFacts} &::= (\ .\ \mathsf{not}(\mathit{PosFact})\ )^*\\
\mathit{PosFact} &::= \mathsf{state}(\mathit{Msg}) \mid \mathsf{msg}(\mathit{Msg}) \mid \mathsf{i\_knows}(\mathit{Msg})\\
\mathit{Msg} &::= \mathit{AtomicMsg} \mid \mathit{ComposedMsg}\\
\mathit{ComposedMsg} &::= \langle \mathit{Msg},\mathit{Msg}\rangle \mid \{\!|\mathit{Msg}|\!\}_{\mathit{Msg}} \mid \{\mathit{Msg}\}_{\mathit{Msg}} \mid \mathit{Msg}^{-1}\\
\mathit{AtomicMsg} &::= \mathcal{C} \mid \mathcal{V} \mid \mathbb{N} \mid \mathsf{fresh}(\mathcal{C},\mathbb{N})
\end{aligned}$$

We write vars(t) to denote the set of variables occurring in a (message, fact, or state) *term* t, and say that t is *ground* when vars$(t) = \emptyset$.

An *atomic message* is a constant, a variable, a natural number, or a *fresh constant*. The fresh constants are used to model the creation of random data, like nonces, during protocol sessions. We model each fresh data item by a unique term $\mathsf{fresh}(C,N)$, where C is an identifier and the number N denotes the particular protocol session C is intended for.

Messages in the IF are either atomic messages or are composed using *pairing* $\langle M_1,M_2\rangle$, or the *cryptographic operators* $\{\!|M_1|\!\}_{M_2}$ and $\{M_1\}_{M_2}$ (for *symmetric* and *asymmetric encryption* of M_1 with M_2), or M^{-1} (the *asymmetric inverse* of M). Note that by default the IF is untyped (and the complexity of messages is not bounded), but it can also be generated in a typed variant, which leads to smaller search spaces at the cost of abstracting away any type-flaw attacks on the protocol.

Note also that we follow the standard *perfect cryptography assumption*, i.e. the only way to decrypt an encrypted message is to have the appropriate key. Moreover, like most other approaches, we here employ the *free algebra assumption* and assume that syntactically different terms represent different messages, facts, or states. In other words, we do not assume that algebraic equations hold on terms, e.g. that pairing is associative. Note too that unlike other models, e.g. [12,22], we are not bound to a fixed public-key infrastructure where every agent initially has a key-pair and knows the public key of every other agent. Rather, we can also consider protocols where keys are generated, distributed, and revoked.

The IF contains both positive and negative *facts*. A *positive fact* represents either the local state of an honest agent, a message on the network (i.e. one sent but not yet received), or that a message is known by the intruder. To ease the formalization of protocols, we introduce *negative facts* as well as additional fact symbols, expressing, e.g. secrecy or set membership; see [5] for details. To illustrate how these additions allow for the explicit encoding of problems in a natural way, consider the Needham-Schroeder public-key protocol with a key-server [9]. In a realistic model of this protocol, an agent should maintain a database of known public keys, which is shared over all protocol executions he participates in, and ask the key-server for the public key of another agent only if this key is not contained in his database. This situation can be directly modeled using negation and an additional fact symbol $\mathsf{knows\_pk}$.

Note also that the set of composed messages can similarly be easily extended, e.g. with cryptographic primitives for hashes and key-tables, without affecting the theoretical results we present below. In this paper, we focus on this smaller language for brevity.

A *state* is a set of positive facts, which we denote as a sequence of positive facts, separated by dots. Note that in our approach we actually employ set rewriting instead of multiset rewriting, as the HLPSL2IF translator ensures that in no reachable state the same positive fact can appear more than once, so we need not distinguish between multisets and sets.

We define in the usual way the notions related to substitution and unification, such as *ground term*, *ground substitution*, *unifier*, *most general unifier (mgu)*, and *matching*; see, e.g., [3]. We denote the application of a substitution σ to a term t by writing $t\sigma$ and denote the composition of substitutions σ_1 and σ_2 by writing $\sigma_1\sigma_2$. As we only consider substitutions with finite domains, we represent a substitution σ with $\mathrm{dom}(\sigma) = \{v_1, \ldots, v_n\}$ by $[v_1 \mapsto v_1\sigma, \ldots, v_n \mapsto v_n\sigma]$. The *identity substitution* id is the substitution with $\mathrm{dom}(\mathrm{id}) = \emptyset$. We say that two substitutions σ_1 and σ_2 are *compatible*, written $\sigma_1 \approx \sigma_2$, if $v\sigma_1 = v\sigma_2$ for every $v \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2)$. For two sets of ground substitutions Σ_1 and Σ_2, we define their *intersection modulo the different domains* as $\Sigma_1 \sqcap \Sigma_2 = \{\sigma_1\sigma_2 \mid \sigma_1 \in \Sigma_1 \ \wedge \ \sigma_2 \in \Sigma_2 \ \wedge \ \sigma_1 \approx \sigma_2\}$. Since the composition of compatible ground substitutions is associative and commutative, so is the $\sqcap$ operator.

A *protocol description ProtocolDescr* is a triple (I, R, G) consisting of an *initial state* I, a *set of rules* R, and a *set of goal states* G. A protocol description constitutes a *protocol* when two restrictions are met: (i) the initial state is ground, and (ii) $\mathrm{vars}(l_1) \supseteq \mathrm{vars}(l_2) \cup \mathrm{vars}(r)$ for every rule $l_1.l_2 \Rightarrow r$ in R, where l_1 contains only positive facts and l_2 contains only negative facts.

Rules describe state transitions. Intuitively, the application of a rule $l_1.l_2 \Rightarrow r$ means that if (i) there is a substitution σ such that no positive fact f with $\mathsf{not}(f) \in l_2$ can be matched under σ with the current state, and (ii) all positive facts in l_1 can be matched under σ with the current state, then $l_1\sigma$ is replaced by $r\sigma$ in the current state. Otherwise, the rule is not applicable.

In this paper, we consider only IF rules of the form

$$\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2\,, \tag{1}$$

where N_1 is a set of negative facts, and P_1 and P_2 are sets of positive facts that do not contain state or msg facts. Moreover, if $\mathsf{i\_knows}(m) \in P_1$ then $\mathsf{i\_knows}(m) \in P_2$, which ensures that the intruder knowledge is *monotonic*, i.e. that the intruder never forgets messages during a transition.

More specifically, every rule describes a transition of an honest agent, since a state fact appears on both the left-hand side (LHS) and the right-hand side (RHS) of the rule. Also, on both sides we have a msg fact representing an incoming message that the agent expects to receive in order to make the transition (LHS) and an answer message from the agent (RHS).

Rules of the form (1) are adequate to describe large classes of protocols (including those discussed in §5); see [5] for examples of actual IF rules.

The Dolev-Yao Intruder. We follow Dolev and Yao [12] and consider the standard, protocol-independent, asynchronous model in which the intruder controls the network but cannot break cryptography. In particular, he can intercept messages and analyze them if he possesses the corresponding decryption keys, and he can generate messages and send them under any agent name.

Definition 1. *For a set M of messages, let $\mathcal{DY}(M)$ (for Dolev-Yao) be the smallest set closed under the following* generation (G) *and* analysis (A) rules*:*

$$\frac{m \in M}{m \in \mathcal{DY}(M)}\,G_{\text{axiom}}, \quad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\langle m_1,m_2\rangle \in \mathcal{DY}(M)}\,G_{\text{pair}}, \quad \frac{\langle m_1,m_2\rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\,A_{\text{pair}_i},$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{|m_2|\}_{m_1} \in \mathcal{DY}(M)}\,G_{\text{scrypt}}, \quad \frac{\{|m|\}_k \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)}\,A_{\text{scrypt}}.$$

The generation rules express that the intruder can compose messages from known messages using pairing and symmetric encryption; the analysis rules describe how he can decompose messages. For brevity, we have omitted the rules for asymmetric encryption and decryption, which are straightforward. Note that this formalization correctly handles non-atomic keys, for instance $m \in \mathcal{DY}(\{\{|m|\}_{(\langle k_1,k_2\rangle)}, k_1, k_2\})$, as opposed to other models such as [1,20,24,26] that only handle atomic keys.

The Semantics of the IF. Using $\mathcal{DY}$, we now define the protocol model provided by the IF in terms of an infinite-state transition system, where the IF rules define a state-transition function. In this definition, we incorporate an optimization that we call *step-compression*, which is based on the idea [1,7,8,10,22] that we can identify the intruder and the network: every message sent by an honest agent is received by the intruder and every message received by an honest agent comes from the intruder. Formally, we compose (or "compress") several steps:

when the intruder sends a message, an agent reacts to it according to his rules, and the intruder diverts the agent's answer. A bisimulation proof shows that, for large classes of properties, the model with such composed transitions (which we present here) is "attack-equivalent" to the model with single (uncompressed) transitions, i.e. we end up in an attack state using composed transitions iff that was the case using uncomposed transitions.

Definition 2. *The* successor function $succ_R(S) = \bigcup_{r \in R} step_r(S)$ *maps a set of rules R and a state S to a set of states, where*

$$
\begin{aligned}
step_r(S) = \{S' \mid \exists\, \sigma.\, \mathrm{ground}(\sigma) \;\wedge\; \mathrm{dom}(\sigma) = \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) && (2)\\
\wedge\;\; m_1\sigma \in \mathcal{DY}(\{\, i \mid \mathsf{i\_knows}(i) \in S\}) && (3)\\
\wedge\;\; \mathsf{state}(m_2\sigma) \in S \;\wedge\; P_1\sigma \subseteq S \;\wedge\; \forall f.\, \mathsf{not}(f) \in N_1 \implies f\sigma \notin S && (4)\\
\wedge\;\; S' = (S \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma)) \cup \mathsf{state}(m_3\sigma) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma\} && (5)
\end{aligned}
$$

for a rule r of the form $\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2$

Here and elsewhere, we simplify notation for singleton sets by writing, e.g., $\mathsf{state}(m_2\sigma) \cup P_1\sigma$ *for* $\{\mathsf{state}(m_2\sigma)\} \cup P_1\sigma$*.*

The step function implements the step-compression technique in that it combines three transitions, based on a rule r of the form (1). The three transitions are: the intruder sends a message that is expected by an honest agent, the honest agent receives the message and sends a reply, and the intruder diverts this reply and adds it to his knowledge. More in detail, condition (3) ensures that the message $m_1\sigma$ (that is expected by the honest agent) can be generated from the intruder knowledge, where according to (2) σ is a ground substitution for the variables in the positive facts of the LHS of the rule r. The conjuncts (4) ensure that the other positive facts of the rule appear in the current state under σ and that none of the negative facts is contained in the current state under σ. Finally, (5) defines the successor state S' that results by removing from S the positive facts of the LHS of r and replacing them with the RHS of r (all under σ).

We define the *set of reachable states* associated with a protocol (I, R, G) as $reach(I, R) = \bigcup_{n \in \mathbb{N}} succ_R^n(I)$. The set of reachable states is ground as no state reachable from the initial state I may contain variables (by the conditions (i) and (ii) in the definition of protocol). As the properties we are interested in are reachability properties, we will sometimes abstract away the details of the transition system and refer to this set as the *ground model* of the protocol.

We say that a protocol is *secure* iff $goalcheck_g(S) = \emptyset$ for all $S \in reach(I, R)$ and all goals $g \in G$, where we define $goalcheck_g(S) = \{\sigma \mid g\sigma \subseteq S\}$.

3 The Lazy Infinite-State Approach

The transition system defines a (computation) tree in the standard way, where the root is the initial system state and children represent the ways that a state can evolve in one transition. The tree has infinitely many states since, by the definition of $\mathcal{DY}$, every node has infinitely many children. It is also of infinite depth, provided we do not bound the number of protocol sessions. The lazy intruder technique presented in the next section uses a symbolic representation to solve the problem of infinite branching, while the *lazy infinite-state approach* [4] allows us to handle the infinitely long branches. As we have integrated the lazy intruder with our previous work, we now briefly summarize the main ideas of [4].

The key idea behind the lazy infinite-state approach is to explicitly formalize an infinite tree as an element of a data-type in a lazy programming language. This yields a finite, computable representation of the model that can be used to generate arbitrary prefixes of the tree on-the-fly, i.e. in a demand-driven way. One can search for an attack by searching the infinite tree for a goal state. Our on-the-fly model-checker OFMC uses iterative deepening to search this infinite tree for an attack state. When an attack is found, OFMC returns the attack trace, i.e. the sequence of exchanged messages leading to the attack state. This yields a semi-decision procedure for protocol insecurity: our procedure always terminates (at least in principle) when an attack exists. Moreover, our search procedure terminates for finitely many sessions (formally: if there are finitely many agents and none of them can perform an unbounded number of transitions) when we employ the lazy intruder to restrict the infinite set of messages the intruder can generate.

The lazy approach has several strengths. It separates (both conceptually and structurally) the semantics of protocols from heuristics and other search reduction procedures, and from search itself. The semantics is given by a transition system generating an infinite tree, and heuristics can be seen as tree transducers that take an infinite tree and return one that is, in some way, smaller or more restricted. The resulting tree is then searched. Although semantics, heuristics, and search are all formulated independently, lazy evaluation serves to co-routine them together in an efficient, demand-driven fashion. Moreover, there are efficient compilers for lazy functional programming languages like Haskell, the language we used.

4 The Lazy Intruder

The *lazy intruder* is an optimization technique that significantly reduces the search tree without excluding any attacks. This technique uses a symbolic representation to avoid explicitly enumerating the possible messages that the Dolev-Yao intruder can generate, by storing and manipulating constraints about what must be generated. The representation is evaluated in a demand-driven way, hence the intruder is called *lazy*.

The idea behind the lazy intruder was, to our knowledge, first proposed by [17] and then subsequently developed by [7,1,22,16,8,10]. Our contributions to the symbolic intruder technique are as follows. First, we simplify the technique, which, as we formally show in [5], also leads to a simpler proof of its correctness and completeness. Second, we formalize its integration into the search procedure induced by the rewriting approach of the IF and, on the practical side, we present an efficient way to organize and implement the combination of state exploration and constraint reduction. Third, we extend the technique to ease the specification and analysis of a larger class of protocols and properties, where the introduction of negative facts alongside standard positive facts in the IF rewrite rules leads to inequality constraints in the lazy intruder.

Constraints. The Dolev-Yao intruder leads to an enormous branching of the search tree when one naïvely enumerates all (meaningful) messages that the intruder can send. The lazy intruder technique exploits the fact that the actual value of certain parts of a message is often irrelevant for the receiver. So, whenever the receiver will not further analyze the value of a particular message part, we can postpone during the search the decision about which value the intruder actually chooses for this part by replacing it with a variable and recording a constraint on which knowledge the intruder can use to generate the message. We express this information using constraints of the form $\mathit{from}(T, \mathit{IK})$, meaning that T is a set of terms generated by the intruder from his set of known messages IK (for "intruder knowledge").

Definition 3. *The* semantics of a constraint $\mathit{from}(T, \mathit{IK})$ *is the set of satisfying ground substitutions* σ *for the variables in the constraint, i.e.* $[\![\mathit{from}(T, \mathit{IK})]\!] = \{\sigma \mid \mathrm{ground}(\sigma) \wedge \mathrm{ground}(T\sigma \cup \mathit{IK}\sigma) \wedge (T\sigma \subseteq \mathcal{DY}(\mathit{IK}\sigma))\}$*. A* constraint set *is a finite set of constraints and its semantics is the intersection of the semantics of its elements, i.e., overloading notation,* $[\![\{c_1, \ldots, c_n\}]\!] = \sqcap_{i=1}^{n} [\![c_i]\!]$*. A constraint set* C *is* satisfiable *if* $[\![C]\!] \neq \emptyset$*. A constraint* $\mathit{from}(T, \mathit{IK})$ *is* simple *if* $T \subseteq \mathcal{V}$*, and we then write* $\mathrm{simple}(\mathit{from}(T, \mathit{IK}))$*. A constraint set* C *is simple if all its constraints are simple, and we then write* $\mathrm{simple}(C)$*.*

Constraint Reduction. The core of the lazy intruder technique is to reduce a given constraint set into an equivalent one that is either unsatisfiable or simple. (As we show in Lemma 2, every simple constraint set is satisfiable.) This reduction is performed using the generation and analysis rules of Fig. 1, which describe possible transformations of the constraint set (for brevity, we have again omitted the rules for asymmetric encryption and decryption, which are straightforward). Afterwards, we show that this reduction does not change the set of solutions, roughly speaking $[\![C]\!] = [\![\mathit{Red}(C)]\!]$, for a relevant class of constraints C.

The rules are of the form $\dfrac{C', \sigma'}{C, \sigma}$, with C and C' constraint sets and σ and σ' substitutions. They express that (C', σ') can be *derived* from (C, σ), which we denote by $(C, \sigma) \vdash (C', \sigma')$. Note that σ' extends σ in all rules. As a consequence, we can apply the substitutions generated during the reduction of C also to the facts of the lazy state.

The generation rules G^{l}_{pair} and G^{l}_{scrypt} express that the constraint stating that the intruder can generate a message composed from submessages m_1 and m_2 (using pairing and symmetric encryption, respectively) can be replaced by the constraint stating that he can generate both m_1 and m_2. The rule G^{l}_{unif}

$$\frac{\mathit{from}(m_1 \cup m_2 \cup T, \mathit{IK}) \cup C, \sigma}{\mathit{from}(\langle m_1,m_2\rangle \cup T, \mathit{IK}) \cup C, \sigma}\ G^l_{\text{pair}}\,,\quad \frac{\mathit{from}(m_1 \cup m_2 \cup T, \mathit{IK}) \cup C, \sigma}{\mathit{from}(\{\!|m_2|\!\}_{m_1} \cup T, \mathit{IK}) \cup C, \sigma}\ G^l_{\text{scrypt}}\,,$$

$$\frac{(\mathit{from}(T, m_2 \cup \mathit{IK}) \cup C)\tau, \sigma\tau}{\mathit{from}(m_1 \cup T, m_2 \cup \mathit{IK}) \cup C, \sigma}\ G^l_{\text{unif}}\ (\tau = \mathit{mgu}(m_1, m_2),\ m_1 \notin \mathcal{V})\,,$$

$$\frac{\mathit{from}(T, m_1 \cup m_2 \cup \langle m_1,m_2\rangle \cup \mathit{IK}) \cup C, \sigma}{\mathit{from}(T, \langle m_1,m_2\rangle \cup \mathit{IK}) \cup C, \sigma}\ A^l_{\text{pair}}\,,$$

$$\frac{\mathit{from}(k, \mathit{IK}) \cup \mathit{from}(T, m \cup \{\!|m|\!\}_k \cup \mathit{IK}) \cup C, \sigma}{\mathit{from}(T, \{\!|m|\!\}_k \cup \mathit{IK}) \cup C, \sigma}\ A^l_{\text{scrypt}}\,.$$

Fig. 1. Lazy intruder: constraint reduction rules

expresses that the intruder can use a message m_2 from his knowledge if this message can be unified with the message m_1 that he has to generate (note that both the terms to be generated and the terms in the intruder knowledge may contain variables). The reason that the intruder is "lazy" stems from the restriction that the G^l_{unif} rule cannot be applied when the term to be generated is a variable: the intruder's choice for this variable does not matter at this stage of the search and hence we postpone this decision.

The analysis of the intruder knowledge is more complex for the lazy intruder than in the ground model, as messages may now contain variables. In particular, if the key of an encrypted message is a variable, then whether or not the intruder can decrypt this message is determined by the substitution we (later) choose for this variable. We solve this problem by using the rule A^l_{scrypt}, where the variable key can be instantiated during further constraint reduction[1]. More specifically, for a message $\{\!|m|\!\}_k$ that the intruder attempts to decrypt, we add the content m to the intruder knowledge of the respective constraint (as if the check was already successful) and add a new constraint expressing that the symmetric key k necessary for decryption must be generated from the same knowledge. Hence, if we attempt to decrypt a message that cannot be decrypted using the corresponding intruder knowledge, we obtain an unsatisfiable constraint set.

Definition 4. *Let $\vdash$ denote the derivation relation described by the rules in Fig. 1. The set of pairs of simple constraint sets and substitutions derivable from* (C, id) *is* $\mathit{Red}(C) = \{(C', \sigma) \mid ((C, \mathrm{id}) \vdash (C', \sigma)) \wedge \mathrm{simple}(C')\}$.

Properties of *Red*. By Theorem 1 below, proved in [5], the *Red* function is correct, complete, and recursively computable (since $\vdash$ is finitely branching). To show completeness, we restrict our attention to a special form of constraint sets, called *well-formed constraint sets*. This is without loss of generality, as all states reachable in the lazy intruder setting obey this restriction (cf. Lemma 3).

[1] This solution also takes care of non-atomic keys since we do not require that the key is contained in the intruder knowledge but only that it can be generated from the intruder knowledge, e.g. by composing known messages.

Definition 5. *A constraint set C is* well-formed *if one can index the constraints, $C = \{from(T_1, IK_1), \ldots, from(T_n, IK_n\})$, so that the following conditions hold: (i) $IK_i \subseteq IK_j$ for $i \leq j$, and (ii)* $\mathrm{vars}(IK_i) \subseteq \cup_{j=1}^{i-1} \mathrm{vars}(T_j)$.

Intuitively, (i) requires that the intruder knowledge increases monotonically and (ii) requires that every variable that appears in intruder-known terms is part of a message that the intruder created earlier, i.e. variables only "originate" from the intruder.

Theorem 1. *Let C be a well-formed constraint set. $Red(C)$ is finite and $\vdash$ is well-founded. Moreover, $[\![C]\!] = [\![Red(C)]\!]$.*

The Lazy Intruder Reachability. We describe now the integration of constraint reduction into the search procedure for reachable states. The space of *lazy states* consists of states that may contain variable symbols (as opposed to the ground model where all reachable states are ground) and that are associated with a set of *from* constraints as well as a collection of inequalities. The inequalities will be used to handle negative facts in the context of the lazy intruder. We assume that the inequalities are given as a conjunction of disjunctions of inequalities between terms. We will use the inequalities to rule out certain unifications, e.g. to express that both the substitutions $\sigma = [v_1 \mapsto t_1, v_2 \mapsto t_2]$ and $\tau = [v_1 \mapsto t_3]$ are excluded in a certain state, we use the inequality constraint $(v_1 \neq t_1 \mathsf{V}\, v_2 \neq t_2) \;\mathsf{\Lambda}\; (v_1 \neq t_3)$, where we write V and $\mathsf{\Lambda}$ to avoid confusion with the respective meta-connectives $\vee$ and $\wedge$.

A lazy state represents the set of ground states that can be obtained by instantiating the variables with ground messages so that all associated constraints are satisfied.

Definition 6. *A* lazy state *is a triple (P, C, N), where P is a sequence of (not necessarily ground) positive facts, C is a constraint set, and N is a conjunction of disjunctions of inequalities between terms. The semantics of a lazy state is $[\![(P, C, N)]\!] = \{P\sigma \mid \sigma \in [\![C]\!] \wedge \sigma \models N\}$, where $\sigma \models N$ is defined for a substitution σ as expected.*

Let $\mathrm{freshvars}_r(S)$ *be a renaming of the variables in a lazy state $S = (P, C, N)$ with respect to a rule r such that* $\mathrm{vars}(\mathrm{freshvars}_r(S))$ *and* $\mathrm{vars}(r)$ *are disjoint. The* lazy successor function $lsucc_R(S) = \cup_{r \in R}\, lstep_r(\mathrm{freshvars}_r(S))$ *maps a set of rules R and a lazy state $S = (P, C, N)$ to a set of lazy states, where*

$$
\begin{aligned}
& lstep_r(P, C, N) = \{(P', C', N') \mid \exists\, \sigma. \\
& \quad \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \cup \mathrm{vars}(P) \cup \mathrm{vars}(C) \cup \mathrm{vars}(N) \\
& \quad \wedge\; C' = (C \cup from(m_1, \{i \mid \mathsf{i\_knows}(i) \in P\}))\sigma && (6) \\
& \quad \wedge\; \mathsf{state}(m_2\sigma) \in P\sigma \;\; \wedge P_1\sigma \subseteq P\sigma && (7) \\
& \quad \wedge\; N' = N \,\mathsf{\Lambda} \textstyle\bigwedge_{\phi \in subCont(N_1\sigma, P\sigma)} \phi && (8)
\end{aligned}
$$

$$\wedge\ P' = (P\sigma \setminus (\mathsf{state}(m_2\sigma) \cup P_1\sigma)) \\ \qquad\qquad \cup\ \mathsf{state}(m_3\sigma) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma\} \qquad (9)$$

for a rule r of the form $\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2$, where $subCont(N,P) = \{\phi \mid \exists\ t, t', \sigma.\ \mathsf{not}(t) \in N\ \wedge\ t' \in P\ \wedge\ t\sigma = t'\sigma$

$$\wedge\ \exists\ v_1, \ldots, v_n, t_1, \ldots, t_n.\ \sigma = [v_1 \mapsto t_1, \ldots, v_n \mapsto t_n]\ \wedge\ \phi = \bigvee_{i=1}^{n} v_i \neq t_i\}$$

Similar to the successor function of the ground model, the lazy successor function also performs step-compression, i.e. it performs three operations in one transition: the intruder sends a message, an honest agent reacts to it, and the intruder adds the answer to his knowledge. The most notable change is the renaming of the variables of the rules to avoid clashes with the variables that may appear in the lazy states. More in detail, the constraint in condition (6) expresses that the message m_1 that occurs on the LHS of the rule r must be generated by the intruder from his current knowledge. Condition (7) is similar to the first two conjuncts in condition (4) in the ground model, where the substitution is now applied also to the set of positive facts in the state (i.e., instead of matching, we now perform unification). Condition (8) states that the inequalities are conjoined with the conjunction of all formulae that $subCont(N_1\sigma, P\sigma)$ yields. For a set of negative facts N and a set of positive facts P, $subCont(N,P)$ generates a disjunction of inequalities that excludes all unifiers between two positive facts t and t' such that $\mathsf{not}(t) \in N$ and $t' \in P$. Note that in the special case that $t = t'$ we obtain the solution $\sigma = []$, and naturally we define $\bigvee_{i=1}^{0} \phi$ to be simply false for any ϕ. Finally, condition (9) describes the positive facts P' in the successor state, which result by removing the positive LHS facts from P (under σ) and adding the RHS facts (under σ).

We define the set of *reachable lazy states* associated to a protocol (I, R, G) as $lreach(I, R) = \bigcup_{n\in\mathbb{N}} lsucc_R^n(I, \emptyset, \emptyset)$. We also call $lreach(I, R)$ the *lazy model* of the protocol (I, R, G). The lazy model is equivalent to the ground model, i.e. they represent the same set of reachable states.

Lemma 1. *$reach(I, R) = \cup_{(P,C,N)\in lreach(I,R)} [\![(P, C, N)]\!]$ for every initial state I and every set R of rules of the form* (1).

Recall that we have defined that a protocol is secure iff *goalcheck* (which represents the negation of the property the protocol aims to provide, i.e. it represents the attacks on the protocol) is empty for all reachable ground states. A similar check suffices in the lazy intruder model. We define the lazy goal-check for a lazy state $S = (P, C, N)$ and a goal state g as $lgoalcheck_g(P, C, N) = \{\sigma \mid g\sigma \subseteq P\sigma\}$. If *lgoalcheck* is not empty in a reachable lazy state S, then either S represents an attack or S is *unsatisfiable*, i.e. its semantics is the empty set.

Theorem 2. *A protocol (I, R, G) is* secure *iff $\sigma \in lgoalcheck_g(P, C, N)$ implies $[\![(P, C, N)\sigma]\!] = \emptyset$ for all $(P, C, N) \in lreach(I, R)$ and all $g \in G$.*

Using the above results, we now show how we can build an effective semi-decision procedure for protocol insecurity based on the lazy intruder. (In the case of a bounded number of sessions, our procedure is actually a decision procedure.) To this end, we have to tackle three problems.

First, the *lstep* function yields in general infinitely many successors, as there can be infinitely many unifiers σ for the positive facts of the rules and the current state. However, as we follow the free algebra assumption on the message terms, two unifiable terms always have a unique *mgu*, and we can, without loss of generality, focus on that unifier. (Note also that there are always finitely many *mgu*'s as the set of rules is finite and a lazy state contains finitely many facts.)

Second, we must represent the reachable states. The lazy infinite-state approach provides a straightforward solution to this problem, where we represent the reachable states as the tree generated using the lazy intruder successor function. (For an unbounded number of sessions, this tree is infinitely deep.) We can apply the lazy goal-check as a filter on this tree to obtain the lazy goal states.

Third, we must check whether one of these lazy goal states is satisfiable, i.e. represents a possible attack. (We will see that this check can also be applied as a filter on the tree.) The constraint reduction is the key to achieve this task. By Theorem 1, we know that, for a well-formed constraint set C, the reduction produces a set of simple constraint sets that together have the same semantics as C. The following lemma shows that a lazy state with a simple constraint set and a satisfiable collection of inequalities is always satisfiable.

Lemma 2. *Let (P, C, N) be a lazy state where C is simple and N is satisfiable (i.e. $\exists\sigma.\ \sigma \models N$). Then $[\![(P, C, N)]\!] \neq \emptyset$.*

The proof, given in [5], is based on the observation that a simple constraint set with inequalities is always satisfiable as the intruder can always generate sufficiently many different messages. This is the key idea behind inequalities in our lazy model.

From this lemma we can conclude the following for a well-formed constraint set C and a collection of inequalities N. If there is at least one solution $(C', \tau) \in \mathit{Red}(C)$ and $N\tau$ is satisfiable, then $[\![(P, N, C)]\!] \neq \emptyset$, since C' is simple and $[\![C']\!] \subseteq [\![C]\!]$, by Theorem 1. Otherwise, if $\mathit{Red}(C) = \emptyset$ or if N is unsatisfiable, then $[\![(P, C, N)]\!] = \emptyset$, also by Theorem 1.

So, for a reachable lazy state (P, C, N) we can decide if $[\![(P, C, N)]\!]$ is empty, as long as C is well-formed. To obtain simple constraint sets, we call *Red*, which only applies to well-formed constraint sets. It thus remains to show that all constraint sets of reachable lazy states are well-formed, which follows from the way new constraints are generated during the *lstep* transitions.

Lemma 3. *For a protocol (I, R, G), if $(P, C, N) \in \mathit{lreach}_R(I)$ then C is well-formed.*

We have now put all pieces together to obtain an effective procedure for checking whether a protocol is secure: we generate reachable lazy states and filter them both for goal states and for constraint satisfiability. We now briefly discuss how to implement this procedure in an efficient way.

Organizing State Exploration and Constraint Reduction. When implementing the lazy intruder we are faced with two design decisions: (i) in which order the two “filters” mentioned above are applied, and (ii) how constraint reduction should be realized.

With respect to (i), note that the definition of reachable lazy states does not prescribe when *Red* should be called; *Red* is only used to determine if a constraint set is satisfiable. In OFMC we apply *Red* after each transition to check if the constraints are still satisfiable. This allows us to eliminate from the search all states with unsatisfiable constraint sets, as the successors of such states will again have unsatisfiable constraint sets. We also extend this idea to checking the inequalities and remove states with unsatisfiable inequalities. In the lazy infinite-state approach this can be realized simply by swapping the order in which the “filters” are applied, i.e. the tree of reachable lazy states is first filtered for satisfiable lazy states (using *Red*), thereby pruning several subtrees, and then for goal states (using *lgoalcheck*). Note that *Red* can lead to case splits if there are several solutions for the given constraint set; in this case, to avoid additional branching of the search tree, we continue the search with the original constraint set.

With respect to (ii), note that the question of how to compute the constraint reduction (in particular, how to analyze the intruder knowledge) is often neglected in other presentations of symbolic intruder approaches. One solution is to proceed on demand: a message in the intruder knowledge is analyzed iff the result of this analysis can be unified with a message the intruder has to generate. We adopt a more efficient solution. We apply the analysis rules to every constraint as long as they are applicable. The result is that the intruder knowledge is “normalized” with respect to the analysis rules. As a consequence, we need not further consider analysis rules during the reduction of the constraints. This has the advantage that to check if the G^l_{unif} rule is applicable to a message m that the intruder has to generate, we must simply check if in the (analyzed) intruder knowledge some message m' appears that can be unified with m. In contrast, with analysis on demand it is in this case necessary to check if a unifiable message may be obtained through analysis.

However, when normalizing the intruder knowledge, we must take into account that the analysis may lead to substitutions. Every substitution restricts the set of possible solutions and in this case the restriction is only necessary if the respective decrypted content of the message is actually used later (which is, in contrast, elegantly handled by applying the analysis on demand)[2]. Our solution to this problem is to distinguish between analysis steps that require a substitu-

[2] As an example, suppose that the intruder wants to analyze the message $\{|\{|\mathsf{m}|\}_{\mathsf{k}}|\}_{\{|\mathsf{M}|\}_{\mathsf{k}}}$, where the variable M represents a message the intruder generated earlier, and that he already knows the message $\{|\mathsf{m}|\}_{\mathsf{k}}$. Obviously the new constraint expressing that the key-term can be derived from the rest of the knowledge, $\mathit{from}(\{|\mathsf{M}|\}_{\mathsf{k}}, \{|\mathsf{m}|\}_{\mathsf{k}})$, is satisfiable, unifying $\mathsf{M} = \mathsf{m}$. The point is that the result of the decryption does not give the intruder any new information (he already knows $\{|\mathsf{m}|\}_{\mathsf{k}}$), hence by unifying $\mathsf{M} = \mathsf{m}$ we unnecessarily limit the possible messages the intruder could have said.

tion and those that do not. The latter can be performed without restriction, the former are not performed; rather, we add to the intruder knowledge the term that would be obtained in case of a successful analysis (without unification), and *mark* the term to express that the intruder *may* know it, but only under a certain substitution. If a marked term is actually needed, then the respective analysis steps are performed, else the marked term stays in the intruder knowledge.

Our strategy, which is a combination of demand-driven analysis and normalization, is somewhat complex but very efficient, as the occurrence of marked terms is rare and the use of a marked term is even rarer.

5 Experimental Results

To assess the effectiveness and performance of OFMC, we have tested it on a large protocol suite, which includes the protocols of the Clark/Jacob library [9,13], as well as a number of industrial-scale protocols. Since OFMC implements a semi-decision procedure, it does not terminate for correct protocols, although it can establish the correctness of protocols for a bounded number of sessions. We give below search times for finding attacks on flawed protocols.

The Clark/Jacob Library. The OFMC can find an attack for 32 of the 33 flawed protocols of the Clark/Jacob library[3]. As the times in Table 1 show, OFMC is a state-of-the-art tool: for each of the flawed protocols, a flaw is found in under 4 seconds and the total analysis of all flawed protocols takes less than one minute of CPU time. (Times are obtained on a PC with a 1.4GHz Pentium III processor and 512Mb of RAM, but note that, due to the use of iterative deepening search, OFMC requires a negligible amount of memory.) To our knowledge, no other tool for finding attacks is this fast and has comparable coverage.

Note that the analysis of the untyped and typed IF specifications may lead to the detection of different kinds of attacks. When this is the case, in Table 1 we report the two attacks found. (In all other cases, the times are obtained using the default untyped model.) Also note that the table contains four variants of protocols in the library, marked with a "*", that we have additionally analyzed, and that "MITM" abbreviates man-in-the-middle attack and "STS" abbreviates replay attack based on a short-term secret. Table 1 also reports a new attack that we have found on the Yahalom protocol, which we describe in [5].

The H.530 Protocol. We have applied OFMC to a number of industrial-scale protocols, such as IKE (for which we found the weaknesses already reported in [21]), and in particular, the H.530 protocol of the ITU [18]. H.530, which has been developed by Siemens, provides mutual authentication and key agreement in mobile roaming scenarios in multimedia communication.

[3] Missing is the CCITT X.509 protocol, where agents may sign messages they cannot analyze completely. OFMC cannot find this attack since the HLPSL does not (yet) allow us to specify an appropriate goal that is violated by this weakness.

Table 1. Performance of OFMC over the Clark/Jacob library

Protocol Name	Attack	Time
ISO symm. key 1-pass unilateral auth.	Replay	0.0
ISO symm. key 2-pass mutual auth.	Replay	0.0
Andrew Secure RPC prot.	Type flaw	0.0
	Replay	0.1
ISO CCF 1-pass unilateral auth.	Replay	0.0
ISO CCF 2-pass mutual auth.	Replay	0.0
Needham-Schroeder Conventional Key	STS	0.3
Denning-Sacco (symmetric)	Type flaw	0.0
Otway-Rees	Type flaw	0.0
Wide Mouthed Frog	Parallel-session	0.0
Yahalom	Type flaw	0.0
Woo-Lam Π_1	Type flaw	0.0
Woo-Lam Π_2	Type flaw	0.0
Woo-Lam Π_3	Type flaw	0.0
Woo-Lam Π	Parallel-session	0.2
Woo-Lam Mutual auth.	Parallel-session	0.3
Needham-Schroeder Signature prot.	MITM	0.1
* *Neuman Stubblebine initial part*	Type flaw	0.0
* *Neuman Stubblebine rep. part*	STS	0.0
Neuman Stubblebine (complete)	Type flaw	0.0
Kehne Langendorfer Schoenw. (rep. part)	Parallel-session	0.2
Kao Chow rep. auth., 1	STS	0.5
Kao Chow rep. auth., 2	STS	0.5
Kao Chow rep. auth., 3	STS	0.5
ISO public key 1-pass unilateral auth.	Replay	0.0
ISO public key 2-pass unilateral auth.	Replay	0.0
* *Needham-Schroeder Public Key NSPK*	MITM	0.0
NSPK with key server	MITM	1.1
* *NSPK with Lowe's fix*	Type flaw	0.0
SPLICE/AS auth. prot.	Replay	4.0
Hwang and Chen's modified SPLICE	MITM	0.0
Denning Sacco Key Distr. with Public Key	MITM	0.5
Shamir Rivest Adelman Three Pass prot.	Type flaw	0.0
Encrypted Key Exchange	Parallel-session	0.1
Davis Swick Private Key Certificates	Type flaw	0.1
(DSPKC), prot. 1	Replay	1.2
DSPKC, prot. 2	Type flaw	0.2
	Replay	0.9
DSPKC, prot. 3	Replay	0.0
DSPKC, prot. 4	Replay	0.0

H.530 is deployed as shown in the left part of Fig. 2: a mobile terminal (MT) wants to establish a secure connection and negotiate a Diffie-Hellman key with the gatekeeper (VGK) of a visited domain. As they do not know each other in advance, the authentication is performed using an authentication facility AuF within the home domain of the MT; both MT and VGK initially have shared keys with AuF. The right part of Fig. 2 shows the messages exchanged: first, both MT and VGK create Diffie-Hellman half-keys, along with hashes that are encrypted for the AuF (denoted by the messages Req_{MT} and Req_{VGK}, respectively). After a successful check of these messages, AuF replies with appropriate acknowledge messages Ack_{MT} and Ack_{VGK} that also contain encrypted hashes for the respective recipients. Finally, MT and VGK perform a mutual challenge-response using the new Diffie-Hellman key that was authenticated by AuF (directly or over a chain of trustworthy servers).

We have applied OFMC to automatically analyze the H.530 protocol in collaboration with Siemens. The main problem that we had to tackle for this analysis is the fact that the protocol employs the Diffie-Hellman key-agreement, which is based on a property of cryptographic algorithms (namely the commutativity of exponents) that violates the free algebra assumption. We lack space here to discuss in detail how we solved this problem in our model. The central point is this: while the messages exchanged in the H.530 protocol are considerably more complex than the ones of the Clark/Jacob protocols, this complexity is not a problem for our approach, unlike for other model-checking tools, e.g. [13,20]. We could directly analyze the original specification of the H.530 without first simplifying the messages.

OFMC takes only 1.6 seconds to detect a previously unknown attack to H.530. It is a replay attack where the intruder first listens to a session between honest agents *mt* in role MT, *vgk* in role VGK, and *auf* in role AuF. Then the intruder starts a new session impersonating both *mt* and *auf*. The weakness that makes the replay possible is the lack of fresh information in the message Ack_{VGK}, i.e. the message where *auf* acknowledges to *vgk* that he is actually

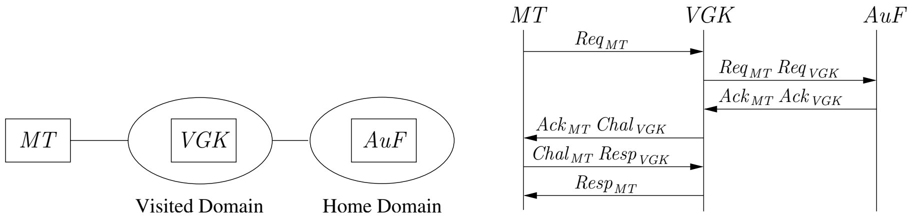


Fig. 2. The H.530 protocol (simplified). The deployment of the protocol is on the left, the messages exchange between the participants are summarized are on the right.

talking with *mt*. Replaying the respective message from the first session, the intruder impersonating *mt* can negotiate a new Diffie-Hellman key with *vgk*, "hijacking" *mt*'s identity. To perform the attack, the intruder must at least be able to eavesdrop and insert messages both on the connection between MT and VGK, and on the connection between VGK and AuF. We have suggested including MT's Diffie-Hellman half-key in the encrypted hash of the message Ack_{VGK} to fix this problem. With this extension we have not found any further weaknesses of the protocol and Siemens has changed the protocol accordingly.

6 Related Work and Concluding Remarks

There are several model-checking approaches similar to ours. As a prominent example, we compare our approach with Casper [13,20], a compiler that maps protocol specifications, written in a high-level language similar to HLPSL (which was inspired by CAPSL [11]), into descriptions in the process algebra CSP. The approach uses finite-state model-checking with FDR2. Casper/FDR2 has successfully discovered flaws in a wide range of protocols: among the protocols of the Clark/Jacob library, it has found attacks on 20 protocols previously known to be insecure, as well as attacks on 10 other protocols originally reported as secure. Experiments indicate that OFMC is considerably faster than Casper/FDR2, despite being based on a more general model: Casper limits the size of messages to obtain a finite-state model. This limitation is problematic for the detection of type-flaw attacks, e.g. Casper/FDR also misses our type-flaw attack on Yahalom (cf. [5]). Finally, Casper does not support non-atomic keys, which hinders its application to protocols like IKE, where each participant constructs only a part of the shared key that is negotiated.

The Athena tool [26] combines model-checking and interactive theorem-proving techniques with strand spaces [15] to reduce the search space and automatically prove the correctness of protocols with arbitrary numbers of concurrent runs. Interactive theorem-proving in this setting allows one to limit the search space by manually proving lemmata (e.g. "the intruder cannot find out a certain key, as it is never transmitted"). However, the amount of user interaction necessary to obtain such statements can be considerable. Moreover, like Casper/FDR2, Athena supports only atomic keys and cannot detect type flaws.

To compare with related approaches to symbolically modeling intruder actions, we expand on the remarks in §4. The idea of a symbolic intruder model has undergone a steady evolution, becoming increasingly simpler and general. In the earliest work [17], both the technique itself and the proof were of substantial complexity. In [1,7], both based on process calculi, the technique and its formal presentation were considerably simplified. [22] generalized the approach to support non-atomic symmetric keys, while [10] improved the approach of [22] by increasing its expressiveness and providing a more efficient implementation. [8] lifted the restriction of a fixed public-key infrastructure, where every agent has a fixed key-pair and knows his own private key and each agent's public key. This work is the closest to ours. Both approaches are based on HLPSL/IF and (multi)set rewriting. However, there are important differences. For instance, we have removed all procedural aspects from the symbolic intruder rules, making the approach more declarative and the proofs simpler. Moreover, we have extended the lazy intruder by introducing inequalities, which, together with the notion of simple constraints, has a very natural interpretation: "the intruder can generate as many different terms as he likes".

As we have seen, most approaches are restricted to atomic keys. This prevents the modeling of many modern protocols like IKE. Moreover, untyped protocol models with atomic keys exclude type-flaw attacks where keys are confused with composed terms. We believe that this is why our type-flaw attack on the Yahalom protocol was not discovered earlier, even though Yahalom has been extensively studied.

To summarize, we have presented an approach to security protocol analysis implemented by the model-checker OFMC, which represents a substantial development of the idea of on-the-fly model-checking proposed in [4]. The original tool required the use of heuristics and, even then, did not scale to most of the protocols in the Clark/Jacob library. The use of the symbolic techniques described here has made an improvement of many orders of magnitude and the techniques are so effective that heuristics play no role in the current system. Moreover, OFMC scales well beyond the Clark/Jacob protocols, as our example of the H.530 suggests. Current work involves applying OFMC to other industrial-scale protocols, such as those proposed by the IETF. Although initial experience is positive, we see an eventual role for heuristics in leading to further improvements. For example, a simple evaluation function could be: "has the intruder learned anything new through this step, and how interesting is what he learned?" We have also been investigating the integration of partial-order reduction techniques in our model-checker and the first results are very positive [6].

References

1. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. Concur'00*, LNCS 1877, pp. 380–394. Springer, 2000.
2. AVISPA: Automated Validation of Internet Security Protocols and Applications. FET Open Project IST-2001-39252, `www.avispa-project.org`.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge U. Pr., 1998.

4. D. Basin. Lazy infinite-state analysis of security protocols. In *Proc. CQRE'99*, LNCS 1740, pp. 30–42. Springer, 1999.
5. D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis (Extended Version). Technical Report 404, ETH Zurich, Computer Science, 2003. `http://www.inf.ethz.ch/research/publications/`.
6. D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. Technical Report 405, ETH Zurich, Computer Science, 2003. `http://www.inf.ethz.ch/research/publications/`.
7. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. ICALP'01*, LNCS 2076, pp. 667–681. Springer, 2001.
8. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proc. ASE'01*. IEEE Computer Society Press, 2001.
9. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
10. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proc. SAS 2002*, LNCS 2477, pp. 326–341. Springer, 2002.
11. G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, 2000.
12. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
13. B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. FMSP'99 (Formal Methods and Security Protocols)*.
14. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proc. FMSP'99 (Formal Methods and Security Protocols)*.
15. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
16. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. CSFW'01*. IEEE Computer Society Press, 2001.
17. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.
18. ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services). 2002.
19. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. LPAR 2000*, LNCS 1955, pp. 131–160. Springer, 2000.
20. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.
21. C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proc. 1999 IEEE Symposium on Security and Privacy.*
22. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. CCS'01*, pp. 166–175, ACM Press, 2001.
23. J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proc. 1997 IEEE Symposium on Security and Privacy.*
24. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
25. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. Modelling and Analysis of Security Protocols. Addison Wesley, 2000.
26. D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

Symmetric Authentication within a Simulatable Cryptographic Library

Michael Backes, Birgit Pfitzmann, and Michael Waidner

IBM Zurich Research Lab
{mbc,bpf,wmi}@zurich.ibm.com

Abstract. Proofs of security protocols typically employ simple abstractions of cryptographic operations, so that large parts of such proofs are independent of cryptographic details. The typical abstraction is the Dolev-Yao model, which treats cryptographic operations as a specific term algebra. However, there is no cryptographic semantics, i.e., no theorem that says what a proof with the Dolev-Yao abstraction implies for the real protocol, even if provably secure cryptographic primitives are used.
Recently we introduced an extension to the Dolev-Yao model for which such a cryptographic semantics exists, i.e., where security is preserved if the abstractions are instantiated with provably secure cryptographic primitives. Only asymmetric operations (digital signatures and public-key encryption) are considered so far. Here we extend this model to include a first symmetric primitive, message authentication, and prove that the extended model still has all desired properties. The proof is a combination of a probabilistic, imperfect bisimulation with cryptographic reductions and static information-flow analysis.
Considering symmetric primitives adds a major complication to the original model: we must deal with the exchange of secret keys, which might happen any time before or after the keys have been used for the first time. Without symmetric primitives only public keys need to be exchanged.

1 Introduction

Proofs of security protocols typically employ simple abstractions of cryptographic operations, so that large parts of such proofs are independent of cryptographic details, such as polynomial-time restrictions, probabilistic behavior and error probabilities. This is particularly true for tool-supported proofs, e.g., [17,16,13,22,23,1,15,18].

The typical abstraction is the Dolev-Yao model [9]: Cryptographic operations, e.g., E for encryption and D for decryption, are considered as operators in a term algebra where only certain cancellation rules hold. (In other words, one considers the initial model of an equational specification.) For instance, encrypting a message m twice does not yield another message from the basic message space but the term $\mathsf{E}(\mathsf{E}(m))$. A typical cancellation rule is $\mathsf{D}(\mathsf{E}(m)) = m$ for all m.

However, there is no cryptographic semantics, i.e., no theorem that says what a proof with the Dolev-Yao abstraction implies for the real protocol, even if provably secure cryptographic primitives are used. In fact, one can construct protocols that are secure in the Dolev-Yao model, but become insecure if implemented with certain provably secure cryptographic primitives [19]. Closing this gap has motivated a considerable amount of research in security and cryptography over the past few years, e.g., [14,20,3,12,21,8,5].

The abstraction we introduced in [5] achieved a major step towards closing this gap: We defined an ideal "cryptographic library" that offers abstract commands for generating nonces and keys, for performing operations (signing, testing, encrypting, decrypting) with these keys on messages, for dealing with lists and arbitrary application data, and for sending and receiving messages. The library further supports nested operations in the intuitive sense. The ideal cryptographic library has a simple deterministic behavior, and cryptographic objects are hidden at the interface, which makes it suitable as a basis for formal protocol verification. While the original Dolev-Yao model was a pure, memoryless algebra, our model is stateful, e.g., to distinguish different nonces and to reflect that cryptographically secure encryption and signature schemes are typically probabilistic. Thus our ideal cryptographic library corresponds more to "the CSP Dolev-Yao model" or "the Strand-space Dolev-Yao model" than the pure algebraic Dolev-Yao model.

This ideal cryptographic library is implemented by a real cryptographic library where the commands are implemented by real cryptographic algorithms and messages are actually sent between machines. The real system can be based on any cryptographically secure primitives. Our definition of security is based on the simulatability approach: security essentially means that anything an adversary against the real system can achieve can also be achieved by an adversary against the ideal system. This is the strongest possible cryptographic relation between a real and an ideal system. The definition in particular covers active attacks. In [5], our ideal and real cryptographic libraries were shown to fulfill this definition. The general composition theorem for the underlying model [21] implies that if a system that uses the abstract, ideal cryptographic library is secure then the same system using the real cryptographic library is also secure.

Only asymmetric cryptographic primitives (public-key encryption, digital signatures) are considered in [5], i.e., all primitives based on shared secret keys were not included. The main contribution of this paper is to add an important symmetric primitive to the framework of [5]: message authentication. We present abstractions for commands and data related to message authentication, e.g., commands for key generation, authentication, and authenticator testing, and we present a concrete realization based on cryptographic primitives. We then show that these two systems fulfill the simulatability definition if they are plugged into the existing cryptographic library. The inclusion of symmetric primitives and the sending of secret keys add a major complication to the original proof, because keys may be sent any time before or after the keys have been used for the first time. In particular, this implies that a real adversary can send messages which cannot immediately be simulated by a known term, because the keys needed to test the validity of authenticators are not yet known, but may be sent later by the adversary. Without symmetric primitives only public keys had to be exchanged, and the problem could be avoided by appropriate tagging of all real messages with the public keys used in them, so that messages could be immediately classified into correct terms or a specific garbage type [5]. Due to lack of space, this paper only briefly sketches the proof; the complete proof can be found in the full version of this paper [4].

Related Work. Abadi et. al. [3,2] started to bridge the abstraction gap by considering a Dolev-Yao model with nested algorithms specifically for symmetric encryption and synchronous protocols. There, however, the adversary is restricted to passive eavesdropping. Consequently, it was not necessary to choose a reactive model of a system and its honest users, and the security goals were all formulated as indistinguishability, i.e., if two abstract systems are indistinguishable by passive adversaries, then this is also true for the two corresponding real systems. This model does not yet contain theorems about composition or property preservation from the abstract to the real system. The price we pay for the greater applicability of reactive simulatability and for allowing active attacks is a much more complicated proof.

Several papers extended this work for specific models of specific classes of protocols. For instance, [11] specifically considers strand spaces, and within this model information-theoretically secure primitives.

The recent definitions of simulatability for reactive systems come with more or less worked-out examples of abstractions of cryptographic systems; however, even with a composition theorem this does not automatically give a cryptographic library in the Dolev-Yao sense, i.e., with the possibility to nest abstract operations. For instance, the abstract secure channels in [21] combine encryption and signatures in a fixed way, while the lower-level encryption subsystem used in that paper, like the examples in [14], does not offer abstract, implementation-independent outputs. The cryptographic primitives in [7,8] are abstract, but do not support nested operations: ideal cryptographic operations are defined through immediate interactions with the adversary, i.e., they are not local to the party that performs them and the adversary learns the structure of every term any honest party ever builds. The ideal system for signatures even reveals every signed message to the adversary. Thus, by composing cryptographic operations already the ideal systems reveal too much information to the adversary; thus they cannot be a sound basis for more complex protocols.

Our cryptographic library overcomes these problems. It supports nested operations in the intuitive sense; operations that are performed locally are not visible to the adversary. It is secure against arbitrary active attacks, and the composition theorem allows for safely using it within the context of arbitrary surrounding interactive protocols. This holds independently of the goals that one wants to prove about the surrounding protocols.

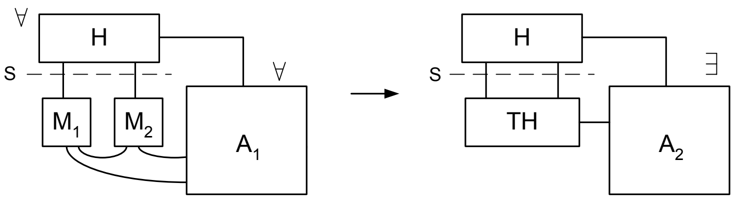


Fig. 1. Overview of the simulatability definition. A real system is shown on the left-hand side, and an ideal system on the right-hand side. The view of H must be indistinguishable.

2 Overview of Simulatability

We start with a brief overview of the underlying security notion of simulatability, which is the basic notion for comparing an ideal and a real system. For the moment, we only need to know that an ideal and a real system each consist of several possible structures, typically derived from an intended structure with a trust model. An intended structure represents a benign world, where each user is honest and each machine behaves as specified. The trust model is then used to determine the potentially malicious machines, i.e., machines which are considered to be under control of the adversary. Moreover, the trust model classifies the "security" of each connection between machines of the structure, e.g., that a connection is authentic, but not secret. Now for each element of the trust model, this gives one separate structure.

Each structure interacts with an adversary A and honest users summarized as a single machine H. The security definition is that for all polynomial-time honest users H and all polynomial-time adversaries A_1 on the real system, there exists an adversary A_2 on the ideal system such that the honest users H cannot notice the difference, as shown in Figure 1.

3 The Ideal System

For modeling and proving cryptographic protocols using our abstraction, it is sufficient to understand and use the ideal cryptographic library described in this section. Thus, applying our results to the verification of cryptographic protocols does not presuppose specific knowledge about cryptography or probabilism. The subsequent sections only justify the cryptographic faithfulness of this ideal library.

The ideal cryptographic library offers its users abstract cryptographic operations, such as commands to encrypt or decrypt a message, to make or test a signature, and to generate a nonce. All these commands have a simple, deterministic behavior in the ideal system. In a reactive scenario, this semantics is based on state, e.g., of who already knows which terms. We store state in a "database". Each entry of the database has a type (e.g., "signature"), and pointers to its arguments (e.g., a key and a message). This corresponds to the top level of a Dolev-Yao term; an entire term can be found by following the pointers. Further, each entry contains handles for those participants who already know it. The reason for using handles to make an entry accessible for higher protocols is that an idealized cryptographic term and the corresponding real message have to be presented in the same way to higher protocols to allow for a provably secure implementation in the sense of simulatability. In the ideal library, handles essentially point to Dolev-Yao-like terms, while in the real library they point to cryptographic messages.

The ideal cryptographic library does not allow cheating by construction. For instance, if it receives a command to encrypt a message m with a certain key, it simply makes an abstract database entry for the ciphertext. Another user can only ask for decryption of this ciphertext if he has handles to both the ciphertext and the secret key. Similarly, if a user issues a command to sign a message, the ideal system looks up whether this user should have the secret key. If yes, it stores that this message has been signed with this key. Later tests are simply look-ups in this database. A send operation makes an entry known to other participants, i.e., it adds handles to the entry. Our model does not only cover crypto operations, but it is an entire reactive system and therefore contains an abstract network model.

In the following, we present our additions to this ideal system for capturing symmetric authentication primitives, i.e., providing abstractions from authentication keys and authenticators, and offering commands for key generation, authentication, and verification. Both authenticators and authentication keys can be included into messages that are sent over the network, which allows for sharing authentication keys with other participants. Before we introduce our additions in detail, we explain the major design decisions. For understanding these decision, it might be helpful for readers not familiar with message authentication to read Section 4.1 before, which contains the cryptographic definition of secure authentication schemes.

First, we have to allow for checking if authenticators have been created with the same secret key; as the definition of secure authentication schemes does not exclude this, it can happen in the real system. For public-key encryption and digital signatures, this was achieved in [5] by tagging ciphertexts respectively signatures with the corresponding public key, so that the public keys can be compared. For authenticators, this is clearly not possible as no public key exists there. We solve this problem by tagging authenticators with an “empty” public key, which serves as a key identifier for the secret key.

Secondly, as authentication keys can be exchanged between the users and the adversary, it might happen that an authenticator is valid with respect to several authentication keys, e.g., because the adversary has created a suitable key. Hence it must be possible to tag authenticators with additional key identifiers during the execution, i.e., authenticators are tagged with a list of key identifiers. This list can also be empty which models authenticators from the adversary for which no suitable key is known yet.

Thirdly, we have to reflect the special capabilities an adversary has in the real system. For example, he might be able to transform an authenticator, i.e., to create a new authenticator for a message for which the correct user has already created another authenticator. Such a transformation is not excluded in the definition of secure authentication schemes, hence it might happen in the real system. The ideal library therefore offers special commands for the adversary to model capabilities of this kind.

3.1 Notation

We write "$:=$" for deterministic and "$\leftarrow$" for probabilistic assignment, and "$\xleftarrow{\mathcal{R}}$" for uniform random choice from a set. By $x := y{+}{+}$ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $|m|$, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The list operation is denoted as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$. A database D is a set of functions, called entries, each over a finite domain called attributes. For an entry $x \in D$, the value at an attribute att is written $x.att$. For a predicate *pred* involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill *pred*. If $D[pred]$ contains only one element, we use the same notation for this element. Adding an entry x to D is abbreviated $D :\Leftarrow x$.

3.2 Structures and Parameters

The ideal system consists of a trusted host $\mathsf{TH}_{\mathcal{H}}$ for every subset $\mathcal{H}$ of a set $\{1, \ldots, n\}$ of users, denoting the possible honest users. It has a port $\mathsf{in}_u?$ for inputs from and a port $\mathsf{out}_u!$ for outputs to each user $u \in \mathcal{H}$ and for $u = \mathsf{a}$, denoting the adversary.

The ideal system keeps track of the length of messages using a tuple L of abstract length functions. We add functions $\mathsf{ska\_len}^*(k)$ and $\mathsf{aut\_len}^*(k, l)$ to L for the length of authentication keys and authenticators, depending on a security parameter k and the length l of the message. Each function has to be polynomially bounded and efficiently computable.

3.3 States

The state of $\mathsf{TH}_{\mathcal{H}}$ consists of a database D and variables *size*, $curhnd_u$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$. The database D contains abstractions from real cryptographic objects which correspond to the top levels of Dolev-Yao terms. An entry has the following attributes:

- $x.ind \in \mathcal{INDS}$, called index, consecutively numbers all entries in D. The set $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$; we use it to distinguish index arguments from others. We use the index as a primary key attribute of the database, i.e., we write $D[i]$ for the selection $D[ind = i]$.

- $x.type \in typeset$ identifies the type of x. We add types ska, pka, and aut to *typeset* from [5], denoting secret authentication keys, "empty" public keys that are needed as key identifier for the corresponding authentication keys, and authenticators.
- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Many values a_i are indices of other entries in D and thus in $\mathcal{INDS}$. We sometimes distinguish them by a superscript "ind".
- $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ are handles by which a user or adversary u knows this entry. $x.hnd_u = \downarrow$ means that u does not know this entry. The set $\mathcal{HNDS}$ is yet another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $x.len \in \mathbb{N}_0$ denotes the "length" of the entry, which is computed by applying the functions from L.

Initially, D is empty. $\mathsf{TH}_\mathcal{H}$ has a counter $size \in \mathcal{INDS}$ for the current number of elements in D. New entries always receive $ind := size$++, and $x.ind$ is never changed. For the handle attributes, it has counters $curhnd_u$ (current handle) initialized with 0, and each new handle for u will be chosen as $i^{\mathsf{hnd}} := curhnd$++.

For each input port $\mathsf{p}?$, $\mathsf{TH}_\mathcal{H}$ further maintains a counter $steps_{\mathsf{p}?} \in \mathbb{N}_0$ initialized with 0 for the number of inputs at that port, each with a bound $bound_{\mathsf{p}?}$. If that bound is reached, no further inputs are accepted at that port, which is used to achieve polynomial runtime of the machine $\mathsf{TH}_\mathcal{H}$ independent of the environment. The underlying IO automata model guarantees that a machine can enforce such bounds without additional Turing steps even if the adversary tries to send more data. The bounds from [5] can be adopted without modification except that the number of permitted inputs from the adversary has to be enlarged. This is just a technical detail to allow for a correct proof of simulatability. We omit further details and refer to the full version [4].

3.4 New Inputs and Their Evaluation

The ideal system has several types of inputs: Basic commands are accepted at all ports $\mathsf{in}_u?$; they correspond to cryptographic operations and have only local effects, i.e., only an output at the port $\mathsf{out}_u?$ for the same user occurs and only handles for u are involved. Local adversary commands are of the same type, but only accepted at $\mathsf{in_a}?$; they model tolerated imperfections, i.e., possibilities that an adversary may have, but honest users do not. Send commands output values to other users. In the following, the notation $j \leftarrow \mathsf{algo}(i)$ for a command algo of $\mathsf{TH}_\mathcal{H}$ means that $\mathsf{TH}_\mathcal{H}$ receives an input $\mathsf{algo}(i)$ and outputs j if the input and output port are clear from the context. We only allow lists to be authenticated and transferred, because the list-operation is a convenient place to concentrate all verifications that no secret keys of the public-key systems from [5] are put into messages.

For dealing with symmetric authentication we have to add new basic commands and local adversary commands; the send commands are unchanged. We now define the precise new inputs and how $\mathsf{TH}_\mathcal{H}$ evaluates them based on its abstract state. Handle arguments are tacitly required to be in $\mathcal{HNDS}$ and existing, i.e., $\leq curhnd_u$, at the time of execution. The underlying model further bounds the length of each input to ensure polynomial runtime; these bounds are not written out explicitly, but can easily be derived from the domain expectations given for the individual inputs.

The algorithm $i^{\mathsf{hnd}} \leftarrow \mathsf{ind2hnd}_u(i)$ (with side effect) denotes that $\mathsf{TH}_{\mathcal{H}}$ determines a handle i^{hnd} for user u to an entry $D[i]$: If $i^{\mathsf{hnd}} := D[i].hnd_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].hnd_u := curhnd_u$++. On non-handles, it is the identity function. $\mathsf{ind2hnd}_u^*$ applies $\mathsf{ind2hnd}_u$ to each element of a list.

Basic Commands. First we consider basic commands. This comprises operations for key generation, creating and verifying an authenticator, and extracting the message from an authenticator. We assume the current input is made at port $\mathsf{in}_u?$, and the result goes to $\mathsf{out}_u!$.

– *Key generation:* $ska^{\mathsf{hnd}} \leftarrow \mathsf{gen\_auth\_key}()$. Set $ska^{\mathsf{hnd}} := curhnd_u$++ and

$$\begin{aligned}D &:\Leftarrow (ind := size\text{++}, type := \mathsf{pka}, arg := (), len := 0);\\ D &:\Leftarrow (ind := size\text{++}, type := \mathsf{ska}, arg := (ind-1),\\ &\qquad hnd_u := ska^{\mathsf{hnd}}, len := \mathsf{ska\_len}^*(k)).\end{aligned}$$

The first entry, an "empty" public key without handle, serves as the mentioned key identifier for the secret key. Note that the argument of the secret key "points" to the empty public key.

– *Authenticator generation:* $aut^{\mathsf{hnd}} \leftarrow \mathsf{auth}(ska^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
Let $ska := D[hnd_u = ska^{\mathsf{hnd}} \wedge type = \mathsf{ska}].ind$ and $l := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. Return $\downarrow$ if either of these is $\downarrow$, or if $length := \mathsf{aut\_len}^*(k, D[l].len) > \mathsf{max\_len}(k)$. Otherwise, set $aut^{\mathsf{hnd}} := curhnd_u$++, $pka := ska - 1$ and

$$\begin{aligned}D :\Leftarrow (&ind := size\text{++}, type := \mathsf{aut}, arg := (l, pka),\\ &hnd_u := aut^{\mathsf{hnd}}, len := length).\end{aligned}$$

The general argument format for entries of type aut is $(l, pka_1, \ldots, pka_j)$. The arguments $pka_1, \ldots, pka_j$ are the key identifiers of those secret keys for which this authenticator is valid. We will see in Section 3.4 that additional key identifiers for an authenticator can be added during the execution, e.g., because the adversary has created a suitable key. Such arguments are appended at the end of the existing list. An empty sequence of arguments pka_i models authenticators from the adversary for which no suitable secret key has been received yet.

– *Authenticator verification:* $v \leftarrow \mathsf{auth\_test}(aut^{\mathsf{hnd}}, ska^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
If $aut := D[hnd_u = aut^{\mathsf{hnd}} \wedge type = \mathsf{aut}].ind = \downarrow$ or $ska := D[hnd_u = ska^{\mathsf{hnd}} \wedge type = \mathsf{ska}].ind = \downarrow$, return $\downarrow$. Otherwise, let $(l, pka_1, \ldots, pka_j) := D[aut].arg$. If $ska - 1 \notin \{pka_1, \ldots, pka_j\}$ or $D[l].hnd_u \neq l^{\mathsf{hnd}}$, then $v := \mathsf{false}$, else $v := \mathsf{true}$.

The test $ska - 1 \in \{pka_1, \ldots, pka_j\}$ is the lookup that the secret key is one of those for which this authenticator is valid, i.e., that the cryptographic test would be successful in the real system.

– *Message retrieval:* $l^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_aut}(aut^{\mathsf{hnd}})$.
Let $l := D[hnd_u = aut^{\mathsf{hnd}} \wedge type = \mathsf{aut}].arg[1]$ and return $l^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(l)$ [1].

Local Adversary Commands. The following local commands are only accepted at the port $\mathsf{in_a}?$. They model special capabilities of the adversary. This comprises *authentication transformation*, which allows the adversary to create a new authenticator for a message provided that the adversary already has a handle for another authenticator for the same message. This capability has to be included in order to be securely realizable by cryptographic primitives, since the security definition of authentication schemes does not exclude such a transformation.

If an authenticator is received from the adversary for which no suitable key has been received yet, we call the authenticator (temporarily) unknown. In the real system, this means that no user will be able to check the validity of the authenticator. In the ideal system, this is modeled by providing a command for generating an *unknown authenticator*. Such an authenticator can become valid if a suitable secret key is received. A command for *fixing authenticators* takes care of this. Finally, we allow the adversary to retrieve all information that we do not explicitly require to be hidden, e.g., arguments and the type of a given handle. This is dealt with by a command for *parameter retrieval*.

– *Authentication transformation:* $trans\_aut^{\mathsf{hnd}} \leftarrow \mathsf{adv\_transform\_aut}(aut^{\mathsf{hnd}})$.
Return $\downarrow$ if $aut := D[hnd_{\mathsf{a}} = aut^{\mathsf{hnd}} \wedge type = \mathsf{aut}].ind = \downarrow$. Otherwise let $(l, pka_1, \ldots, pka_j) := D[aut].arg$, set $trans\_aut^{\mathsf{hnd}} := curhnd_{\mathsf{a}}{+}{+}$ and

$$D :\Leftarrow (ind := size{+}{+}, type := \mathsf{aut}, arg := (l, pka_1), \\ hnd_{\mathsf{a}} := trans\_aut^{\mathsf{hnd}}, len := D[aut].len).$$

– *Unknown authenticator:* $aut^{\mathsf{hnd}} \leftarrow \mathsf{adv\_unknown\_aut}(l^{\mathsf{hnd}})$.
Return $\downarrow$ if $l := D[hnd_{\mathsf{a}} = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind = \downarrow$ or $length := \mathsf{aut\_len}^*(k, D[l].len) > \mathsf{max\_len}(k)$. Otherwise, set $aut^{\mathsf{hnd}} := curhnd_a{+}{+}$ and

$$D :\Leftarrow (ind := size{+}{+}, type := \mathsf{aut}, arg := (l), hnd_{\mathsf{a}} := aut^{\mathsf{hnd}}, len := length).$$

Note that no key identifier exists for this authenticator yet.

– *Fixing authenticator:* $v \leftarrow \mathsf{adv\_fix\_aut\_validity}(ska^{\mathsf{hnd}}, aut^{\mathsf{hnd}})$.
Return $\downarrow$ if $aut := D[hnd_{\mathsf{a}} = aut^{\mathsf{hnd}} \wedge type = \mathsf{aut}].ind = \downarrow$ or if $ska := D[hnd_u = ska^{\mathsf{hnd}} \wedge type = \mathsf{ska}].ind = \downarrow$. Let $(l, pka_1, \ldots, pka_j) := D[aut].arg$ and $pka := ska - 1$. If $pka \notin \{pka_1, \ldots, pka_j\}$ set $D[aut].arg := (l, pka_1, \ldots, pka_j, pka)$ and output $v := \mathsf{true}$. Otherwise, output $v := \mathsf{false}$.

[1] This command implies that real authenticators must contain the message. The simulator in the proof needs this to translate authenticators from the adversary into abstract ones. Thus we also offer message retrieval to honest users so that they need not send the message separately.

– *Parameter retrieval:* $(type, arg) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$.
  This existing command always sets $type := D[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].type$, and for most types $arg := \mathsf{ind2hnd}_{\mathsf{a}}^{*}(D[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].arg)$. This applies to the new types pka, ska, and aut.

Note that for authenticators, a handle to the "empty" public key is output in adv_parse. If the adversary wants to know whether two given authenticators have been created using the same secret key, it simply parses them yielding handles to the corresponding "empty" public keys, and compares these handles.

Send Commands. The ideal cryptographic library offers commands for virtually sending messages to other users. Sending is modeled by adding new handles for the intended recipient and possibly the adversary to the database entry modeling the message. These handles always point to a list entry, which can contain arbitrary application data, ciphertexts, public keys, etc., and now also authenticators and authentication keys. These commands are unchanged from [5]; as an example we present those modeling insecure channels, which are the most commonly used ones, and omit secure channels and authentic channels.

– $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$, for $v \in \{1, \ldots, n\}$. Intuitively, the list l shall be sent to user v. Let $l^{\mathsf{ind}} := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$, then output $(u, v, \mathsf{ind2hnd}_{\mathsf{a}}(l^{\mathsf{ind}}))$ at $\mathsf{out_a}!$.
– $\mathsf{adv\_send\_i}(u, v, l^{\mathsf{hnd}})$, for $u \in \{1, \ldots, n\}$ and $v \in \mathcal{H}$ at port $\mathsf{in_a}?$. Intuitively, the adversary wants to send list l to v, pretending to be u. Let $l^{\mathsf{ind}} := D[hnd_{\mathsf{a}} = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$ output $(u, v, \mathsf{ind2hnd}_v(l^{\mathsf{ind}}))$ at $\mathsf{out}_v!$.

4 Real System

The real cryptographic library offers its users the same commands as the ideal one, i.e., honest users operate on cryptographic objects via handles. There is one separate database for each honest user in the real system, each database contains real cryptographic keys, real authenticators, etc., and real bitstrings are actually sent between machines. The commands are implemented by real cryptographic algorithms, and the simulatability proof will show that nevertheless, everything a real adversary can achieve can also be achieved by an adversary in the ideal system, or otherwise the underlying cryptography can be broken. We now present our additions and modifications to the real system, starting with a description of the underlying algorithms for key generation, authentication, and authenticator testing.

4.1 Cryptographic Operations

We denote a memoryless symmetric authentication scheme by a tuple $\mathcal{A} = (\mathsf{gen_A}, \mathsf{auth}, \mathsf{atest}, \mathsf{ska\_len}, \mathsf{aut\_len})$ of polynomial-time algorithms. For authentication key generation for a security parameter $k \in \mathbb{N}$, we write

$$sk \leftarrow \mathsf{gen_A}(1^k).$$

The length of sk is $\mathsf{ska\_len}(k) > 0$. By

$$aut \leftarrow \mathsf{auth}_{sk}(m)$$

we denote the (probabilistic) authentication of a message $m \in \{0,1\}^+$. Verification

$$b := \mathsf{atest}_{sk}(aut, m)$$

is deterministic and returns true (then we say that the authenticator is valid) or false. Correctly generated authenticators for keys of the correct length must always be valid. The length of aut is $\mathsf{aut\_len}(k, |m|) > 0$. This is also true for every aut' with $\mathsf{atest}_{sk}(aut', m) = \mathsf{true}$ for a value $sk \in \{0,1\}^{\mathsf{ska\_len}(k)}$. The functions $\mathsf{ska\_len}$ and $\mathsf{aut\_len}$ must be bounded by multivariate polynomials.

As the security definition we use security against existential forgery under adaptive chosen-message attacks similar to [10]. We only use our notation for interacting machines, and we allow that also the test function is adaptively attacked.

Definition 1. *(Authentication Security) Given an authentication scheme, an* authentication machine Aut *has one input and one output port, a variable sk initialized with* $\downarrow$*, and the following transition rules:*

- *First generate a key as* $sk \leftarrow \mathsf{gen_A}(1^k)$.
- *On input* (auth, m_j)*, return* $aut_j \leftarrow \mathsf{auth}_{sk}(m_j)$.
- *On input* $(\mathsf{test}, aut', m')$*, return* $v := \mathsf{atest}_{sk}(aut', m')$.

The authentication scheme is called existentially unforgeable under adaptive chosen-message attack *if for every probabilistic polynomial-time machine* $\mathsf{A_{aut}}$ *that interacts with* Aut*, the probability is negligible (in k) that* Aut *outputs* $v = \mathsf{true}$ *on any input* $(\mathsf{test}, aut', m')$ *where* m' *was not authenticated until that time, i.e., not among the* m_j*'s.* ◇

Note that the definition does not exclude authenticator transformation, i.e., if a message m_i has been properly authenticated, creating another valid authenticator for m_i is not excluded. This is why we introduced the command $\mathsf{adv\_transform\_aut}$ as a tolerable imperfection in Section 3.4. A well-known example of an authentication scheme that is provably secure under this definition is HMAC [6].

4.2 Structures

The intended structure of the real cryptographic library consists of n machines $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$. Each M_u has ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$, so that the same honest users can connect to the ideal and the real system. Each M_u has connections to each M_v exactly as in [5], in particular an insecure connection called $\mathsf{net}_{u,v,\mathsf{i}}$ for normal use. They are called network connections and the corresponding ports network ports. Any subset $\mathcal{H}$ of $\{1,\dots,n\}$ can denote the indices of correct machines. The resulting actual structure consists of the correct machines with modified channels according to a channel model. In particular, an insecure channel is split in the actual structure so that both machines actually interact with the adversary. Details of the channel model are not needed here. Such a structure then interacts with honest users H and an adversary A.

4.3 Lengths and Bounds

In the real system, we have length functions $\mathsf{list\_len}$, $\mathsf{nonce\_len}$, $\mathsf{ska\_len}$, and $\mathsf{aut\_len}$, corresponding to the length of lists, nonces, authentication keys, and authenticators, respectively. These functions can be arbitrary polynomials. For given functions $\mathsf{list\_len}$, $\mathsf{nonce\_len}$, $\mathsf{ska\_len}$, and $\mathsf{aut\_len}$, the corresponding ideal length functions are computed as follows:

– $\mathsf{ska\_len}^*(k) := \mathsf{list\_len}(|\mathsf{ska}|, \mathsf{ska\_len}(k), \mathsf{nonce\_len}(k))$; this must be bounded by $\mathsf{max\_len}(k)$;
– $\mathsf{aut\_len}'(k,l) := \mathsf{aut\_len}(k, \mathsf{list\_len}(\mathsf{nonce\_len}(k), l))$;
– $\mathsf{aut\_len}^*(k,l) := \mathsf{list\_len}(|\mathsf{aut}|, \mathsf{nonce\_len}(k), \mathsf{nonce\_len}(k), l, \mathsf{aut\_len}'(k,l))$.

4.4 States of a Machine

The state of each machine M_u consists of a database D_u and variables $curhnd_u$ and $steps_{\mathsf{p?}}$ for each input port $\mathsf{p?}$. Each entry x in D_u has the following attributes:

– $x.hnd_u \in \mathcal{HNDS}$ consecutively numbers all entries in D_u. We use it as a primary key attribute, i.e., we write $D_u[i^{\mathsf{hnd}}]$ for the selection $D_u[hnd_u = i^{\mathsf{hnd}}]$.
– $x.word \in \{0,1\}^+$ is the real representation of x.
– $x.type \in typeset \cup \{\mathsf{null}\}$ identifies the type of x. The value null denotes that the entry has not yet been parsed.
– $x.add\_arg$ is a list of ("additional") arguments. For entries of our new types it is always ().

Initially, D_u is empty. M_u has a counter $curhnd_u \in \mathcal{HNDS}$ for the current size of D_u. The subroutine

$$(i^{\mathsf{hnd}}, D_u) :\leftarrow (i, type, add\_arg)$$

determines a handle for certain given parameters in D_u: If an entry with the word i already exists, i.e., $i^{\mathsf{hnd}} := D_u[word = i \wedge type \notin \{\mathsf{sks}, \mathsf{ske}\}].hnd_u \neq \downarrow$ [2], it

[2] The restriction $type \notin \{\mathsf{sks}, \mathsf{ske}\}$ (abbreviating secret keys of signature and public-key encryption schemes) is included for compatibility to the original library. Similar statements will occur some more times, e.g., for entries of type pks and pke denoting public signature and encryption keys. No further knowledge of such types is needed for understanding the new work.

returns i^{hnd}, assigning the input values *type* and *add_arg* to the corresponding attributes of $D_u[i^{\mathsf{hnd}}]$ only if $D_u[i^{\mathsf{hnd}}].type$ was null. Else if $|i| > \mathsf{max\_len}(k)$, it returns $i^{\mathsf{hnd}} = \downarrow$. Otherwise, it sets and returns $i^{\mathsf{hnd}} := curhnd_u{+}{+}$, $D_u :\Leftarrow (i^{\mathsf{hnd}}, i, type, add\_arg)$.

For each input port $\mathsf{p}?$, M_u maintains a counter $steps_{\mathsf{p}?} \in \mathbb{N}_0$ initialized with 0 for the number of inputs at that port. All corresponding bounds $bound_{\mathsf{p}?}$ are adopted from the original library without modification. Length functions for inputs are tacitly defined by the domains of each input again.

4.5 Inputs and Their Evaluation

Now we describe how M_u evaluates individual new inputs.

Constructors and One-Level Parsing. The stateful commands are defined via functional constructors and parsing algorithms for each type. A general functional algorithm

$$(type, arg) \leftarrow \mathsf{parse}(m),$$

then parses arbitrary entries as follows: It first tests if m is of the form $(type, m_1, \ldots, m_j)$ with $type \in typeset \setminus \{\mathsf{pka}, \mathsf{sks}, \mathsf{ske}, \mathsf{garbage}\}$ and $j \geq 0$. If not, it returns $(\mathsf{garbage}, ())$. Otherwise it calls a type-specific parsing algorithm $arg \leftarrow \mathsf{parse\_}type(m)$. If the result is $\downarrow$, parse again outputs $(\mathsf{garbage}, ())$. By

"parse m^{hnd}"

we abbreviate that M_u calls $(type, arg) \leftarrow \mathsf{parse}(D_u[m^{\mathsf{hnd}}].word)$, assigns $D_u[m^{\mathsf{hnd}}].type := type$ if it was still null, and may then use *arg*. By

"parse m^{hnd} if necessary"

we mean the same except that M_u does nothing if $D_u[m^{\mathsf{hnd}}].type \neq \mathsf{null}$.

Basic Commands and parse_*type*. First we consider basic commands. They are again local. In M_u this means that they produce no outputs at the network ports. The term "tagged list" means a valid list of the real system. We assume that tagged lists are efficiently encoded into $\{0,1\}^+$.

- *Key constructor:* $sk^* \leftarrow \mathsf{make\_auth\_key}()$.
  Let $sk \leftarrow \mathsf{gen_A}(1^k)$, $sr \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{nonce\_len}(k)}$, and return $sk^* := (\mathsf{ska}, sk, sr)$.
- *Key generation:* $ska^{\mathsf{hnd}} \leftarrow \mathsf{gen\_auth\_key}()$.
  Let $sk^* \leftarrow \mathsf{make\_auth\_key}()$, $ska^{\mathsf{hnd}} := curhnd_u{+}{+}$, and $D_u :\Leftarrow (ska^{\mathsf{hnd}}, sk^*, \mathsf{ska}, ())$.
- *Key parsing:* $arg \leftarrow \mathsf{parse\_ska}(sk^*)$.
  If sk^* is of the form (ska, sk, sr) with $sk \in \{0,1\}^{\mathsf{ska\_len}(k)}$ and $sr \in \{0,1\}^{\mathsf{nonce\_len}(k)}$, return $()$, else $\downarrow$.
- *Authenticator constructor:* $aut^* \leftarrow \mathsf{make\_auth}(sk^*, l)$, for $sk^*, l \in \{0,1\}^+$.
  Set $r \xleftarrow{\mathcal{R}} \{0,1\}^{\mathsf{nonce\_len}(k)}$, $sk := sk^*[2]$ and $sr := sk^*[3]$. Authenticate as $aut \leftarrow \mathsf{auth}_{sk}((r,l))$, and return $aut^* := (\mathsf{aut}, sr, r, l, aut)$.

- *Authenticator generation:* $aut^{\mathsf{hnd}} \leftarrow \mathsf{auth}(ska^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
  Parse l^{hnd} if necessary. If $D_u[ska^{\mathsf{hnd}}].type \neq \mathsf{ska}$ or $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$, then return $\downarrow$. Otherwise set $sk^* := D_u[ska^{\mathsf{hnd}}].word$, $l := D_u[l^{\mathsf{hnd}}].word$, and $aut^* \leftarrow \mathsf{make\_auth}(sk^*, l)$. If $|aut^*| > \mathsf{max\_len}(k)$, return $\downarrow$, else set $aut^{\mathsf{hnd}} := curhnd_u$++ and $D_u :\Leftarrow (aut^{\mathsf{hnd}}, aut^*, \mathsf{aut}, ())$.
- *Authenticator parsing:* $arg \leftarrow \mathsf{parse\_aut}(aut^*)$.
  If aut^* is not of the form $(\mathsf{aut}, sr, r, l, aut)$ with $sr, r \in \{0,1\}^{\mathsf{nonce\_len}(k)}$, $l \in \{0,1\}^+$, and $aut \in \{0,1\}^{\mathsf{aut\_len}'(k,|l|)}$, return $\downarrow$. Also return $\downarrow$ if l is not a tagged list. Otherwise set $arg := (l)$.
- *Authenticator verification:* $v \leftarrow \mathsf{auth\_test}(aut^{\mathsf{hnd}}, ska^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
  Parse aut^{hnd} yielding $arg =: (l)$, and parse ska^{hnd}. If $D_u[aut^{\mathsf{hnd}}].type \neq \mathsf{aut}$ or $D_u[ska^{\mathsf{hnd}}].type \neq \mathsf{ska}$, return $\downarrow$. Else let $(\mathsf{aut}, sr, r, l, aut) := D_u[aut^{\mathsf{hnd}}].word$ and $sk := D_u[ska^{\mathsf{hnd}}].word[2]$. If $sr \neq D_u[ska^{\mathsf{hnd}}].word[3]$
  or $l \neq D_u[l^{\mathsf{hnd}}].word$, or $\mathsf{atest}_{sk}(aut, (r, l)) = \mathsf{false}$, output $v := \mathsf{false}$, else $v := \mathsf{true}$.
- *Message retrieval:* $l^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_aut}(aut^{\mathsf{hnd}})$.
  Parse aut^{hnd} yielding $arg =: (l)$. If $D_u[aut^{\mathsf{hnd}}].type \neq \mathsf{aut}$, return $\downarrow$, else let $(l^{\mathsf{hnd}}, D_u) :\leftarrow (l, \mathsf{list}, ())$.

Send Commands and Network Inputs. Similar to the ideal system, there is a command $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$ for sending a list l from u to v, but now using the port $\mathsf{net}_{u,v,\mathsf{i}}!$, i.e., using the real insecure network: On input $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$ for $v \in \{1, \ldots, n\}$, M_u parses l^{hnd} if necessary. If $D_u[l^{\mathsf{hnd}}].type = \mathsf{list}$, M_u outputs $D_u[l^{\mathsf{hnd}}].word$ at port $\mathsf{net}_{u,v,\mathsf{i}}!$.

Inputs at network ports are simply tested for being tagged lists and stored as in [5].

5 Simulator

We now start with the proof that the real system is as secure as the ideal one. The main step is to construct a simulator $\mathsf{Sim}_{\mathcal{H}}$ for each set $\mathcal{H}$ of possible honest users such that for every real adversary A, the combination $\mathsf{Sim}_{\mathcal{H}}(\mathsf{A})$ of $\mathsf{Sim}_{\mathcal{H}}$ and A achieves the same effects in the ideal system as the adversary A in the real system, cf. Section 2. This is shown in Figure 2. This figure also shows the ports of $\mathsf{Sim}_{\mathcal{H}}$. Roughly, the goal of $\mathsf{Sim}_{\mathcal{H}}$ is to translate real bitstrings coming from the adversary into abstract handles that represent corresponding terms in $\mathsf{TH}_{\mathcal{H}}$, and vice versa. This will be described in the following.

5.1 States of the Simulator

The state of $\mathsf{Sim}_{\mathcal{H}}$ consists of a database D_{a} and a variable $curhnd_{\mathsf{a}}$. Each entry in D_{a} has the following attributes:

- $x.hnd_{\mathsf{a}} \in \mathcal{HNDS}$ is used as the primary key attribute in D_{a}. However, its use is not as straightforward as in the ideal and real system, since entries are created by completely parsing an incoming message recursively.

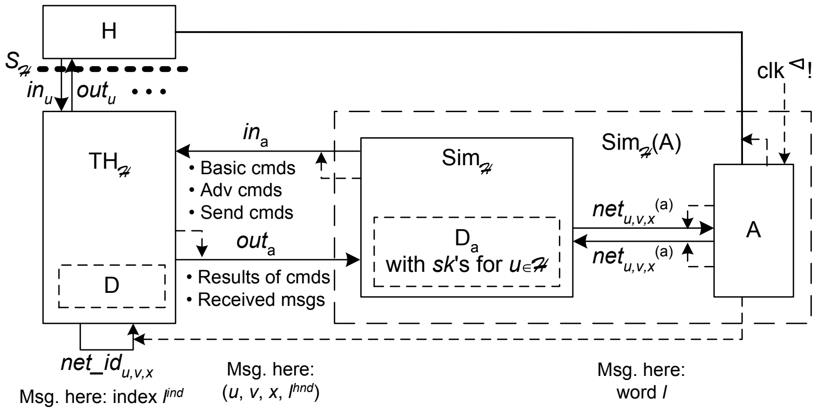


Fig. 2. Set-up of the simulator.

– $x.word \in \{0,1\}^*$ is the real representation of x.
– $x.add\_arg$ is a list of additional arguments. Typically it is (). However, for our key identifiers it is (adv) if the corresponding secret key was received from the adversary, while for keys from honest users, where the simulator generated an authentication key, it is of the form (honest, sk^*).

The variable $curhnd_{\mathsf{a}}$ denotes the current size of D_{a}, except temporarily within an algorithm id2real. Similar to $\mathsf{TH}_{\mathcal{H}}$, the simulator maintains a counter $steps_{\mathsf{p?}} \in \mathbb{N}_0$ for each input port p? for the number of inputs at that port, initialized with 0. Similar to the definition of $\mathsf{TH}_{\mathcal{H}}$, the corresponding bounds $bound_{\mathsf{p?}}$ can be adopted one-to-one from [5], with the only exception that the bound for $\mathsf{out_a}$? has to be enlarged.

5.2 Input Evaluation of Send Commands

When $\mathsf{Sim}_{\mathcal{H}}$ receives an "unsolicited" input from $\mathsf{TH}_{\mathcal{H}}$ (in contrast to the immediate result of a local command), this is the result $m = (u, v, \mathsf{i}, l^{\mathsf{hnd}})$ of a send command by an honest user (here for an insecure channel). $\mathsf{Sim}_{\mathcal{H}}$ looks up if it already has a corresponding real message $l := D_{\mathsf{a}}[l^{\mathsf{hnd}}].word$ and otherwise constructs it by an algorithm $l \leftarrow \mathsf{id2real}(l^{\mathsf{hnd}})$ (with side-effects). It outputs l at port $\mathsf{net}_{u,v,\mathsf{i}}!$.

The algorithm id2real is recursive; each layer builds up a real word given the real words for certain abstract components. We only need to add new type-dependent constructions for our new types, but we briefly repeat the overall structure to set the context.

1. Call $(type, (m_1^{\mathsf{hnd}}, \ldots, m_j^{\mathsf{hnd}})) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$ at $\mathsf{in_a}!$, expecting $type \in typeset \setminus \{\mathsf{sks}, \mathsf{ske}, \mathsf{garbage}\}$ and $j \leq \mathsf{max\_len}(k)$, and $m_i^{\mathsf{hnd}} \leq \mathsf{max\_hnd}(k)$ if $m_i^{\mathsf{hnd}} \in \mathcal{HNDS}$ and otherwise $|m_i^{\mathsf{hnd}}| \leq \mathsf{max\_len}(k)$ (with certain domain

expectations in the arguments m_i^{hnd} that are automatically fulfilled in interaction with $\mathsf{TH}_{\mathcal{H}}$, also for the now extended command $\mathsf{adv\_parse}$ for the new types).

2. For $i := 1, \ldots, j$: If $m_i^{\mathsf{hnd}} \in \mathcal{HNDS}$ and $m_i^{\mathsf{hnd}} > curhnd_{\mathsf{a}}$, set $curhnd_{\mathsf{a}}$++.
3. For $i := 1, \ldots, j$: If $m_i^{\mathsf{hnd}} \notin \mathcal{HNDS}$, set $m_i := m_i^{\mathsf{hnd}}$. Else if $D_{\mathsf{a}}[m_i^{\mathsf{hnd}}] \neq \downarrow$, let $m_i := D_{\mathsf{a}}[m_i^{\mathsf{hnd}}].word$. Else make a recursive call $m_i \leftarrow \mathsf{id2real}(m_i^{\mathsf{hnd}})$. Let $arg^{\mathsf{real}} := (m_1, \ldots, m_j)$.
4. Construct and enter the real message m depending on *type*; here we only list the new types:
   - If $type = \mathsf{pka}$, call $sk^* \leftarrow \mathsf{make\_auth\_key}()$ and set $m := \epsilon$ and $D_{\mathsf{a}} :\Leftarrow (m^{\mathsf{hnd}}, m, (\mathsf{honest}, sk^*))$.
   - If $type = \mathsf{ska}$, let $pka^{\mathsf{hnd}} := m_1^{\mathsf{hnd}}$. We claim that $D_{\mathsf{a}}[pka^{\mathsf{hnd}}].add\_arg$ is of the form (honest, sk^*). Set $m := sk^*$ and $D_{\mathsf{a}} :\Leftarrow (m^{\mathsf{hnd}}, m, ())$.
   - If $type = \mathsf{aut}$, we claim that $pka^{\mathsf{hnd}} := m_2^{\mathsf{hnd}} \neq \downarrow$. If $D_{\mathsf{a}}[pka^{\mathsf{hnd}}].add\_arg[1] = \mathsf{honest}$, let $sk^* := D_{\mathsf{a}}[pka^{\mathsf{hnd}}].add\_arg[2]$, else $sk^* := D_{\mathsf{a}}[pka^{\mathsf{hnd}} + 1].word$. Further, let $l := m_1$ and set $m \leftarrow \mathsf{make\_auth}(sk^*, l)$ and $D_{\mathsf{a}} :\Leftarrow (m^{\mathsf{hnd}}, m, ())$.

5.3 Evaluation of Network Inputs

When $\mathsf{Sim}_{\mathcal{H}}$ receives a bitstring l from A at a port $\mathsf{net}_{w,u,\mathsf{i}}?$ with $|l| \leq \mathsf{max\_len}(k)$, it verifies that l is a tagged list. If yes, it translates l into a corresponding handle l^{hnd} and outputs the abstract sending command $\mathsf{adv\_send\_i}(w, u, l^{\mathsf{hnd}})$ at port $\mathsf{in_a}!$.

For an arbitrary message $m \in \{0,1\}^+$, $m^{\mathsf{hnd}} \leftarrow \mathsf{real2id}(m)$ works as follows. If there is already a handle m^{hnd} with $D_{\mathsf{a}}[m^{\mathsf{hnd}}].word = m$, then $\mathsf{real2id}$ reuses that. Otherwise it recursively parses the real message, builds up a corresponding term in $\mathsf{TH}_{\mathcal{H}}$, and enters all messages into D_{a}. For building up the abstract term, $\mathsf{real2id}$ makes extensive use of the special adversary capabilities that $\mathsf{TH}_{\mathcal{H}}$ provides. In the real system, the bitstring may, e.g., contain an authenticator for which no matching authentication key is known yet. Therefore, the simulator has to be able to insert such an authenticator with "unknown" key into the database of $\mathsf{TH}_{\mathcal{H}}$, which explains the need for the command $\mathsf{adv\_unknown\_aut}$. Similarly, the adversary might send a new authentication key, which has to be added to all existing authenticator entries for which this key is valid, or he might send a transformed authenticator, i.e., a new authenticator for a message for which the correct user has already created another authenticator. Such a transformation is not excluded by the definition of secure authentication schemes, hence it might occur in the real system. All these cases can be covered by using the special adversary capabilities.

Formally, $\mathsf{id2real}$ sets $(type, arg) := \mathsf{parse}(m)$ and calls a type-specific algorithm $add\_arg \leftarrow \mathsf{real2id\_}type(m, arg)$. After this, $\mathsf{real2id}$ sets $m^{\mathsf{hnd}} := curhnd_{\mathsf{a}}$++ and $D_{\mathsf{a}} :\Leftarrow (m^{\mathsf{hnd}}, m, add\_arg)$. We have to provide the type-specific algorithms for our new types.

- $add\_arg \leftarrow \mathsf{real2id\_ska}(m, ())$. Call $ska^{\mathsf{hnd}} \leftarrow \mathsf{gen\_auth\_key}()$ at $\mathsf{in_a}!$ and set $D_{\mathsf{a}} :\Leftarrow (curhnd_{\mathsf{a}}\text{++}, \epsilon, (\mathsf{adv}))$ (for the key identifier), and $add\_arg = ()$ (for the secret key).

Let m =: (ska, sk, sr); this format is ensured by the preceding parsing. For each handle aut^{hnd} with $D_{\mathsf{a}}[aut^{\mathsf{hnd}}].type = \mathsf{aut}$ and $D_{\mathsf{a}}[aut^{\mathsf{hnd}}].word = (\mathsf{aut}, sr, r, l, aut)$ for $r \in \{0,1\}^{\mathsf{nonce\_len}(k)}$, $l \in \{0,1\}^+$, and $aut \in \{0,1\}^{\mathsf{aut\_len}'(k,|l|)}$, and $\mathsf{atest}_{sk}(aut,(r,l)) = \mathsf{true}$, call $v \leftarrow \mathsf{adv\_fix\_aut\_validity}(ska^{\mathsf{hnd}}, aut^{\mathsf{hnd}})$ at $\mathsf{in_a}!$. Return add_arg.

– $add\_arg \leftarrow \mathsf{real2id\_aut}(m,(l))$. Make a recursive call $l^{\mathsf{hnd}} \leftarrow \mathsf{real2id}(l)$ and let $(\mathsf{aut}, sr, r, l, aut) := m$; parsing ensures this format.
  Let $Ska := \{ska^{\mathsf{hnd}} \mid D_{\mathsf{a}}[ska^{\mathsf{hnd}}].type = \mathsf{ska} \wedge D_{\mathsf{a}}[ska^{\mathsf{hnd}}].word[3] = sr \wedge \mathsf{atest}_{sk}(aut,(r,l)) = \mathsf{true}$ for $sk := D_{\mathsf{a}}[ska^{\mathsf{hnd}}].word[2]\}$ be the set of keys known to the adversary for which m is valid.
  Verify whether the adversary has already seen another authenticator for the same message with a key only known to honest users:
  Let $Aut := \{aut^{\mathsf{hnd}} \mid D_{\mathsf{a}}[aut^{\mathsf{hnd}}].word = (\mathsf{aut}, sr, r, l, aut') \wedge D_{\mathsf{a}}[aut^{\mathsf{hnd}}].type = \mathsf{aut}\}$. For each $aut^{\mathsf{hnd}} \in Aut$, let $(\mathsf{aut}, arg_{aut^{\mathsf{hnd}}}) \leftarrow \mathsf{adv\_parse}(aut^{\mathsf{hnd}})$ and $pka_{aut^{\mathsf{hnd}}} := arg_{aut^{\mathsf{hnd}}}[2]$.
  We claim that there exists at most one $pka_{aut^{\mathsf{hnd}}}$ such that the corresponding secret key was generated by an honest user, i.e., such that $D_{\mathsf{a}}[pka_{aut^{\mathsf{hnd}}}].add\_arg[1] = \mathsf{honest}$. If such a $pka_{aut^{\mathsf{hnd}}}$ exists, let $sk^* := D_{\mathsf{a}}[pka_{aut^{\mathsf{hnd}}}].add\_arg[2]$ and $v := \mathsf{atest}_{sk^*[2]}(aut,(r,l))$. If $v = \mathsf{true}$, call $trans\_aut^{\mathsf{hnd}} \leftarrow \mathsf{adv\_transform\_aut}(aut^{\mathsf{hnd}})$ at $\mathsf{in_a}!$ and after that call $v \leftarrow \mathsf{adv\_fix\_aut\_validity}(ska^{\mathsf{hnd}}, trans\_aut^{\mathsf{hnd}})$ at $\mathsf{in_a}!$ for every $ska^{\mathsf{hnd}} \in Ska$. Return ().
  Else if $Ska \neq \emptyset$, let $ska^{\mathsf{hnd}} \in Ska$ arbitrary. Call $aut^{\mathsf{hnd}} \leftarrow \mathsf{auth}(ska^{\mathsf{hnd}}, l^{\mathsf{hnd}})$ at $\mathsf{in_a}!$, and for every $ska'^{\mathsf{hnd}} \in Ska \setminus \{ska^{\mathsf{hnd}}\}$ (in any order), call $v \leftarrow \mathsf{adv\_fix\_aut\_validity}(ska'^{\mathsf{hnd}}, aut^{\mathsf{hnd}})$ at $\mathsf{in_a}!$. Return ().
  If $Ska = \emptyset$, call $aut^{\mathsf{hnd}} \leftarrow \mathsf{adv\_unknown\_aut}(l^{\mathsf{hnd}})$ at $\mathsf{in_a}!$ and return ().

5.4 Properties of the Simulator

Two important properties have to be shown for the simulator. First, it has to be polynomial-time, as the joint adversary $\mathsf{Sim}_{\mathcal{H}}(\mathsf{A})$ might otherwise not be a valid polynomial-time adversary on the ideal system. Secondly, we have to show that the interaction between $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$ in the recursive algorithms cannot fail because one of the machine reaches its runtime bound.

Essentially, this can be shown as in [5], except that the interaction of $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$ in $\mathsf{real2id}$ can additionally increase the number of steps linearly in the number of existing authenticators and existing keys, since a new secret key might update the arguments of each existing authenticator entry, and a new authenticator can get any existing key as an argument. This is the reason why we had to enlarge the bounds at $\mathsf{in_a}?$ and $\mathsf{out_a}?$ to maintain the correct functionality of the simulator, cf. Section 3.3 and 5.1. However, only a polynomial number of authenticators and keys can be created (a coarse bound is $n \cdot \mathsf{max\_in}(k)$ for entries generated by honest users, where $\mathsf{max\_in}(k)$ denotes the permitted number of inputs for each honest user, plus the polynomial runtime of A for the remaining ones). We omit further details.

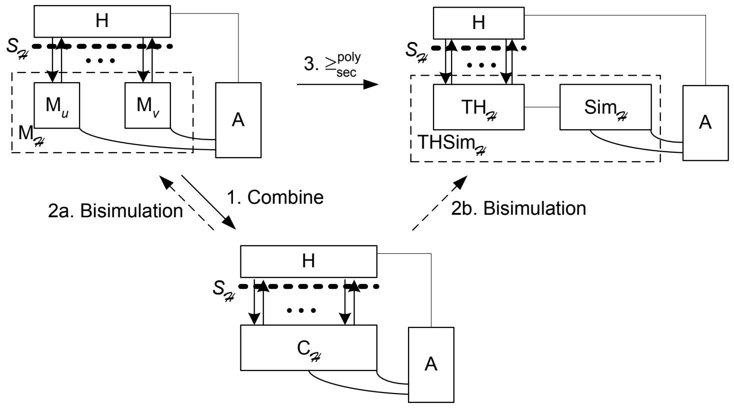


Fig. 3. Overview of the Simulatability Proof.

6 Proof of Correct Simulation

Given the simulator, we show that even the combination of arbitrary polynomial-time users H and an arbitrary polynomial-time adversary A cannot distinguish the combination $M_{\mathcal{H}}$ of the real machines M_u from the combination $THSim_{\mathcal{H}}$ of $TH_{\mathcal{H}}$ and $Sim_{\mathcal{H}}$ (for all sets $\mathcal{H}$ indicating the correct machines). We do not repeat the precise definition of "combinations" here. As the rigorous proof of correct simulation takes 11 pages, we only give a brief sketch here.

The proof is essentially a bisimulation. This means to define a mapping between the states of two systems and a sufficient set of invariants so that one can show that every external input to the two systems (in mapped states fulfilling the invariants) keeps the system in mapped states fulfilling the invariants, and that outputs are identical. We need a probabilistic bisimulation because the real system and the simulator are probabilistic , i.e., identical outputs should yield mapped states with the correct probabilities and identically distributed outputs.

However, the states of both systems are not immediately comparable: a simulated state has no real versions for data that the adversary has not yet seen, while a real state has no global indices, adversary handles, etc. We circumvent this problem by conducting the proof via a combined system $C_{\mathcal{H}}$, from which both original systems $THSim_{\mathcal{H}}$ and $M_{\mathcal{H}}$ can be derived. The two derivations are two mappings, and we perform the two bisimulations in parallel. By the transitivity of indistinguishability (of the families of view of the same A and H in all three configurations), we obtain the desired result. This is shown in Figure 3.

In addition to standard invariants, we have an information-flow invariant which helps us to show that the adversary cannot guess certain values in these final proofs for the error sets. Although we can easily show that the probability of a truly random guess hitting an already existing value is negligible, we can only exploit this if *no* information (in the Shannon sense) about the particular value has been given to the adversary. We hence have to show that the adversary did not even receive any *partial* information about this value, which could be derivable since, e.g., the value was hidden within a nested term. Dealing with aspects of this kind is solved by incorporating static information-flow analysis in the bisimulation proof.

Finally, the adversary might succeed in attacking the real system with very small probability, which is impossible in the ideal system. We collect these cases in certain "error sets", and we show that these sets have negligible probability (in the security parameter) at the end; this is sufficient for computational indistinguishability.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.
2. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
4. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. IACR Cryptology ePrint Archive 2003/145, July 2003. `http://eprint.iacr.org/`.
5. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003. `http://eprint.iacr.org/`.
6. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology: CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.
7. R. Canetti. A unified framework for analyzing security of protocols. IACR Cryptology ePrint Archive 2000/067, Dec. 2001. `http://eprint.iacr.org/`.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.
9. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
10. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
11. J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.
12. M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.
13. R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

14. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
15. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
16. C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
17. J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
18. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
19. B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. Presented at the DERA/RHUL Workshop on Secure Architectures and Information Flow, 1999, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000. `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`.
20. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
21. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.
22. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, 1995.
23. S. Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

An Improved Reference Flow Control Model for Policy-Based Intrusion Detection

Jacob Zimmermann, Ludovic Mé, and Christophe Bidan

Supélec, Campus de Rennes, équipe SSIR
{firstname.lastname}@supelec.fr

Abstract. In this paper, we describe a novel approach to policy-based intrusion detection. The model we propose checks legality of information flows between objects in the system, according to an existing security policy specification. These flows are generated by executed system operations. Illegal flows, i.e., not authorized by the security policy, are signaled and considered as intrusion symptoms. This model is able to detect a large class of attacks, referred to as "attacks by delegation" in this paper. Since the approach focuses really on attack effects instead of attack scenarii, unknown attacks by delegation can be detected.

Keywords: Policy-based intrusion detection, information flow control, access control

1 Introduction

Traditional intrusion detection techniques build on two distinct approaches: scenario-based detection and anomaly-based detection. In the former case, an intrusion detector includes a database of existing attack scenarii and algorithms that correlate observed system events in order to detect known patterns. In the latter case, observed events are compared to a "normal behaviour" profile that usually results from a statistical learning process. While these intrusion detection systems (IDSes) have been effective in many cases, they raise a number of inherent problems [1]:

– **maintenance costs:** for a scenario-based IDS to remain useful, its database must be constantly upgraded to include new attack signatures as they are discovered. Anomaly-based detectors require a new profile building phase whenever new applications with legitimate but different behaviour are put in service;
– **detection reliability:** a scenario-based IDS is by definition limited to known attack scenarii, which are less likely to be used by a possible intruder. Novel attack detection remains limited or impossible. This may produce false negatives, i.e., successful attack not reported by the detector;
– **detection accuracy:** while successful attacks may remain undetected, an IDS also produces false-positives, i.e., alerts reporting intrusions that did *not* occur.

We believe that to a large extent, these problems arise from the fact that anomaly or scenario-based intrusion detection relies on a context-free definition of what a potential attack may be: while by definition any attack is a security policy violation, the security policy itself is not taken into account in these approaches.

There has been increasing interest in *policy-based* intrusion detector development in the recent years [2]. A policy-based IDS consists in a logical security policy specification and an execution trace validation algorithm. Such an approach has the potential to improve significantly over statistical anomaly detection and scenario detection in terms of reliability, accuracy and required maintenance. Ideally, maintenance is needed only to change the specified policy, there is no need for permanent upgrades or adaptation phases.

The present work pursues these goals by modeling a system in terms of objects, operations and information flows between objects. A criterion to distinguish between legal and illegal information flows according to a security policy is established. Information flows that appear as illegal are symptomatic of security policy violation attempts and should be reported as intrusion alerts.

Our purpose is not to enforce a security policy *a priori*. Instead, we use information flow control as a means to detect given access control criteria violations. We want the policy enforcement mechanism to remain transparent to users as much as possible. In this paper, we focus on discretionary access control policies, as these are used in the actual targeted operating systems. Nevertheless, the approach is general enough to be able to handle also for instance a Chinese Wall policy or a Bell&LaPadula policy.

In this paper, we present a formal intrusion detection model. Some implementation issues are discussed and we show the ability of the model to detect actual intrusions.

The rest of this paper is organized as follows. In section 2, we present the class of attacks we focus on and related work that provide partial solutions to prevent or detect such attacks. The proposed intrusion detection model is formally introduced in section 3. Section 4 describes our implementation on the Linux OS, as well as an actual attack detection example. Finally, section 5 presents the conclusion of our current work and some future directions.

2 Background

In this section, we present a class of host-based confidentiality or integrity violations, referred to as “attacks by delegation”. These attacks can be described and modeled in terms of information flows. We briefly present and discuss several existing techniques and models that can be used to deal with these attacks.

2.1 Attacks by Delegation

Integrity and confidentiality policies in operating systems are primarily enforced using access control mechanisms. If the access control implementation is correct, then it forbids any action that explicitly violates desired confidentiality and integrity criteria. However, experience shows that security policy violations, i.e., intrusions, are still possible. Even if the attacker is prohibited from executing a specific action, an intrusion can be achieved by various means:

- The well-known access leakage problem, inherent to HRU-based access control models [3], can be exploited. In this case, the action eventually executed by the attacker is *explicitly authorized* by the access control, but is *semantically equivalent* to an action declared as illegal in terms of access to information;
- Race conditions can be exploited by interfering with the actions executed by some other process or subject. The attacker knows that a specific action will eventually be executed and schedules its own actions in order to take advantage of this. While all the actions performed by the attacker are legal by themselves, the security policy is violated by the *effect of their scheduling*;
- Abusing the privileges of other subjects is another form of interference, as in *confused deputy*-based attacks. In this case, the policy is violated by the fact that this action is executed as a consequence of an action of the attacker. Under some circumstances, *Trojan horses* and *back-doors* can be considered as confused deputy exploits, where the confused deputy itself is designed by the attacker to meet his needs. Once installed using some other attack, it can be activated by the attacker purposely to perform operations that are not allowed to him;
- We can consider some *buffer overflow*-based attacks as special cases of Trojan horses. Here, a misbehaving program executing actions programmed by an attacker using a buffer overflow exploit has the same effect as a Trojan horse installed by the attacker.

These intrusion scenarii consist in series of actions, where each one is legal from the access control point of view when considered separately. However, when combined and executed in a particular context, they lead to some effect that is explicitly forbidden by access control.

We call such attacks *attacks by delegation.* Indeed, an attacker aiming at executing some action, but not allowed to do so, will "delegate" the action to some other subject, or a special context he produces. Note that this concept is orthogonal to the more common classification of attacks (DoS, read access to data, execution of code), because in each of these categories, number of attacks can be modeled as attacks by delegation.

Attacks by delegation can be described in terms of information flows between objects in the system. As a simple example, let us consider the well-known attack against *lpr* (please refer to table 1). While each step is legal *per-se*, the final result is that the attacker prints /etc/shadow, which is prohibited by the DAC policy. This attack results in an information flow from a file (in this example, /etc/shadow) to the printer device, while the printer is not an allowed destination for this flow. The printer daemon is used as a confused deputy to produce such a flow (in step 6), by combining other legal information flows using a race condition (in steps 3 and 4).

Table 1. The *lpr* attack

	lpr daemon	attacker
1		lpr /home/user/doc.txt
2	read /var/spool/lpd/job-****	
3		rm /home/user/doc.txt
4		ln -s /home/user/doc.txt /etc/shadow
5	read /home/user/doc.txt	
6	send document to printer	

2.2 Related Work

Information Flow Control. In the general case, access control mechanisms that consider each operation separately cannot protect a system against attacks by delegation. In most cases, rights to execute operations are granted on a strictly context-free subject/object identity basis, while interference and cooperation schemes take into account context and execution history when dealing with access to data and data modification. In other words - the security policy is implemented at the *object access control level*, while attackers operate at the *information flow level*.

In theory, this problem can be solved by using information flow control models; a large number of these has been proposed [4,5,6,7,8,9,10,11]. However, when focusing on implementing given security policies on existing general-purpose operating systems, their actual usability remains limited due to various practical reasons:

1. Models that enforce complete control over all possible information flow channels, including hidden ones, require knowledge of control flows and dependency graphs [4,5]. In our opinion, such a fine granularity is unrealistic on a large-scale OS running third-party software.
2. Coarse-grained, multi-level models that enforce control over well-defined channels have been proposed (such as the Bell&LaPadula [6], Biba [7] and Chinese Wall [8] models). A common basis to many of these models is that they enforce unidirectional information flow channels and ensure that the graph of all possible information flows is acyclic [9]. This is generally considered too strict for practical usage, so that exceptions have to be introduced, in the form of group intersections, declassification operations, method waivers [10], etc. Again, static program analysis may be required in such cases to avoid substantial overhead of run-time information flow checking [11];
3. Particular models designed for specific applications rely on assumptions met by a highly specialized operating environment, but remain unusable in the general case. For instance, the model described in [12] exploits the Java visibility rules and security constraints to infer knowledge of possible information flow channels, which is a form of static program analysis.

Neither of these approaches is directly usable on a generic OS such as Linux, Unix or Windows, especially if the existing security policy remains unchanged and the system runs unmodified closed-source software.

Access Control. Confused deputy attacks can be avoided using a capability-based access control model such as [13,14]. By binding possible operations of a process directly to objects, these models prevents attacks such as those based on symbolic links or file renaming. Similarly, the problem of access leakage in access control models can be addressed in multiple ways. Typed access control models [15,16] provide means to define safety invariants and ensure that a subject will not be able to gain certain undesired privileges. Nevertheless these models remain unable to prevent malicious cooperation or interference. They also fall short in dealing with back-doors or operation side-effects.

To deal with privilege and information transfer from an access control point of view, the "Take-Grant" model and its derivatives were developed [17,18,19]. This formalism allows to distinguish between *de jure* transfers, where a subject gains privileges that allow him to access some informations, and *de facto* transfers, that consist in accessing some copy of that information [20]. Later, the problems of privilege theft, conspiracy [21,22] and information flow between subjects and objects [23] have been modeled using this framework. The proposed solutions provide means to validate a given access control policy, modeled using the "Take-Grant" formalism, against potential *de facto* transfers or conspiracy scenarii.

Some attacks by delegation actually create *de facto* transfers or privilege thefts, and could thus be prevented by modifying the policy using these approaches. However, others forms of attacks involving illegal information flows (such as Trojan horses or back-doors) cannot be modeled so easily this way.

Intrusion Detection. In the intrusion detection field, an interesting approach to race conditions-based attacks detection has been proposed [24]. The authors define a property of subjects non-interfering with data and propose an algorithm for system integrity enforcement. They show their model's ability to detect novel security policy violations, however security policies taken into account in this work focus only on subjects being forbidden to overwrite a piece of data. In Appendix A, we show that policies of this form can easily be handled also using our model.

An approach to prevent privilege abuse and confused deputy attacks is the "privilege graph" model [25], that takes into account implemented subject privileges, as well as vulnerabilities that allow some subject to gain additional privileges. This model can be used to evaluate a given security policy by estimating the global security level of that policy and the impact of policy changes (subject creation, subject privilege modification, etc.). From an intrusion detection point of view, this can be seen as an event correlation model. However it is not a runtime intrusion detection mechanism by itself.

2.3 Positioning of Our Work

Since our goal is to detect attacks by delegation that produce illegal information flows, we propose to enforce a simplified information flow control by restricting our focus only to flows between system objects. In practical terms, anything that is identified as an "object" by the underlying operating system fits in the model: in a Windows or Unix-like system, this includes files, pipes, FIFOs, sockets, message queues, shared memory buffers etc.

Our purpose is not to enforce a security policy *a priori*. Instead, we use information flow control as a means to detect given access control criteria violations by various forms of interference or confused-deputy exploits at runtime. In this paper, we focus on **discretionary access control policies**, as these are used in the targeted operating systems. Nevertheless, the approach is general enough to be able to handle also for instance a Chinese Wall policy or a Bell&LaPadula policy.

3 Intrusion Detection Model

We propose to simplify information flow control by considering only flows between system objects. Any access to an information is modeled by a method call on the object containing that information. By executing series of method calls, system operations produce information flows between objects. A security policy is enforced by defining domains that specify sets of method calls, each combination of these calls producing authorized information flows. By definition, to be legal, information flows must occur within an unique domain; any flow that violates this rule violates by definition a confidentiality or integrity requirement.

3.1 Objects and Operations

The elementary unit of information in our model is an *atomic object state*. Atomic objects provide information containers. Their state is set and retrieved using method calls.

According to the usual definition, the state of an object is defined by the values of its attributes. Atomic objects[1] are objects whose attributes depend on each other and cannot be modified separately, thus the security policy can only deal with the object as a whole (for instance, a file contents, not a part of it). In practical terms, our model can enforce a policy if a file contents is accessed using the *read* and *write* methods as an atomic objects, but not if it is accessed through a raw device or by mapping the file as part of the virtual memory of a process (which are not atomic)[2].

[1] In this paper, we further refer to *atomic objects* simply as to *objects*.

[2] In current operating systems, non-atomic object exist; however, the problem is known and is being addressed in current development [26]. Thus, our model only handles atomic objects, while the implementation will use the possibility of emulating atomic objects as soon as it will be released.

By allowing to access and modify an object state, its methods in effect produce information flows. Any operation executed on the system that accesses or modifies the state of an object thus has a meaning in terms of generated information flows between objects. As such, it is defined by a set of objects and the set of called methods. We use the symbol $\gg$ to denote such operations: an operation

$$\{o.m\} \gg \{o'.m'\}$$

creates an information flow from object o to object o' by accessing the state of o using its method m and, as a consequence, setting the state of o' using its method m'.

Since we focus only on flows produced by method calls, information flows using hidden channels are clearly outside of our scope.

Our operating system model is formally specified as follows:

Definition 31 *A system is a tuple* $(O, M, \Omega, S, exec)$ *where*

- O *is a set of object identifiers;*
- M *is a set of object methods;*
- $\Omega \subseteq \{(O \times M)^* \gg (O \times M)^*\}$ *is the set of operations that generate information flows between objects,* $*$ *being used as a notation for the powerset;*
- S *is a set of system states;*
- $exec : \Omega \to (S \to S)$ *are the information flow semantics of operation execution, i.e., the system state transition functions.*

This model provides an instance of an EM enforcement mechanism [27]. The set of system states S and state transition functions $exec$ will be formally defined in the following sections.

3.2 Domains and References

An attack by delegation symptom is an information flow between objects that violates the enforced security policy. To detect operations that create such flows, we need to define a "legality condition" for the generated flows.

As implemented by an access control mechanism, a security policy explicitly declares legal and illegal flows. Let us consider the sample DAC policy from table 2. When applied to a given system state (i.e., the states of all the system objects), this policy declares an operation such as $\{m.r\} \gg \{n.w\}$ as legal. On the opposite, we consider that any access that is not explicitly authorized is forbidden. Thus, an operation such as $\{m.r\} \gg \{p.w\}$ is illegal.

In the general case, a security policy implies legal combinations of object method calls to be used by operations. We call *domain* a set of method calls such that any combination of these is legal as regard to the security policy. Therefore, an operation that executes only method calls that belong to the same domain is legal by definition. Similarly, an operation that consists in method calls of several distinct domains is illegal, as the information flow it generates is not considered as authorized by the security policy.

Table 2. Sample access control policy

	Alice	Bob
m	$\{r, w\}$	$\emptyset$
n	$\{r, w\}$	$\{r, w\}$
o	$\{r, w\}$	$\{r, w\}$
p	$\emptyset$	$\{w\}$

Formally, the method calls that belong to a given domain are defined by a set of *references*. A reference $R_d o.m$ allows to execute the $o.m$ method call in the domain d. Given a system state (i.e., the states of all objects), each object is bound to a set of references.

Definition 32 *Let D be a set of domain identifiers. A system state is a objects-to-references binding:*

– $S : O \to \{R_d o.m | d \in D, o \in O, m \in M\}$

Given an object $o \in O$ and a state $s \in S$, $s(o) \subseteq \{R_d o.m | d \in D, m \in M\}$ is the set of all references that provide legal accesses to the o object in the s state.

Defining domains and related reference sets in an initial system state is the means to implement the existing security policy. The actual domain building algorithm depends on the security policy one wish to implement, the common DAC case is treated in section 4.2.

For a single information flow, the "domain uniqueness rule" is formally defined by the *islegal* predicate:

Definition 33 *We define the function $dom : (O \times M)^* \times S \to D^*$ as the set of all domains that allow a set of object method calls to be executed in a system state. Given an operation and a system state, we define the predicate $islegal : \Omega \times S \to \{true, false\}$ as true if all the executed method calls are allowed within some same domain, i.e. if there exists some domain where the appropriate method calls on both the source objects and the destination objects can be executed.*

– $dom[\{o_1.m_1, \ldots, o_p.m_p\}, s] = \{d \in D | \forall i \in [1 \ldots p] R_d o_i.m_i \in s(o_i)\}$
– $islegal(src \gg dst, s) = true \text{ if } dom(src, s) \cap dom(dst, s) \neq \emptyset$

This predicate checks that there exists at least one domain where accessing both the source objects (*src*) and the destination objects (*dst*) is allowed. Since any flow within some domain is by definition legal, the "domain uniqueness rule" ensures that no illegal information flow occurs when executing the $src \gg dst$ operation.

3.3 Causal Dependency and Reference Flow

Information flows between objects are also generated by sequences of causally dependent operations. According to Lamport's causal ordering [28], an operation

$\omega_2 : src_2 \gg dst_2$ is a causal consequence of some operation $\omega_1 : src_1 \gg dst_1$ if it accesses an object state set by ω_1, i.e., if there exists an object $o \in O$ and two methods $(m, m') \in M \times M$ such that $o.m \in dst_1$ and $o.m' \in src_2$. In this case, the sequence $[src_1 \gg dst_1; src_2 \gg dst_2]$ actually creates an information flow $src_1 \gg dst_2$. Thus, after having executed $src_1 \gg dst_1$, $src_2 \gg dst_2$ is legal if, and only if, $src_1 \gg dst_2$ is legal.

Since information flow legality is enforced using references, that means that if $src_1 \gg dst_1$ occurs, the dst_1 objects must then have references similar to those of the src_1 objects. An information flow thus results in a flow of references. More precisely, whenever a legal information flow occurs, references from the source objects are propagated to the destination objects. Executing $src \gg dst$ in some system state s thus generates a new system state s' through the state transition function $exec : \Omega \to (S \to S)$.

Definition 34 *$exec(src \gg dst) = s \to s'$ such that $\forall o \in O$,*

- *if $\exists m \in M | o.m \in dst$*
  then $s'(o) = Prop(o, s, src \gg dst) \cup Create(o, src \gg dst)$ where
  - *$Prop(o, s, src \gg dst) = \{R_d o.m' | d \in dom(src, s)$ and $\exists o' \in O, m'' \in M | o'.m'' \in src \wedge R_d o'.m' \in s(o')\}$*
  - *$Create(o, src \gg dst)$ is a set of new references.*
- *else $s'(o) = s(o)$*

When a flow from an object o' to an object o occurs in the state s, the operations possible on o in the new state s' must be at most the operations possible on o' in the previous state s. This is enforced by the reference flow, formalized by the definition 34: *Prop* is the set of flowing references, whereas *Create* is used in operations that create new objects. The first condition in *Prop*, i.e. $dom(src, s)$, selects all the domains in which a flow sourcing at the *src* objects in state s is legal. The second condition select all references that relate to source objects (i.e. objects that have methods called in *src*) in these domains. Notice that theoretically, some of these methods may not exist for the destination objects. However, a non-existing method will by definition not be called, so these references will never be involved in any further reference flow. Moreover, practically, all objects involved in a given operation have the same interfaces: thus this case does not even occur in the implementation.

Object creation operations are not formalized specifically, since they simply consist of information flows to objects. However, for the new objects being accessible, appropriate created references to these objects must be bound to $dom(src, s) \cap dom(dst, s)$. Just like representing known operations using the $src \gg dst$ formalism, this depends on the actual system implementation.

As an example, let us consider again the DAC policy from table 2. In the scenario shown on table 3, operation at step 1 generates a flow[3] from m to n; a flow from m to p would have been illegal since the DAC policy allows either

[3] In table 3, $n \to p$ denotes an authorized flow from n to p, i.e., $islegal(\{n.m_1\} \gg \{p.m_2\}, s) = true$, $m_1, m_2 \in M$.

Table 3. Illegal sequence of operations

s	operation	legal flows	islegal
0		$m \leftrightarrow n; m \leftrightarrow o; o \leftrightarrow n; o \rightarrow p; n \rightarrow p$	
1	$\{m.r\} \gg \{n.w\}$	$m \leftrightarrow n; m \leftrightarrow o; o \leftrightarrow n; o \rightarrow p$	true
2	$\{n.r\} \gg \{p.w\}$		false

reading m or writing p, but never both simultaneously. Thus, the flow from n to p at step 2 is also illegal, because it would result in a flow from m to p through n.

3.4 Intrusion Detection

Let us consider a $(O, M, \Omega, S, exec)$ system model with S and $exec$ as defined in the previous section. We suppose the system to be in some initial state s_0 that binds objects and their methods into domains as implied by the access control policy. Thus, any cross-domain operation is supposed to be an intrusion symptom. To detect such symptoms in an execution trace, i.e., a sequence of operations $[\omega_1, \omega_2 \ldots \omega_n]$, we have to check the "no cross-domain" property for each of the operations. That is:

1. For each operation ω_i, the *islegal* predicate must be *true*.
2. Each operation modifies the state of the system as defined by the *exec* function,

4 Implementation

We have developed a run-time intrusion detector using this model on Linux. This system matches well the object-based model with a discretionary access control policy. In addition, the system being open-source provides an ideal testbed environment.

Our implementation has shown promising abilities to detect different realistic attacks, including unknown attacks. In this section, we give some details about this implementation and show, on one interesting example, its ability to detect actual intrusions.

4.1 Modeling the Linux OS

In this implementation, we consider the following objects as information containers:

– Filesystem objects (i.e., regular files, directories and named FIFOs). Special "device files" are not considered;
– User and client related metadata: authentication accounts handled through the PAM system and IP ports

Table 4. Sample reference requirements

System call	Reference requirements	Created flows
sendfile(out,in,offset,count)	$R_d out.w, R_d in.r$	$\{in.r\} \gg \{out.w\}$
read(fd,buf,count)	$R_d fd.r, R_d buf.r, R_d buf.w$	$\{fd.r, buf.r\} \gg \{buf.w\}$
mmap(s,l,p,f,fd,o)[4]	$R_d s.r, R_d s.w, fd.r$	$\{s.r, fd.r\} \gg \{s.w\}$

- Input/output descriptors for filesystem objects, pipes and sockets;
- POSIX inter-process messages;
- POSIX shared memory objects;
- process memory images.

The subset of system calls that we must handle to deal with these objects is quite small: while Linux kernel version 2.4 defines over 250 system calls, only 16 generate actual information flows between objects and are taken into account in our present implementation. These are calls that create or use I/O descriptors (file, pipe and socket operations), the `mmap` call, the `msgget` and `msgsnd` calls. Each has well-known semantics regarding information flows, from which we derive a precise specification in terms of reference requirements, such as those shown on table 4, and also reference creation where it applies. These rules are defined once and for all by the implementation.

As described in section 3.1, the implemented model assumes objects to be atomic. Nevertheless, some objects in Linux are not atomic: for instance, each line the */etc/shadow* file can theoretically be overwritten only by information flowing from a given user's console. This cannot be easily modeled, so some attacks (such as *Alice* changing *Bob*'s password instead of her own) will not be detected. However, a new filesystem is currently being developed that will eventually solve the problem by allowing to represent any information as a set of atomic files [26]. Once this feature will become available, we will take advantage of it.

4.2 Building the Initial State

Since Linux enforces a discretionary access control policy, the initial object-to-references binding s_0 is easily deduced from the DAC matrix by forbidding *de facto* access rights transfers [20]. Since DAC is the primary means to implement security policies in Linux (and other current operating systems), we use it as a security policy specification. Thus, our goal is not to refine or correct DAC permissions, but to enforce the existing policy in cases where it could be violated by some attack by delegation.

Basically, by reading and writing objects he is allowed to access, any subject can generate information flows between these objects. Our interpretation of the

Algorithm 1 Building the initial state s_0:

```
1. Let $U$ be the set of Linux subjects other than $root$.
2. Initially
   (a) $D = U$
   (b) $\forall o \in O, s_0(o) = \emptyset$
3. For each $u$ in $U$ :
   (a) for each method $m$ in $M$, let $All_m(u)$ be the set of objects that $u$ is
       allowed to execute $m$ on.
   (b) If there exists an $u'$ in $U$, $u' \neq u$, such that for any method $m$, $All_m(u) \subseteq$
       $All_m(u')$
       i. then $D \leftarrow D - \{u\}$
       ii. else for each $o$ in $All_m(u)$, $s_0(o) \leftarrow R_u o.m$
```

policy is that such a flow is legal if some subject is allowed to read the source objects and write the destination objects. Thus, all flows a given subject can generate belong by definition to the same domain(s).

This interpretation of the Linux DAC is problematic in one case: suppose that *Bob*'s access rights are strictly a subset of *Alice*'s ones. That means that any flow *Bob* can possibly produce can also be generated by *Alice*; so all flows that either *Bob* or *Alice* can generate belong to the same domain(s). Therefore, if an attack by delegation attempts at creating a flow between *Alice*'s and *Bob*'s objects, it will remain undetected.

We argue, however, that there is no harm due to this. For the situation to arise, *Bob* would actually have to share *everything* with *Alice*, including her authentication account. If this is the case, then no action of *Bob* could be distinguished from an *Alice*'s action, which is undoubtedly a policy anomaly.

The set of domains D and the initial state s_0 are built by the algorithm 1. For any subject but *root*, it checks whether his rights are superseded by those of some other subject. If it is not the case, then all flows this subject can produce define one domain.

4.3 Running the Checker

Our implementation consists in a user-space daemon running as a low-priority background process. A trivial kernel module hooks on the observed system calls and sends a message to the daemon whenever any of them is executed by a process. Thus, actual reference flow processing and checking the *islegal* condition is asynchronous. Using this design, the performance degradation on the target host is kept low (please see table 5 for an example).

Theoretically, the maximum number of domains is *number of objects * number of subjects*, assuming a DAC policy. However, most of the domains actually overlap; by considering all flows that may be generated by a given DAC subject as belonging to the same domain, the maximum number of automatically gen-

Table 5. Linux kernel compilation benchmark *(make bzImage)*

# of operations	12051434
default	9m 40s
with active checker	11m 23s

erated domains is the *number of subjects*. Let us notice that the initial domain construction phase takes approximately 15 minutes on a standard PC running Linux with 13 users and over 120000 objects, or 8 minutes on our mail server (which has a much faster SCSI disk).

The *islegal* test complexity is linear in the number of checked references. There can be at most *number of methods* ∗ *number of domains* references per object; most objects have only three methods (*read*, *write* and *exec*). Because no new domains are created during execution[5], the maximum number of references per object remains constant.

The maximum number of propagated references per object is *number of methods*number of domains*, in the worst case (a flow being allowed in all the domains at the same time, such as a flow between two world-readable, writable and executable objects). In our test on the mail server, this worst number is 39. However, most of the time, a flow is allowed in only one domain; thus, the number if propagated references per object is *number of methods*, i.e. 2 (*read* and *write* methods, since executable objects are rarely involved in flows).

4.4 Actual Attack Detection Example

As an actual intrusion detection example, we present a Trojan horse-based attack against OpenSSH [29]. The purpose of this attack is to force the login process started by OpenSSH to execute a *setuid(0)* operation and thus start a *root* shell, regardless of the attacker's authentication. The exploit is based on the fact that the OpenSSH daemon can be configured to use an external login program, and to allow users set their own environment variables before the login procedure begins.

Please refer to table 6 for the attack scenario. It consists in creating a shared library (*libroot.so*) that overloads the *setuid* system call. When executed through this library, *setuid* effectively sets the process' *uid* to 0. The attacker then uses the *LD_PRELOAD* environment variable to force loading of this library when the */bin/login* executable is started. After successful authentication, the login program executes an *setuid* operation to set the authenticated user's identity. Because of *libroot.so* being loaded, this becomes in effect *setuid(0)*. As a result, the attacker gets a *root* shell.

At step 1, the *libroot.so* file is created as a destination object of an information flow whose source is primarily an object that represents the attacker's console.

[5] Currently, new user creation or policy modification is not handled by our implementation.

Table 6. OpenSSH attack

	operation	required references
1	attacker creates *libroot.so*	$R_{d1}console.r$, $R_{d1}librootso.w$
2	attacker sets *LD_PRELOAD*	
3	attacker starts ssh session	
4	sshd executes */bin/login*	$R_{d2}login.r$, $R_{d2}login.x$, $R_{d2}img.w$
5	login loads *libroot.so*	$R_{d1}librootso.r$, $R_{d2}img.r$, $R_{d2}img.w$
6	authentication process	
7	login executes *setuid(0)*	
8	login executes */bin/sh*	
9	*root* shell is available	

We consider that this flow occurs within some domain d_1, and that no flow is permitted between the console and the running *sshd* daemon. At step 4, the memory image of the *login* process (here noted as *img*) results from a child of the *ssh* daemon executing */bin/login*; that is, a flow from */bin/login* to *img*. It belongs thus to some domain d_2 where it is possible both to read the contents of */bin/login* (most probably, this is possible in any domain) and to overwrite the memory image of the calling process *img*.

These two flows converge at step 5, by the contents of *libroot.so* flowing into *img* when the *mmap* system call is executed. Since it is necessary to write *img* again, this could occur but in the domain d_2. However, the read reference available for *libroot.so* is $R_{d1}librootso.r$. Since this flow requires thus references from two domains, it is illegal and should be considered as an intrusion symptom.

The security policy goal enforced here is precisely that no information produced by the attacker should modify the *login* program. No hypothesis is made on how this modification may actually occur. It can be:

- by actually overwriting somehow the */bin/login* file;
- by using the described attack against OpenSSH;
- by using the same attack against another program[6];
- and so on.

All these cases lead to an identical effect in terms of information flows between objects: a cross-domain flow occurs between an object produced by the attacker and an object causally bound to the *login* program. As such, the attack will be detected in these terms in all cases, including possible novel ones.

Note: In this attack, the model in effect results in forbidding usage of user-defined shared libraries in programs that aren't executed under the user's own identity. Note that this very behaviour is implemented *ad-hoc* in the ELF dynamic linker, precisely to avoid such misuse. The OpenSSH attack is an attack by delegation that overcomes this prevention mechanism.

[6] Originally, this was a known problem of the telnet daemon, which was resolved. The same vulnerability being discovered later in OpenSSH shows the limits of this "patch and pray" approach.

5 Conclusion

We have proposed a mechanism to detect host-based attacks by delegation by checking information flows between system objects through known and expected channels. This mechanism requires neither actual attack scenarii knowledge, nor empirical "normal behaviour" profile. As such, it is able to detect known as well as novel attacks and is suitable for host-based, policy-based intrusion detection.

The domains definition, resulting from a security policy specification, provides in effect legal execution scenarii constructs. The proposed approach is thus as special case of anomaly-based intrusion detection. Detected illegal operations provide information about attempted attack semantics (as an attacker trying to modify the *login* program execution), but no information about the precise attack scenario that led to this operation.

We have developed a runtime implementation on the Linux operating system. Our current test are encouraging, the system actually proves to be able to detect known as well as unknown host-based attacks, as well as some remote attacks that lead to a cross-domain information flow on the target system. The experiments show that the implementation of the model raises alarms only if an illegal flow actually occurs. False alarms exist, but are due to special cases where neither the policy specification, nor the model, are precise enough. These cases remain extremely rare and thus the number of false alarms is very low (4 in 26 million of processed events). These results are discussed in [30].

Acknowledgments

The work presented in this paper was partly funded by the French Ministry of research in the context of the DICO RNTL project. We would like to thank the Ministry and the RNTL for their support. We also would like to thank Heiko Mantel for in-depth comments and remarks about the detection model presented in this paper, as well as the anonymous reviewers for many valuable comments.

References

1. Herve Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.
2. Chuck Boeckman. Getting closer to policy-based intrusion detection. *Information Security Bulletin*, pages 13–20, May 2000.
3. M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *ACM*, 19(8):461–471, August 1976.
4. Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
5. Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *j-TOCS*, 1(3):256–277, August 1983.
6. D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical report, The Mitre Corp., 1976.

7. K. Biba. Integrity considerations for secure computer systems. *MTR-3153, Mitre Corporation*, 1975.
8. David F.C. Brewer and Michael J. Nash. The chinese wall security policy. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.
9. K. G. Walter et al. Primitive models for computer security. Technical Report ESD-TR4 -117, Case Western Reserve University, 1974.
10. E. Ferrari, P. Samarati, E. Bertino, and S. Jajodia. Providing flexibility in information flow control for object-oriented systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 130–140, 1997.
11. Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Symposium on Operating Systems Principles*, pages 129–142, 1997.
12. Thomas P. Jensen, Daniel Le Metayer, and Tommy Thorn. Verification of control flow based security properties. In *IEEE Symposium on Security and Privacy*, pages 89–103, 1999.
13. Daniel Hagimont, Jacques Mossiere, Xavier Rousset de Pina, and F. Saunier. Hidden software capabilities. In *International Conference on Distributed Computing Systems*, pages 282–289, 1996.
14. Alan H. Karp, Rajiv Gupta, Guillermo Rozas, and Arindam Banerji. Split capabilities for access control. *HP Laboratories Palo Alto, HPL-2001-164*, 2001.
15. R. S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, 1992.
16. Jonathon Tidswell, Geoffrey H. Outhred, and John Potter. Dynamic rights: Safe extensible access control. In *ACM Workshop on Role-Based Access Control*, pages 113–120, 1999.
17. Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *JACM*, 24(3):455–464, 1977.
18. J. Biskup. Some variants of the take-grant protection model. *Information Processing Letters*, 19(3):151–156, 1984.
19. M. Dacier. A petri net representation of the take-grant model. In *6th IEEE Computer Security Foundations Workshop*, pages 99–108, Franconia, NH, 15-17 June 1993. IEEE Computer Society Press.
20. M. Bishop and L. Snyder. The transfer of information and authority in a protection system. In *Proceedings of the Seventh Symposium in Operating Systems Principles*, pages 45–54, December 1979.
21. L. Snyder. Theft and conspiracy in the take-grant protection model. *Journal of Computer and System Sciences*, 23:333–347, 1981.
22. Matt Bishop. Theft of Information in the Take-Grant Protection Model. Technical Report PCS-TR88-137, Dartmouth College, Computer Science, Hanover, NH, 1988.
23. M. Bishop. Conspiracy and information flow in the take-grant protection model. *Journal of Computer Security*, 4(4):331–359, 1996.
24. Calvin Ko and Timothy Redmond. Noninterference and intrusion detection. In *Proccedings of the IEEE Symposium on Security and Privacy*, 2002.
25. M. Dacier and Y. Deswarte. Privilege graph: an extension to the typed access matrix. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 1994.
26. The reiser4 filesystem - in progress draft document. http://www.namesys.com/v4/v4.html.
27. Fred B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.

28. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
29. CMU CERT/CC. Vu#40327: Openssh uselogin option allows remote execution of commands as root. http://www.kb.cert.org/vuls/id/40327, November 2001.
30. J. Zimmermann, L. Mé, and C. Bidan. Experimenting a policy-based hids based on the reference flow model. Technical Report SSIR-2003-01, http://www.supelec-rennes.fr/rennes/si/equipe/lme/ZMB03.pdf, Supélec, March 2003.

Appendix A

The model published by Ko & Redmond in [24] focuses on noninterference between a group of users G and a piece of data o, noted $G\triangleright \parallel \{o\}$. Integrity policy goals consist in such noninterference requirements, that prevent specific users from modifying critical data. A system trace, as defined by the authors, is a sequence of system state changes where each information modification is performed on behalf of some subject.

We do not define "subjects" explicitly. However, any interaction with users is done through some terminal object, i.e. a *tty* device, a network socket, a removable device etc. Thus, an operation is executed by some user if it involves a flow from this user's terminal object(s).

It is up to the authentication procedure (in the case of Linux, this is done through the PAM authentication system) to create terminal objects and appropriate references. Let us consider a single terminal object per user. For each user u, $dlogin(u)$ is the set of domains the user's terminal object (denoted tu) is bound to. A user login operation is thus modeled as follows:

– $login(u) = \emptyset \gg \{tu.w\}$
– $Create(tu, \ldots) = \{R_d tu.read, R_d tu.write\}, d \in dlogin(u)$

Theorem 51 *Let G be a group of users, o an object, $M_w(o)$ the subset of its methods that change its state and $M_r(o)$ the subset of its methods that read its state. Given an initial system state s_0, the policy $G\triangleright \parallel \{o\}$ is enforced if $\forall u \in G, \forall m \in M_w(o), dlogin(u) \cap dom(o.m, s_0) = \emptyset$. $(*)$*

Proof:

Let us consider a system trace

$$[src_1 \gg dst_1, src_2 \gg dst_2, \ldots, src_n \gg dst_n]$$

1. If $o \notin dst_i$ for any $i \leq n$, then the policy is trivially enforced.
2. Let us suppose that $o \in dst_i$ for some operation i. If $tu \in src_i$, $u \in G$, then the policy is violated if $islegal(src_i \gg dst_i) = true$. Since $(*)$ implies $islegal(src_i \gg dst_i) = false$, the policy is enforced if $(*)$ holds.
3. If $o \in dst_i$ for some operation i and $tu \notin src_i$ for any $u \in G$, then the policy $G\triangleright \parallel \{o\}$ is enforced if $G\triangleright \parallel \{o'_k\}$ is enforced for any o'_k accessed in src_i or $islegal(src_i \gg dst_i) = false$.

(a) If $G \triangleright \parallel \{o'_k\}$ is enforced for any o'_k, then $G \triangleright \parallel \{o\}$ is trivially enforced.
(b) If $G \triangleright \parallel \{o'_k\}$ is not enforced for some o'_k, then, as a result from the *Prop* rule, $dom(src_i, s_i) \subseteq dlogin(u)$ for some $u \in G$. Thus, $(*)$ implies that $islegal(src_i \gg dst_i) = false$, and $G \triangleright \parallel \{o\}$ is thus enforced if $(*)$ holds.

Of course, practical effectiveness of this mechanism requires all considered system calls being accurately modeled in terms of information flows, including parameter passing (for instance, passing a file path parameter to the *open* system call means reading from memory).

Visualisation for Intrusion Detection
Hooking the Worm

Stefan Axelsson

Department of Computer Science
Chalmers University of Technology
Göteborg, Sweden
sax@cs.chalmers.se

Abstract. Even though intrusion detection systems have been studied for a number of years several problems remain; chiefly low detection rates and high false alarm rates.
Instead of building automated alarms that trigger when a computer security violation takes place, we propose to visualise the state of the computer system such that the operator himself can determine whether a violation has taken place. In effect replacing the "burglar alarm" with a "security camera".
In order to illustrate the use of visualisation for intrusion detection purposes, we applied a trellis plot of parallel coordinate visualisations to the log of a small personal web server. The intent was to find patterns of malicious activity from so called *worms*, and to be able to distinguish between them and benign traffic. Several such patterns were found, including one that was unknown at the time to the security community at large.

1 Introduction

Computer security addresses the problem of protecting computers and data against malicious attack. Several forms of protection have been devised. Many of these fall under the guise of separating data and computing resources from potential attack(ers) by devising some sort of perimeter protection.

A fundamentally different approach to the computer security problem is to employ the principle of monitoring. By reason of analogy, the former methods try to erect a fence around the protected resources, and solve the problems of opening the fence in just the right way for those that are to have legitimate access. Monitoring and surveillance, seeks instead to enable the operator to detect malfeasors so that they may be rendered harmless.

This paper explores the possibilities of employing a trellis plot of parallel coordinate visualisations to the log of a small personal web server. The aim being to enable the operator to tell apart the access patterns of automated (or semi automated) attacks and more normal, benign access patterns.

1.1 Intrusion Detection

An intrusion detection system (IDS for short) processes information about the monitored system, and looks for evidence of malicious activity. The information about the system is most often in the form of a log of the system activity. This log does not necessarily have to have been collected solely for security purposes. These logs are typically very large; e.g. Yahoo inc. reports that they collect on the order of five gigabytes of log data per day.

A detector then processes these logs and raise the alarm when there is enough evidence that an intrusion or other computer security violation has taken place. The reader interested in intrusion detection is referred to the extensive literature in the field e.g. [DDW99,Amo99,Axe00b,Bac99].

An important problem with intrusion detection systems today, are a high number of false alarms [Axe00a]. This is perhaps not surprising if we make an analogy with the common burglar alarm. Burglar alarms operate under a very restricted security policy. Any activity whatsoever is suspicious. No door or window can be opened. No suspect heat sources, or movement may be detected, before the alarm is triggered. Intrusion detection systems on the other hand work on the assumption that the computer system is in full operation. Hence a lot of benign activity and processing is taking place. Thus, the analogy with a burglar alarm is not apt.

1.2 Worms

Worms (e.g. [Pfl97, pp. 179,192]) are self replicating programs that attack a remote computer system (often by exploiting some weakness) to gain access. They then transfer a copy of themselves to the subverted system and start running there. Once they are established in the subverted system the cycle begins anew, and the worm starts scanning for other systems to infect.

Worms may or may not carry some sort of payload (logic bomb or otherwise) that perform an additional sinister task. The *Code red* worm [CER01a] for example, launched a denial-of-service attack against a fixed IP address (belonging to the whitehouse.gov web site) on the 20–27 of each month.

Worms have spread far and wide in the last few years, with far reaching consequences. Often they are combined with viruses i.e. the worm has viral properties also, but not always. The names of the most successful worms have percolated up to the common consciousness, with instances reported widely in the press and other news media e.g. the outage of *The New York Times* servers when they were attacked by the Nimda worm [USA01].

Worms often exist in several variants, which are smaller or larger variations on the same basic theme. The Nimda worm is reported in five different variants in the wild [CER01b]. Finding different variants of worms, if and when they occur, is therefore interesting from a security perspective, since they may exploit new vulnerabilities that may yet not have been addressed, and for research into the spread of worms.

1.3 Visualisation for Intrusion Detection

We previously pointed out the flaw in the analogy between the intrusion detection system and the burglar alarm. We propose that a better analogy from the world of physical security is found in the shoplifting scenario. In this scenario an ordinary burglar alarm would not do since there would be a multitude and mass of normal, benign activity taking place in the shop – the shopkeeper even encouraging this – and hence a substantial potential for false alarms. This situation is addressed by performing surveillance – i.e. human supervision – of the potential shoplifters. Often the human is too expensive to employ directly using his senses unaided, and therefore technology is brought to bear in the form of video cameras, video recorders, etc. By reason of analogy then, we wish to build; not a 'burglar alarm for the Internet,' but a 'security camera for the Internet' to coin a phrase.

The security camera analogy taken literally leads to the idea of the application of visualisation to the problem of intrusion detection. We believe this will have a number of desirable effects, mainly that the site does not need as detailed a security policy as with pre-programmed intrusion detection systems (The operator can in effect 'make it up as he goes along.'), and that an understanding of the underlying security principles being violated is furthered.

2 The Monitored System

We have chosen to take the access log file of a small personal web server, that has been continuously available on the Internet for some months. This web server serves a small number of web pages from the home of the author, to the circle of his immediate family and close friends.

This web server is perhaps not representative of the majority of web servers on the Internet in that it requires authentication for all accesses. This would of course make it relatively simple in some sense to sort the illegitimate access attempts from the legitimate ones, but we've chosen not to take the result codes of the operation into account in our visualisation, and therefore feels that the study of such a system could be generalised to include systems which do not require authentication.

A perhaps more significant problem is that the web server does not have much in the way of legitimate traffic. It is small and personal, and is only accessed by a dozen people or so. One could argue that this could make illegitimate accesses stand out more, not having much in the way of legitimate traffic to 'hide' in. Even so, since we are looking for worms that often account for the majority of the traffic on much larger web sites, we still think the study of such a small system worth while, even though it remains to be seen if the results from this study can be generalised to larger systems. It is interesting to note in this context that the accesses patterns on this webserver is similar to what would be seen on a *honey pot* webserver, i.e. a server set up for the sole purpose of drawing attacks to it in order to study them further.

The web server runs thttpd, a small, secure and fast web server written and maintained by Jef Poskanzer, see http://www.acme.com for more information. Thttpd does not at the time of writing have any known security vulnerabilities (and has in fact never had any). The log records (see appendix C for a few examples) from the web server contain the following fields:

IP address. This is the IP-address of the system the request originated from.

Remote username. This is ostensibly the username of the (remote) user that the request originated from. We know of no web browser (or other client) that divulges this information.

Authenticated username. The username the client provides authentication for. Thttpd (and other web servers) provide for authentication in the form of requesting a *username–password* pair from the originating web browser. If the authentication fails the attempted username is not logged, instead this field is left blank.

Time. The time and date the request was received.

Http request. The request string exactly as it was received the client. This is formed by the access method (*GET*, *HEAD*, etc), followed by the URL of the resource the client requested.

Http status code. The status code that was returned to the client. Unfortunately the full list of codes is too long to reproduce here. Noteworthy are the codes: *200* which denotes the successful delivery of the requested page, and *404* which signals the well known 'page not found' error.

Number of bytes. This is the number of bytes that was sent in response to the request (if any). The HTTP response is not included in the count, only the actual page that was sent. Hence this value is blank for all erroneous requests.

Referring URL. If this request resulted from the user clicking a link on another web page, the client has the option of sending the URL of that web page (the 'referring' page) as part of the request. Not all web browsers do this so this information is not always available.

User agent. The name of the client software, if divulged by same. Note that for compatibility reasons many browsers let the user modify the value sent, to be able to masquerade as using another browser than they actually do.

Thus the log contains a maximum of nine degrees of freedom. In our data set this is reduced to eight, since no web client divulged the remote login name of the user, and hence this field is always empty. All the other fields have values for at least a subset of the records.

The log contains some 15000 records, and begins in earnest on 25 Sept 2002. It ends on 1 Jan 2003, and thus covers some three months of activity. As a comparison, the web server log for the computer science department at Chalmers for the month of November 2002 contains on the order of 1.2 million accesses, comprised of circa 200 000 unique requests.

3 Scientific Visualisation

Since computers do not have a visual form – that is interesting for our purposes anyway – we have to find some way of giving our log data ‘body’, as it were. The problem of giving abstract data a tangible visible form is addressed by the field of *information visualisation.* Good introductions to this area can be found in [Spe01] and [CMS99].

Applying information visualisation to the problem of intrusion detection may seem obvious (at least in retrospect) but success is by no means assured. The main problem is one of scale. Most information visualisation techniques cannot in general handle the large amounts of data that we are faced with, at least not in a straight forward manner. On the order of a few thousands of data objects is the norm [Spe01].

Our security log data typically has multiple dimensions with no a priori dependent variables and is therefore multivariate in nature. Spence [Spe01, pp. 45] lists only a handful of methods for the visualisation of multivariate data. Of these we have chosen the tried and tested techniques of the parallel coordinate plot [Ins97] combined with a trellis plot in one variable [Spe01, pp. 168].

The main reasons for choosing parallel coordinate plots over the other methods were:

- The visualisation does not give preference to any dimension at the cost of other dimensions. This is important if we don’t have any indication about which data may be more important from the point of view of making a successful visualisation of the data set.
- Parallel coordinate plots can visualise data with more dimensions than the other methods. It is not unreasonable to visualise data with ten or even twenty dimensions. Most of the other available methods strain when faced with four. As we shall see this turns out to be less important to us as we will reduce the data set to five dimensions. Since our original data set contains eight degrees of freedom, and it wouldn’t be unreasonable to visualise them all in an operational setting, the ability to visualise many dimensions is still an important consideration in choosing the visualisation method.
- The visualisation methods lends itself to trellis plots, i.e. where we make a plot of plots (see below), in one of the variables. This is an effective method in seeing trends in higher dimensional data, when one variable has been singled out, in our case to group the requests. Not all the other methods lend themselves to trellis plots as well as the parallel coordinate plot, if at all.
- The parallel coordinate plot is very generally applicable; it can handle both continuous and categorical data (though admittedly some of the key benefits are lost then) and it can be used to both see statistical correlations between data points, and as a more general profile plot, where the appearance of the graph itself (rather than the curve forms as in an ordinary x-y scatter plot) can be used to identify features. We will use the parallel coordinate plot in the latter capacity.

– The last, but by no means the least, of our considerations is that the parallel coordinate plot is well researched and the theory around it somewhat more mature than is true of many of the alternatives.

3.1 The Parallel Coordinate Plot

We have used the tool Spotfire to make the parallel coordinate plot, more information can be found at http://www.spotfire.com. A parallel coordinate plot is prepared by taking each data point and projecting it as a line joining the components of the vector onto a set of parallel coordinate axes. This form of visualisation does not only let the viewer learn about the dependencies between the variables, but also lets the viewer quickly see emerging patterns, and compare different datasets for similarities and differences [Ins97].

Figure 1 illustrates the case where 68 different points in eight dimensional space have been mapped onto (eight) parallel coordinate axes. In this case the dataset was chosen by limiting the log file from our webserver to the first 68 data points. We choose the eight dimensions that had data. They are in order in the figure:

Date The date and time the request was made. In this case the data has been limited by selection, and hence the available dates only cover a small percentage of the axis. Spotfire doesn't rescale the axis when the visible data is limited in this way, to preclude the effect where abrupt scale changes makes the user lose his orientation as different ranges of data are selected.
Note the *date* slider, in the *query devices* sidebar, that has been manipulated to select the narrow range of records that are displayed.

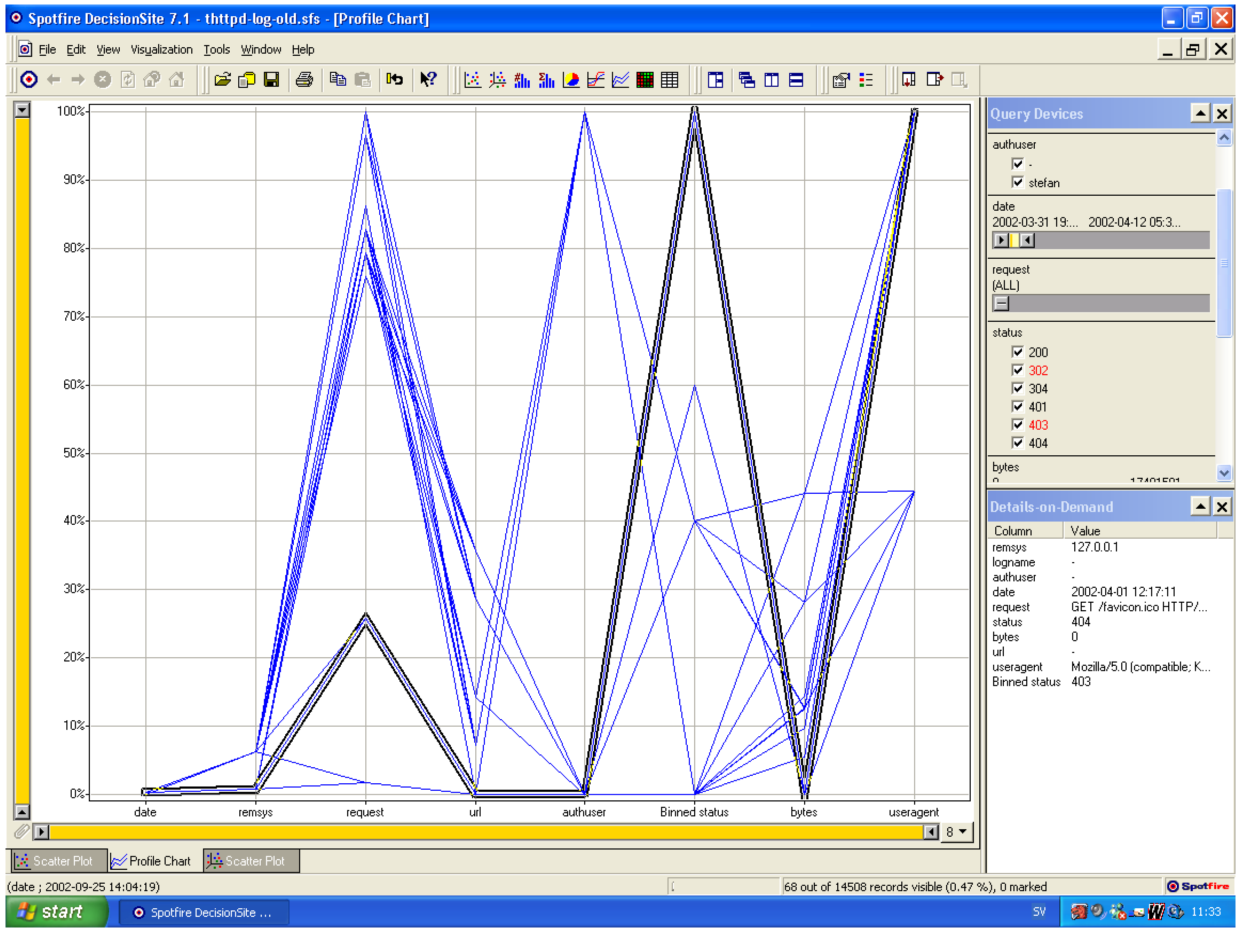
Remsys/Request/Url This variables are imported as strings, lexicographically sorted, and plotted as categorical data.

Authuser The username of an authenticated user, or the minus sign if no authentication has been performed. Note the check boxes corresponding to the two possible values in this dataset at the very top of the *query devices* side bar.

Binned status The status (e.g. *404–not found*) that the request resulted in. This data was imported as integers. However the result codes aren't integral per se – i.e. magnitude comparisons between the result codes *200* and *404* interpreted as the integers 200 and 404 respectively, makes little sense. Instead the result codes have been rescaled by being sorted into bins chosen so that each result code ended up in a bin of its own. This has transformed the integral data into categorical data which is plotted with each bin equidistant on the vertical axis.

Bytes The number of bytes that were sent in response to the request. Imported as integers, and in this case the data has a magnitude as opposed to the *status* data. Spotfire has the ability to logscale such data (and many other possible scalings can be applied as well) should the user so chose.

Useragent Typically the browser that made the request, if that information has been divulged. Imported as a string and hence treated as lexicographically sorted categorical data.



Column	Value
remsys	127.0.0.1
logname	-
authuser	-
date	2002-04-01 12:17:11
request	GET /favicon.ico HTTP/...
status	404
bytes	0
url	-
useragent	Mozilla/5.0 (compatible; K...
Binned status	403

(date ; 2002-09-25 14:04:19)

68 out of 14508 records visible (0.47 %), 0 marked

Fig. 1. A simple parallel coordinate plot

For the purpose of this paper we will not use the possibility of finding correlations between data points using the parallel coordinate plot directly. However, just to hint at the possibilities; in figure 1 we see a strong inverse correlation between the result code (*binned status*) and the *auth user* field, indicating that the lack of authentication leads to "access denied" types of error codes, and vice versa. This is not surprising given that we've already stated that the webserver was configured to use authentication for all accesses. However, a similar plot was used by the author to discover that the access controls had been misconfigured at one time (after having upgraded the webserver) giving access to the world.

It should be noted that while Spotfire does not mark the axes with any label indicating the range in absolute terms (only relative) hovering the mouse over any specific value displays the coordinate of the axis, as well as all coordinates for the closest record. Data points can also be marked and the data for the corresponding records displayed in the sidebar *details-on-demand.* Here we've marked one record (indicated by the thicker lines surrounding it) and the corresponding record is visible in the lower right corner of figure 1.

Another sidebar (*query devices*) allows the user to dynamically adjust the data that is displayed, e.g. a continuous variable can have a slider (or other user selectable input element) that lets the user select the relevant parts of the data set for display. This sidebar is in the upper right corner in the figure. In this case it displays three examples of query devices: *check boxes*, a *range slider* and an *item slider.*

As these are dynamic user interface properties, the awareness of the data this lends the user is of course difficult to give justice to in a paper.

3.2 The Trellis Plot

The *trellis plot* (or *prosection matrix* in the case of continuous data) is described in [Spe01, pp. 168]. It was originally used as a way of extending two dimensional scatter plots into three dimensions, without introducing a 3D view and all the complications that follow. Instead a pair of the variables is singled out and a plot of subplots is made, typically arranged in a triangular matrix, where the axes represent the different possible choices for the pair of parameters, and the x–y position in the matrix contains a plot of the other parameters for the x–y value of the singled out parameters. In the case of continuous data, it is of course not possible to make a continuum of of subplots. Instead a discrete number of ranges of values of the pair of parameters is chosen and the corresponding subplots made.

In our case, we will chose only one variable, not a pair. We will single out the request string, which is already a categorical entity. Since we only make one parameter choice, the x–y position of the subplot within the trellis plot will not carry any further information – conceptually the subplots would be laid out in a linear fashion one after another – but are laid out on the plane so as to use screen real estate effectively. By doing the trellis plot this way we hope to find similarities in access patterns corresponding to *different* requests, and hence being able to visually cluster them corresponding to the entities (human or worm) that lie behind the particular requests in that cluster. This is also how we use the parallel coordinate plot as a profile plot. It is the profiles ('blobs' if you will) that exhibit similarities between the different subplots and we will use these similarities to group the requests into clusters.

4 Visual Analysis of the Log File

The aim of this investigation is to find support for the hypotheses that the web server was targeted by some of the more popular worms (that attacked web servers) during the period in question. We would also like to be able to tell apart the different types of worms, if any, and also to differentiate between the access patterns made by worms, and those made by others, i.e. more or less ordinary users. We intend to perform this by correlating different requests with each other and see if they cluster into patterns of activity that can be distinguished from each other, both in terms of distinguishing benign accesses from malicious, but also the various malicious accesses from each other.

We feel justified in assuming that such differences will be present, e.g. Feinstein et. al. [FSBK03] reports differences in the distributions of source IP addresses between automated denial-of-service type attacks and benign traffic. These differences are significant and enables the attacked site (or intervening router) to differentiate between these types of traffic based on source IP address alone.

Since our web server requires authentication it would be natural to divide access into *allowed* and *denied* as a first step, but as we have mentioned earlier; since most web servers aren't configured this way, we'll refrain from using this data. Furthermore, since we don't see how this would allow us to tell different *kinds* of worms apart, another approach is necessary.

Attack of web servers typically have to do with exploiting weaknesses in the web servers handling of input data, either by having the server misinterpret it, doing something the designers never intended, or by triggering a buffer overrun, thereby allowing the attacker to take control at a lower level. Since the only real source of input data is the *request* the web client (browser/worm) makes, it is (as we mentioned earlier) natural to make the visualisation pivot on the request. We've already mentioned that one way of accomplishing this – using Spotfires parallel coordinate plot – is to make a trellis plot of the log data, with the request being the controlling variable.

In figure 2, a specific parallel coordinate plot has been made for each of the unique request strings (59 in total), i.e. the request string has been held constant, and all data pertaining to other requests have been filtered out in each of the 59 plots. As a consequence the request string was removed from the subplots themselves as it wouldn't add any useful information. In fact it would have detracted from the similarities of the plots since it would have been different for each of the subplots.

In order not to taint the results of the investigation with data that pertains to the success or failure of authentication, we've reduced the dataset to the following four variables: *Date*, *Remsys*, *Url* and *Useragent*.

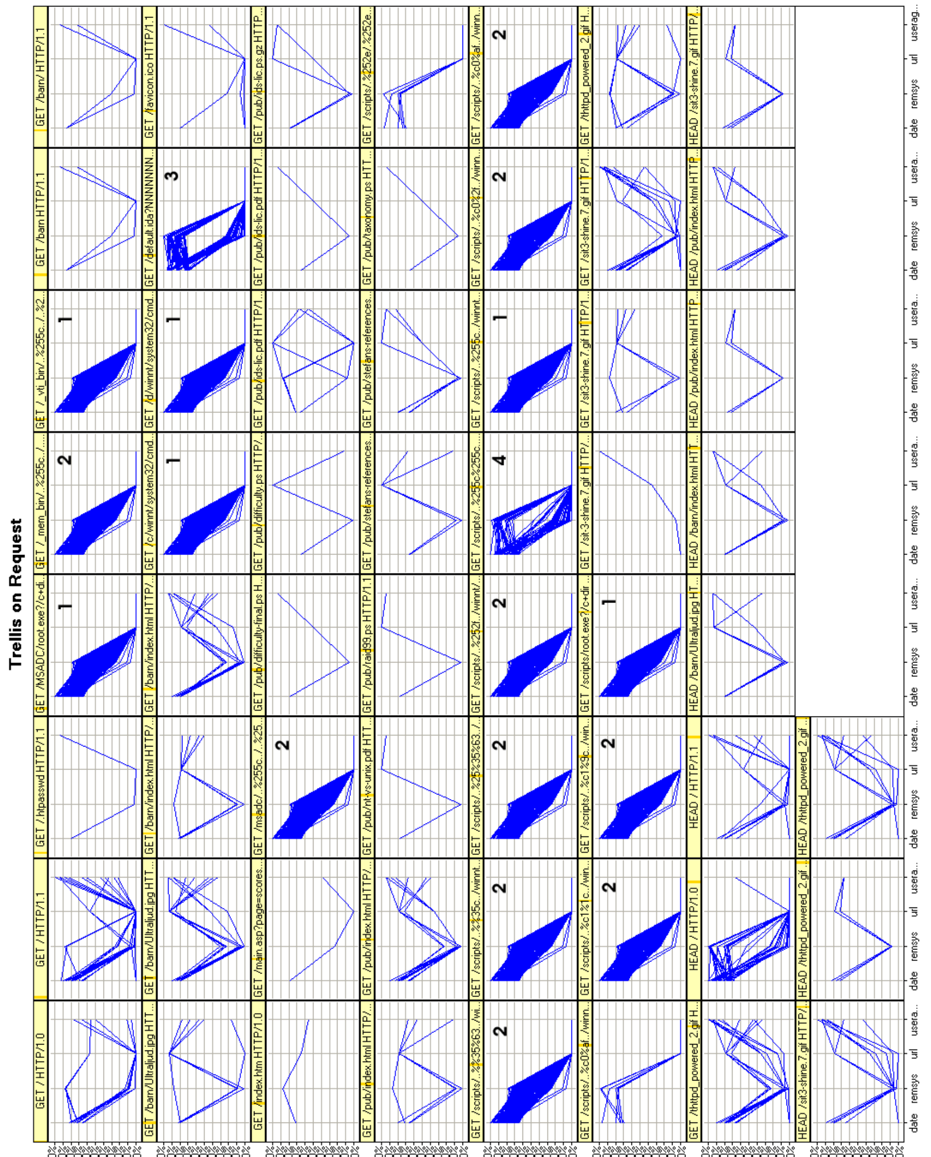


Fig. 2. A trellis of parallel coordinate plots

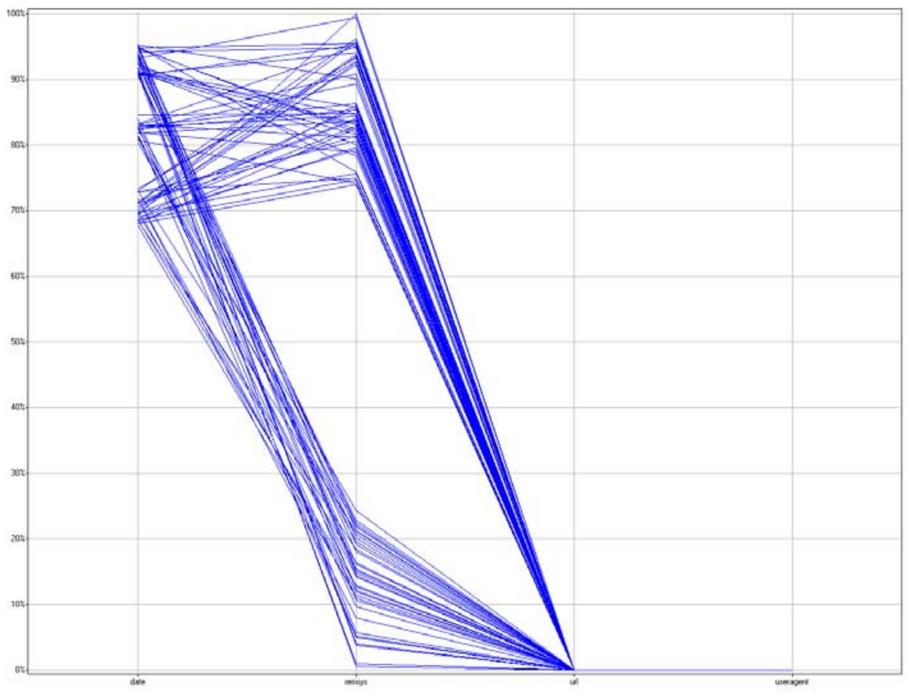


Fig. 3. A plot of the “Code-red” worm access pattern

The variables *Authuser*, *Status* and *Bytes* had to be removed since they tainted the experiment by leaking information (directly or indirectly) about the success or failure of the authentication.

Removing data from the visualisation actually strengthens the results of the experiment in that we remove security relevant information, making the task more difficult. In an operational setting we most likely would not perform this reduction.

To illustrate the concept in greater detail; figure 3 is an enlargement of the plot marked ‘3’ in figure 2. The axes are labelled with the respective variable.

5 Results of the Investigation

Even a quick glance at figure 2 reveals four interesting patterns (marked 1–4 in the figure). Looking closer at the individual plots we can see that they originate from a large number of systems, and dates. They are also comprised of a lot of repetitive accesses (i.e. a large number of records). Other patterns can also be identified. They look markedly different even ignoring the fact that they are comprised of much fewer accesses. Even without knowing the contents of the web site, it’s is difficult to believe that there would be four sets of pages that would give rise to such access patterns. Indeed, the four patterns are quite distinct when compared to the rest of the subplots of the trellis plot. If we draw on knowledge about the contents of the website the game is immediately up.

There are one or two different additional suspicious looking candidates but viewing the request themselves gives that game away. It is not unreasonable to view the requests to eliminate suspects; we envision this method as mainly useful for the system administrator of the site, i.e. someone that is familiar with the site and manner in which it is used. Indeed someone who was not familiar with the site could not make the sort of security policy decision on the spot that we alluded to in the introduction to the paper.

The four suspicious request patterns may be indicative of worm activity and merit further investigation. Dissecting the clusters with regards to the number of different requests results in:

Pattern 1 Six different requests.
Pattern 2 Ten different requests.
Pattern 3 One request.
Pattern 4 One request.

Viewing the access requests themselves (we've refrained from a detailed listing of all of them here to save space, but appendix A lists the six requests of the first pattern) and searching computer security sources we find evidence of two different instances of Nimda [CER01b] in patterns one and two. These two worms seems to have invaded the same types of systems, given the similarities in the IP-address ranges they originate from. Nimda is interesting in that it attacks web servers (Microsoft IIS) by either scanning for back doors left by other successful worms (Code red), or by so called *Unicode attacks*, where the URL is modified (certain characters escaped) to avoid subroutines in the web server that clean the request of certain 'dangerous' characters e.g. characters that modify the search path to requested resource.

The third pattern consists of only one type of access. It was found equally quickly in the literature, since it consists of one very long access request, designed to overflow a buffer in IIS [CER01a]. The Code red worm does not probe for a wide variety of weaknesses in IIS, as Nimda does, relying solely on the one buffer overflow to gain entrance.

Comparing these two worms is illustrative in that we see a marked difference in the range of IP-addresses of infected systems. We presume that this is because Nimda can infect not only web servers, but also web clients (Microsoft Internet Explorer 5.0) – when the user visits a subverted web page, or be attached as an email virus. Thus home users, who presumably does not run web servers on their home computers frequently, are susceptible to infection by Nimda. Nimda then goes on to spread through all available means, including scanning nearby IP segments for vulnerable web servers and hence end up in our logs.

Code red, on the other hand, relies solely on web server infection, and hence will infect other IP address ranges, that have not been reserved by Internet service providers who cater predominantly to the home users. Therefore we see such a marked difference in the access pattern.

Pattern four is the *pièce de résistance* of the investigation. Neither the literature nor any of the major security information sources on the Internet at the time of investigation listed this particular access request in conjunction with any known worm. They did list several similar access requests (as they are part of larger class of IIS vulnerabilities), and indeed had we only looked at the access requests themselves – see appendix B – we might have missed that pattern four is different from the Nimda patterns since the request strings themselves look strikingly similar. The total number of accesses was small; only 71 access requests (about on the same order as for Code red, and an order of magnitude less than Nimda). Another odd characteristic was that they were not repeated from the same system twice (with one or two exceptions). We concluded that we were either dealing with a careful worm writer, or perhaps not even a worm at all, but rather the activities of so called *script kiddies*, who follow a pre made recipe found somewhere on the Internet 'underground.' As far as we could tell this was a new type of worm, or a new type of intrusive activity.

In preparing this paper a very detailed search of the available information sources revealed that these access requests had in fact been spotted earlier and identified as being the product of the manual application of the 'cracking tool' `sfind.exe` [Jel02]. This tool is employed by the *pubstro* movement [Bra], that break into web and file servers in order to build a network of computers with which to distribute software copied in breach of copyright (so called *warez*). We were justified in our observation that the access requests were very similar to Nimda and may have been mistaken as such had we only looked at the access requests in isolation. Indeed many system administrators let this particular activity go unnoticed mistaking it for an instance of Nimda [Jel02].

6 Discussion

A sticking point in discussing the applicability of these results is the issue of scalability of the method demonstrated here. The log file we investigated has only 59 unique requests, and as earlier pointed out; more realistic log files from larger installations can contain as much as 200 000 unique requests. The method of inspecting the requests directly, using a trellis plot, as we have done here is unfeasible when the number of unique requests is as large as 200 000. We conjecture that a data set of not much more than on the order of 100 unique requests could be investigated using the method developed here.

We see two ways of addressing this problem to enable this method to scale to larger data sets.

The first is to reduce the number of requests before applying the visualisation. One method of doing so could be to apply some form of anomaly detection to the requests, sorting out unusual requests, and then visualising the output from the anomaly detection system to. This would allow us to tell apart the malicious accesses from the inevitable false alarms that would creep into the output from the anomaly detection system. An advantage of such an approach [Axe00a] is that it would allow the anomaly detection system to be tuned so that the number of false alarms were relatively high – and hence the likelihood of detection would be correspondingly higher – since we would not observe the output directly, but use it as input for our visualisation method.

The other approach (perhaps most realistically used in conjunction with the first) is to reduce the number of unique requests by doing stepwise elimination by first identifying two or more similar patterns and then combining them into one pattern (sub plot). Hereby iteratively reducing the number of subplots until it is small enough that an overview of all of them can be made, analogous with what we have performed in this experiment.

7 Conclusion

We have demonstrated that worm activity can be detected and analysed by applying a trellis plot of parallel coordinate visualisations on the log of a small web server. The different requests made by worms can be correlated to the particular type of worm making the requests. Furthermore, the clusters formed by worm requests are markedly different from the clusters formed by benign requests. The visualisation was successful even though the number of data points visualised was larger than what is generally considered the limit for such methods.

Four different worm (or worm like) activities were found. Two of these were found to be indicative of the Nimda worm, one of the Code red worm, and the last of a then largely unknown malicious activity, later identified as emanating from the manual application of the tool `sfind.exe`.

8 Related Work

There have been a few attempts at bringing visualisation to the field of intrusion detection. The first instance of mention in the literature is [VFM98]. This work has the nature of a position paper on a theoretical model for visualising the security state of a system. No argument is presented how this should be accomplished (i.e. the security state of the system be determined), instead the proposed the visual model (named spicule) is presented and developed.

More recently Erbacher et. al. [EWF02] have presented work building on the previous work by Frincke et. al. [FTM98]. This work is based on encoding information about network traffic and alarms from a network of intrusion detection sensors as glyphs onto a stylised map of the network. As such their approach is very different from ours, in that we don't map the traffic as such, but rather try and visualise meta data from the application layer in a manner that makes the dependencies between the underlying variables apparent.

A different tack from that of Erbacher was taken by Theo et. al. [TMWZ02]. They visualise communication between pairs of routers on the Internet using the BGP (Border Gateway Protocol) routing protocol. Their choice of problem and visualisation techniques are different from ours, and they do not delve as deeply into the analysis of the security problems they detect (they are not as clearly *security problems* as ours), but they do visualise a greater amount of data more compactly than we do and still manage to detect anomalies in the BGP traffic.

9 Future Work

This investigation has really only scratched the surface of both what security relevant information is hiding in data sets such as this and of what visualisation in general, and how parallel coordinate plots in particular can be brought to bear to extract it. Also, the data set itself could be expanded with more realistic traffic, and more advanced simulated attacks.

Acknowledgements

We would like to thank Spotfire inc for letting us use 'Spotfire Decision Site'. The valuable discussions with, and support from, my supervisor Prof. David Sands, and my research colleague Dr. Rogardt Heldal is also gratefully acknowledged.

We would also like to thank the anonymous reviewers and in particular Prof. John McHugh for their helpful comments and support.

References

Amo99. Edward Amoroso. *Intrusion Detection.* intrusion.net books, P.O. Box 78, Sparta, New Jersey 07871, 1999. http://www.intrusion.net.

Axe00a. Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

Axe00b. Stefan Axelsson. Intrusion-detection systems: A Taxonomy and Survey. Technical Report 99–15, Department of Computer Engineering, Chalmers University of Technology, SE–412 96, Göteborg, Sweden, March 2000.

Bac99. Rebecca Bace. *Intrusion Detection.* Macmillan Technical Publishing, first edition, December 1999. ISBN 1-57-870185-6.

Bra. Richard Braithwaite. The 'pubstro' Phenomenon: Robin Hoods of the Internet. Avaliable as: http://www.deakin.edu.au/infosys/docs/seminars/-handouts/RichardBraithwaite.pdf Verified: 2003–07–24.

CER01a. CERT Advisory CA-2001-19 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. CERT Advisory by CERT/CC, Email: cert@cert.org, CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213-3890, U.S.A., 19 July revised 17 January 2001. http://www.cert.org.

CER01b. CERT Advisory CA-2001-26 Nimda Worm. CERT advisory by CERT/CC, Email: cert@cert.org, CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213-3890, U.S.A., 18 September revised 25 September 2001. http://www.cert.org.

CMS99. Stuart K. Card, Jock D. MacKinlay, and Ben Shneiderman. *Readings in Information Visualization – Using Vision to Think.* Series in Interactive Technologies. Morgan Kaufmann, Morgan Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Fransisco, CA 94104-3205, USA, first edition, 1999. ISBN 1-55860-533-9.

DDW99. Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a Taxonomy of Intrusion-Detection Systems. *Computer Networks*, 31(8):805–822, April 1999.

EWF02. Robert F. Erbacher, Kenneth L. Walker, and Deborah A. Frincke. Intrusion and Misuse Detection in Large-Scale Systems. *Computer Graphics and Applications*, 22(1):38–48, January 2002.

FSBK03. Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical Approaches to DDoS Attack Detection and Response. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, page 303. IEEE Computer Society, IEEE, 22–24 April 2003.

FTM98. Deborah A. Frincke, Donald L. Tobin, and Jesse C. McConnell. Research Issues in Cooperative Intrusion Detection Between Multiple Domains. In *Proceedings of Recent advances in intrusion detetection RAID'98*, 1998.

Ins97. Alfred Inselberg. Multidimensional Detective. In *Proceedings of InfoVis'97, IEEE Symposium on Information Visualization*, pages 100–107. IEEE Information visualisation, IEEE, 1997.

Jel02. Peter Jelver. Pubstro-hacking – Systematic Establishment of Warez Servers on Windows Internet Servers. Avaliable as: http://www.esec.dk/pubstro.pdf Verified: 2003–07–24, 23 July 2002.

Pfl97. Charles P. Pfleeger. *Security in Computing*. Prentice Hall, second edition, 1997. ISBN 0-13-185794-0.

Spe01. Robert Spence. *Information Visualization*. ACM Press Books, Pearson education ltd., Edinburgh Gate, Harlow, Essex CM20 2JE, England, first edition, 2001. ISBN 0-201-59626-1.

TMWZ02. Soon Tee Teoh, Kwan-Liu Ma, S. Felix Wu, and Xiaoliang Zhao. Case Study: Interactive Visualization for Internet Security. In *Proceedings of IEEE Visualization 2002*, The Boston Park Plaza hotel, Boston, Massachusetts, USA, 27 October to 1 November 2002. IEEE Computer society.

USA01. 'Ny Times' Outage Caused by Nimda virus. http://www.usatoday.com/-tech/news/2001/11/01/new-york-times-outage.htm, 1 December 2001. Verified 2003–04–11.

VFM98. Greg Vert, Deborah A. Frincke, and Jesse C. McConnell. A Visual Mathematical Model for Intrusion Detection. In *Proceedings of the 21st National Information Systems Security Conference*, Crystal City, Arlington, VA, USA, 5–8 October 1998. NIST, National Institute of Standards and Technology/National Computer Security Center.

A Nimda Example

These are the six different requests made by pattern number one in figure 2:

```
"GET /MSADC/root.exe?/c+dir HTTP/1.0"
"GET /_vti_bin/..%255c../..%255c../..%255c../winnt/system32/
     cmd.exe?/c+dir HTTP/1.0"
"GET /c/winnt/system32/cmd.exe?/c+dir HTTP/1.0"
"GET /d/winnt/system32/cmd.exe?/c+dir HTTP/1.0"
"GET /scripts/root.exe?/c+dir HTTP/1.0"
"GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir HTTP/1.0"
```

B Requests Made by the Elusive Fourth Class

This is the request made by the elusive pattern number four in figure 2. Compare the similarity with the last request made in appendix A.

```
"GET /scripts/..%255c%255c../winnt/system32/cmd.exe?/c+dir HTTP/0.9"
```

C Logfile Example

This section contains a few records of the webserver log file:

```
213.37.31.61 - - [25/Sep/2002:17:01:56 +0200] "GET /scripts/..%%35c../
        winnt/system32/cmd.exe?/c+dir HTTP/1.0" 404 - "" ""
172.16.0.3 - stefan [25/Sep/2002:19:28:28 +0200] "HEAD /sit3-shine.7.gif
        HTTP/1.1" 304 2936 "http://server/" "Mozilla/5.0 (X11; U;
        Linux i686; en-US; rv:1.1) Gecko/20020827"
172.16.0.3 - - [25/Sep/2002:19:28:46 +0200] "GET /pub/ids-lic.pdf HTTP/1.1"
        200 615566 "http://server/pub/index.html" "Mozilla/5.0 (X11; U;
        Linux i686; en-US; rv:1.1) Gecko/20020827"
213.64.153.92 - - [25/Sep/2002:22:57:51 +0200] "GET /scripts/root.exe?/c+dir
        HTTP/1.0" 404 - "" ""
```

On the Detection of Anomalous System Call Arguments

Christopher Kruegel, Darren Mutz, Fredrik Valeur, and Giovanni Vigna

Reliable Software Group*
Department of Computer Science
University of California, Santa Barbara
{chris,dhm,fredrik,vigna}@cs.ucsb.edu

Abstract. Learning-based anomaly detection systems build models of the expected behavior of applications by analyzing events that are generated during their normal operation. Once these models have been established, subsequent events are analyzed to identify deviations, given the assumption that anomalies usually represent evidence of an attack. Host-based anomaly detection systems often rely on system call traces to build models and perform intrusion detection. Recently, these systems have been criticized, and it has been shown how detection can be evaded by executing an attack using a carefully crafted exploit. This weakness is caused by the fact that existing models do not take into account all available features of system calls. In particular, some attacks will go undetected because the models do not make use of system call arguments. To solve this problem, we have developed an anomaly detection technique that utilizes the information contained in these parameters. Based on our approach, we developed a host-based intrusion detection system that identifies attacks using a composition of various anomaly metrics.
This paper presents our detection techniques and the tool based on them. The experimental evaluation shows that it is possible to increase both the effectiveness and the precision of the detection process compared to previous approaches. Nevertheless, the system imposes only minimal overhead.

Keywords: Intrusion detection, anomaly models, system calls.

1 Introduction

Intrusion detection techniques have traditionally been classified as either *misuse-based* or *anomaly-based*.

Systems that use misuse-based techniques [7,17,18] contain a number of attack descriptions, or signatures, that are matched against a stream of audit data looking for evidence of modeled attacks. These systems are usually efficient and generate few erroneous detections, called false positives. The main disadvantage of misuse-based techniques is the fact that they can only detect those attacks

* This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484.

that have been specified previously. That is, they cannot detect intrusions for which they do not have a signature.

Anomaly-based techniques [6,9,12] follow an approach that is complementary to misuse detection. In their case, detection is based on models of normal behavior of users and applications, called 'profiles'. Any deviations from such established profiles are interpreted as attacks. The main advantage of anomaly-based techniques is that they are able to identify previously unknown attacks. By defining an expected, normal state, any abnormal behavior can be detected, whether it is part of the threat model or not. The advantage of being able to detect previously unknown attacks is usually paid for with a high number of false positives.

In the past, a number of host-based anomaly detection approaches have been proposed that build profiles using system calls [8,25]. More specifically, these systems rely on models of legitimate system call sequences issued by the application during normal operation. During the detection process, every monitored sequence that is not compliant with previously established profiles is considered part of an attack.

Recent research [23,24,26] has examined techniques to evade this type of detection by using mimicry attacks. Mimicry attacks operate by crafting the injected malicious code in a way that imitates (or mimics) a legitimate system call sequence. The results show that many models can be easily bypassed.

The weakness of existing systems is mostly due to the lack of comprehensive models that take advantage of the information contained in system call traces. In particular, satisfactory machine generated models for system call arguments have not been developed because the problem has been considered either too difficult or too expensive computationally.

This paper presents a novel anomaly detection technique that takes advantage of the information contained in system calls by performing an analysis of their arguments. We present several models to derive profiles for different types of arguments and a mechanism to evaluate a system call by composing the results delivered by different anomaly metrics for each of the system call parameters. In addition, we describe a host-based intrusion detection tool that implements our approach. A qualitative analysis shows that the system is capable of performing effective detection while a quantitative evaluation confirms that it is very efficient.

2 Related Work

Many different anomaly detection techniques have been developed that gather input from a variety of sources. Examples include data mining on network traffic [16], statistical analysis of audit records [11], and the analysis of operating system call sequences [8]. As our approach is based on system calls, work in this area is particularly relevant. Previously presented research falls into the areas of specification-based and learning-based approaches.

Specification-based techniques rely on application-specific models that are either written manually (e.g., [12], [3], [5]) or derived using static program analysis techniques (e.g., [25]). [10] and [19] describe systems that interactively create application-specific profiles with the help of the user. The profiles are then used as the input to a real-time intrusion detection system that monitors the corresponding application. When a non-conforming system call invocation is detected, an alarm is raised.

A major problem of specification-based systems is the fact that they exhibit only a very limited capability for generalizing from written or derived specifications. An additional disadvantage of hand-written specification-based models is the need for human interaction during the training phase. Although it is possible to include predefined models for popular applications, these might not be suitable for every user, especially when different application configurations are used. Systems that use automatically generated specifications often suffer from significant processing overhead. This is caused by the complexity of the underlying models. Another drawback is the fact that previously presented approaches do not take into account system call arguments, unless they are constants or can be easily determined by data flow analysis [25].

Learning-based techniques do not rely on any *a priori* assumptions about the applications. Instead, profiles are built by analyzing system call invocations during normal execution. An example of this approach is presented by Forrest [8]. During the training phase, the system collects all distinct system call sequences of a certain specified length. During detection, all actual system call sequences are compared to the set of legitimate ones, raising an alarm if no match is found. This approach has been further refined in [15] and [27], where the authors study similar models and compare their effectiveness to the original technique. However, these models suffer from the limitation that information about system call arguments is discarded.

To mitigate this weakness, we propose a learning-based technique that focuses on the analysis of system call arguments. By doing so, it is possible to considerably reduce the ability of an attacker to evade detection. We propose to use different models that examine different features of the arguments of a system call. This allows anyone to easily extend our system by introducing new models. To assess an entire system call, the results of the models are combined into a single anomaly score.

3 Design

Our anomaly detection mechanism is based on the application-specific analysis of individual system calls. The input to the detection process consists of an ordered stream $S = \{s_1, s_2, \dots\}$ of system call invocations recorded by the operating system. Every system call invocation $s \in S$ has a return value r^s and a list of argument values $< a_1^s, \dots, a_n^s >$. We do not take into account relationships between system calls or sequences of invocations.

For each system call used by an application, a distinct profile is created. For example, for the `sendmail` application the system builds a profile for each of the system calls invoked by `sendmail`, such as `read`, `write`, `exec`, etc. Each of these profiles captures the notion of a 'normal' system call invocation by characterizing 'normal' values for one or more of its arguments.

The expected ‘normal’ values for individual parameters are determined with the help of models. A model is a set of procedures used to evaluate a certain feature of a system call argument, such as the length of a string.

A model can operate in one of two modes, learning or detection. In learning mode, the model is trained and the notion of ‘normality’ is developed by inspecting examples. Examples are values which are considered part of a regular execution of a program and are either derived directly from a subset of the input set S (learning on-the-fly) or provided by previous program executions (learning from a training set). It is important that the input to the training phase is as exhaustive and free from anomalous system calls as possible, although some models exhibit a certain degree of robustness against polluted or incomplete training data. The gathering of quality training data is a difficult problem by itself and is not discussed in this paper. We assume that a set of system calls is available that was created during normal operation of the program under surveillance. Section 6 describes how we obtained the training data for our experiments.

In detection mode, the task of a model is to return the probability of occurrence of a system call argument value based on the model’s prior training phase. This value reflects the likelihood that a certain feature value is observed, given the established profile. The assumption is that feature values with a sufficiently low probability (i.e., abnormal values) indicate a potential attack. To evaluate the overall anomaly score of an entire system call, the probability values of all models are aggregated.

Note that anomaly detection is performed separately for each program. This means that different profiles are created for the same system calls when they are performed by different applications. Although the same models are used to examine the parameters of identical system calls, they are instantiated multiple times and can differ significantly in their notion of ‘normality’.

There are two main assumptions underlying our approach. The first is that attacks will appear in the arguments of system calls. If an attack can be carried out without performing system call invocations or without affecting the value of the parameters of such invocations, then our technique will not detect it. The second assumption is that the system call parameters used in the execution of an attack differ substantially from the values used during the normal execution of an application. If an attack can be carried out using system call parameter values that are indistinguishable from the values used during normal execution then the attack will not be detected. The ability to identify abnormal values depends on the effectiveness and sophistication of the models used to build profiles for the system call features. Good models should make it extremely difficult to perform an attack without being detected.

Given the two assumptions above, we developed a number of models to characterize the features of system calls. We used these models to analyze attack data that escaped detection in previous approaches, data that was used in one of the most well-known intrusion detection evaluations [13], as well as data collected on a real Internet server. In all cases, our assumptions proved to be reasonable and the approach delivered promising results.

4 Models

The following section introduces the models that are used to characterize system call arguments and to identify anomalous occurrences. For each model, we describe the creation process (i.e., the learning phase) and explain the mechanism to derive a probability for an argument value (i.e., the detection phase). This probability is then used to obtain an anomaly score for the corresponding argument.

4.1 String Length

Usually, system call string arguments represent canonical file names that point to an entry in the file system. These arguments are commonly used when files are accessed (`open`, `stat`) or executed (`execve`). Their length rarely exceeds a hundred characters and they mostly consist of human-readable characters.

When malicious input is passed to programs, it is often the case that this input also appears in arguments of system calls. For example, consider a format string vulnerability in the log function of an application. Assume further that a failed open call is logged together with the file name. To exploit this kind of flaw, an attacker has to carefully craft a file name that triggers the format string vulnerability when the application attempts and fails to open the corresponding file. In this case, the exploit code manifests itself as an argument to the `open` call that contains a string with a length of several hundred bytes.

Learning. The goal of this model is to approximate the actual but unknown distribution of the lengths of a string argument and detect instances that significantly deviate from the observed normal behavior. Clearly, we cannot expect that the probability density function of the underlying real distribution would follow a smooth curve. We also have to assume that it has a large variance. Nevertheless, the model should be able to identify obvious deviations.

We approximate the mean $\dot{\mu}$ and the variance $\dot{\sigma}^2$ of the real string length distribution by calculating the sample mean μ and the sample variance σ^2 for the lengths $l_1, l_2, \ldots, l_n$ of the argument strings processed during the learning phase.

Detection. Given the estimated string length distribution with parameters μ and σ^2, it is the task of the detection phase to assess the regularity of an argument string with length l. The probability of l is calculated using the Chebyshev inequality [4].

The Chebyshev inequality puts an upper bound on the probability that the difference between the value of a random variable x and μ exceeds a certain threshold t, for an arbitrary distribution with variance σ^2 and mean μ. Note that although this upper bound is symmetric around the mean, the underlying distribution is not restricted (and our data shows that the string length was not symmetric in the experiments). To obtain an upper bound on the probability that the length of a string deviates more from the mean than the current instance, the threshold t is substituted with the distance between the string length l of the current instance and the mean μ of the string length distribution.

Only strings with lengths that exceed μ are assumed to be malicious. This is reflected in our probability calculation as only the upper bound for strings that are longer than the mean is relevant. Note that an attacker cannot disguise malicious input by padding the string and thus increasing its length, because an increase in length can only reduce the probability value.

The probability value $p(l)$ for a string with length l, given that $l \geq \mu$, is calculated as shown below. For strings shorter than μ, $p(l) = 1$.

$$p(l) = p(|x - \mu| > |l - \mu|) = \frac{\sigma^2}{(l - \mu)^2} \tag{1}$$

We chose the Chebyshev inequality as a reasonable and efficient metric to model decreasing probabilities for strings with lengths that increasingly exceed the mean. In contrast to schemes that define a valid interval (e.g., by recording all strings encountered during the training phase), the Chebyshev inequality takes the variance of the data into account and provides the advantage of gradually changing probability values (instead of a simple 'yes/no' decision).

4.2 String Character Distribution

The string character distribution model captures the concept of a 'normal' or 'regular' string argument by looking at its character distribution. The approach is based on the observation that strings have a regular structure, are mostly human-readable, and almost always contain only printable characters.

A large percentage of characters in such strings are drawn from a small subset of the 256 possible 8-bit values (mainly from letters, numbers, and a few special characters). As in English text, the characters are not uniformly distributed, but occur with different frequencies. However, there are similarities between the character frequencies of parameters of legitimate system calls. This becomes apparent when the relative frequencies of all possible 256 characters are sorted in descending order.

Our algorithm is based only on the frequency values themselves and does not rely on the distributions of particular characters. That is, it does not matter whether the character with the most occurrences is an '`a`' or a '`/`'. In the following, the *sorted*, relative character frequencies of a string are called its *character distribution.* For example, consider the text string '`passwd`' with the corresponding ASCII values of '112 97 115 115 119 100'. The absolute frequency distribution is 2 for 115 and 1 for the four others. When these absolute counts are transformed into sorted, relative frequencies (i.e., the character distribution), the resulting values are 0.33, 0.17, 0.17, 0.17, 0.17 followed by 0 occurring 251 times.

The character distribution of an argument that is perfectly normal (i.e., non-anomalous) is called the argument's *idealized character distribution ($\mathcal{ICD}$).* The idealized character distribution is a discrete distribution with:

$\mathcal{ICD} : \mathfrak{D} \mapsto \mathfrak{P}$ with $\mathfrak{D} = \{n \in \mathcal{N} | 0 \leq n \leq 255\}$, $\mathfrak{P} = \{p \in \mathfrak{R} | 0 \leq p \leq 1\}$ and $\sum_{i=0}^{255} \mathcal{ICD}(i) = 1.0$.

In contrast to signature-based approaches, the character distribution model has the advantage that it cannot be evaded by some well-known attempts to hide malicious code inside a string. In fact, signature-based systems often contain rules that raise an alarm when long sequences of 0x90 bytes (the `nop` operation in Intel x86-based architectures) are detected in a packet. An intruder may substitute these sequences with instructions that have a similar behavior (e.g., `add rA,rA,0`, which adds 0 to the value in register A and stores the result back to A). By doing this, it is possible to prevent signature-based systems from detecting the attack. Such sequences, nonetheless, cause a distortion of the string's character distribution, and, therefore, the character distribution analysis still yields a high anomaly score.

Learning. The idealized character distribution is determined during the training phase. For each observed argument string, its character distribution is stored. The idealized character distribution is then approximated by calculating the average of all stored character distributions. This is done by setting $\mathcal{ICD}(n)$ to the mean of the n^{th} entry of the stored character distributions $\forall n : 0 \leq n \leq 255$. Because all individual character distributions sum up to unity, their average will do so as well. This ensures that the idealized character distribution is well-defined.

Detection. Given an idealized character distribution $\mathcal{ICD}$, the task of the detection phase is to determine the probability that the character distribution of an argument is an actual sample drawn from its $\mathcal{ICD}$. This probability is calculated by a statistical test.

This test should yield a high confidence in the correctness of the hypothesis for normal (i.e., non-anomalous) arguments while it should reject anomalous ones. A number of statistical tests can be used to determine the agreement between the idealized character distribution and the actual sample. We use a variant of the Pearson χ^2-test as a 'goodness-of-fit' test [4]. It was chosen because it is simple and efficient to assess the 'normality' of the character distribution.

The χ^2-test requires that the function domain is divided into a small number of intervals, or bins. In addition, it is preferable that all bins contain at least 'some' elements; the literature considers five to be sufficient for most cases. As the exact division of the domain does not influence the outcome of the test significantly, we have chosen the six segments for the domain of $\mathcal{ICD}$ as follows: {[0], [1,3], [4,6], [7,11], [12,15], [16,255]}. Although the choice of these six bins is somewhat arbitrary, it reflects the fact that the relative frequencies are sorted in descending order. Therefore, the values of $\mathcal{ICD}(x)$ are higher when x is small, and thus all bins contain some elements with a high probability.

When a new system call argument is analyzed, the number of occurrences of each character in the string is determined. Afterward, the values are sorted in descending order and combined by aggregating values that belong to the same segment. The χ^2-test is then used to calculate the probability that the given sample has been drawn from the idealized character distribution. The derived probability value p is used as the return value for this model. When the probability that the sample is drawn from the idealized character distribution increases, p increases as well.

4.3 Structural Inference

Often, the manifestation of an exploit is immediately visible in system call arguments as unusually long strings or strings that contain repetitions of non-printable characters. There are situations, however, when an attacker is able to craft her attack in a manner that makes its manifestation appear more regular. For example, non-printable characters can be replaced by groups of printable characters. In such situations, we need a more detailed model of the system call argument. This model can be acquired by analyzing the argument's structure. For our purposes, the structure of an argument is the regular grammar that describes all of its normal, legitimate values.

For example, consider the first parameter of the `open` system call. It is a null-terminated character string that specifies the canonical name of the file that should be opened. Assume that during normal operation, an application only opens files that are located in the application's home directory and its sub-directories. For this application, the structure of the first argument of the `open` system call should reflect the fact that file names always start with the absolute path name to the program's home directory followed by a (possibly empty) relative path and the file name. In addition, it can be inferred that the relative path is an alternation of slashes and strings. If the directory names consist of lowercase characters only, this additional constraint can be determined as well. When an attacker exploits a vulnerability in this application and attempts to open an 'anomalous' file such as '`/etc/passwd`', an alert should be raised, as this file access does not adhere to the inferred pattern.

Learning. When structural inference is applied to a system call argument, the resulting grammar must be able to produce at least all training examples. Unfortunately, there is no unique grammar that can be derived from a set of input elements. When no negative examples are given (i.e., elements that should not be derivable from the grammar), it is always possible to create either a grammar that contains exactly the training data or a grammar that allows production of arbitrary strings. The first case is a form of over-simplification, as the resulting grammar is only able to derive the learned input without providing any level of abstraction. This means that no new information is deduced. The second case is a form of over-generalization because the grammar is capable of producing all possible strings, but there is no structural information left.

The basic approach used for our structural inference is to generalize the grammar as long as it seems to be 'reasonable' and stop before too much structural information is lost. The notion of 'reasonable generalization' is specified with the help of Markov models and Bayesian probability.

In a first step, we consider the set of training items as the output of a *probabilistic* grammar. A probabilistic grammar is a grammar that assigns probabilities to each of its productions. That means that some words are more likely to be produced than others. This fits well with the evidence gathered from system calls, as some parameter values appear more often, and is important information that should not be lost in the modeling step.

A probabilistic regular grammar can be transformed into a non-deterministic finite automaton (NFA). Each state S of the automaton has a set of n_S possible output symbols o which are emitted with a probability of $p_S(o)$. Each transition t is marked with a probability $p(t)$ that characterizes the likelihood that the transition is taken. An automaton that has probabilities associated with its symbol emissions and its transitions can also be considered a Markov model.

The output of the Markov model consists of all paths from its start state to its terminal state. A probability value can be assigned to each output word w (that is, a sequence of output symbols $o_1, o_2, \ldots, o_k$). This probability value (as shown in Equation 2) is calculated as the sum of the probabilities of all distinct paths through the automaton that produce w. The probability of a single path is the product of the probabilities of the emitted symbols $p_{S_i}(o_i)$ and the taken transitions $p(t_i)$. The probabilities of all possible output words w sum up to 1.

$$p(w) = p(o_1, o_2, \ldots, o_k) \;=\; \sum_{(paths\ p\ for\ w)} \; \prod_{(states\ \in\ p)} p_{S_i}(o_i) * p(t_i) \tag{2}$$

The target of the structural inference process is to find a NFA that has the highest likelihood for the given training elements. An excellent technique to derive a Markov model from empirical data is explained in [21]. It uses the Bayesian theorem to state this goal as

$$p(Model|TrainingData) \;=\; \frac{p(TrainingData|Model) * p(Model)}{p(TrainingData)} \tag{3}$$

The probability of the training data is considered a scaling factor in Equation 3 and it is subsequently ignored. As we are interested in maximizing the *a posteriori* probability (i.e., the left-hand side of the equation), we have to maximize the product shown in the enumerator on the right-hand side of the equation. The first term – the probability of the training data given the model – can be calculated for a certain automaton (i.e., for a certain model) by adding the probabilities calculated for each input training element as discussed above. The second term – the prior probability of the model – is not as straightforward. It has to reflect the fact that, in general, smaller models are preferred. The model probability is calculated heuristically and takes into account the total number of states as well as the number of transitions and emissions at each state. This is justified by the fact that smaller models can be described with less states as well as fewer emissions and transitions.

The model building process starts with an automaton that exactly reflects the input data and then gradually merges states. This state merging is continued until the *a posteriori* probability no longer increases. The interested reader is referred to [21] and [22] for details.

Detection. Once the Markov model has been built, it can be used by the detection phase to evaluate string arguments. When the word is a valid output from the Markov model, the model returns 1. When the value cannot be derived from the given grammar, the model returns 0.

4.4 Token Finder

The purpose of the token finder model is to determine whether the values of a certain system call argument are drawn from a limited set of possible alternatives (i.e., they are tokens or elements of an enumeration). An application often passes identical values to certain system call parameters, such as flags or handles. When an attack changes the normal flow of execution and branches into maliciously injected code, such constraints are often violated. When no enumeration can be identified, it is assumed that the values are randomly drawn from the argument type's value domain (i.e., random identifiers for every system call).

Learning. The classification of an argument as an enumeration or as a random identifier is based on the observation that the number of different occurrences of parameter values is bound by some unknown threshold t in the case of an enumeration, while it is unrestricted in the case of random identifiers.

When the number of different argument instances grows proportional to the total number of arguments, the use of random identifiers is indicated. If such an increase cannot be observed and the number of different identifiers follows a standard diminishing growth curve [14], we assume an enumeration. In this case, the complete set of identifiers is stored for the subsequent detection phase.

The decision between an enumeration and unique identifiers can be made utilizing a simple statistical test, such as the non-parametric Kolmogorov-Smirnov variant as suggested in [14]. This paper discusses a problem similar to our token finder for arguments of SQL queries and its solution can be applied to our model.

Detection. When it was determined that the values of a system call argument are tokens drawn from an enumeration, any new value is expected to appear in the set of known identifiers. When it does, 1 is returned, 0 otherwise. When it is assumed that the parameter values are random identifiers, the model always returns 1.

5 Implementation

Based on the models presented in the previous section, we have implemented an intrusion detection system (IDS) that detects anomalies in system call arguments for Linux 2.4. The program retrieves events that represent system call invocations from an operating system auditing facility called Snare [20]. It computes an anomaly score for each system call and logs the event when the derived score exceeds a certain threshold.

Our intrusion detection tool monitors a selected number of security-critical applications. These are usually programs that require `root` privileges during execution such as server applications and `setuid` programs. For each program, the IDS maintains data structures that characterize the normal profile of every monitored system call. A system call profile consists of a set of models for each system call argument and a function that calculates the anomaly score for input events from the corresponding model outputs.

Before the IDS can start the actual detection process, it has to complete a training phase. This training phase is needed to allow the used models to determine the characteristics of normal events and to establish anomaly score thresholds to distinguish between regular and malicious system calls.

The training phase is split into two steps. During the first step, our system establishes profiles for the system calls of each monitored application. The received input events are directly utilized for the learning process of the models.

During the second step, suitable thresholds are established. This is done by evaluating input events using the profiles created during the previous step. For each profile, the highest anomaly score is stored and the threshold is set to a value that is a certain, adjustable percentage higher than this maximum. The default setting for this percentage (also used for our experiments) is 10%. By modifying the percentage, the user can adjust the sensitivity of the system and perform a trade-off between the number of false positives and the expected detection accuracy. As each profile uses its own threshold, the decision can be made independently for each system call for every monitored application. This fine-grained resolution allows one to precisely tune the IDS.

Once the profiles have been established – that is, the models have learned the characteristics of normal events and suitable thresholds have been derived – the system switches to detection mode. In this mode, each system call executed by an application is evaluated with respect to the corresponding profile. If the computed anomaly value is higher than the established threshold, an alarm is raised.

The anomaly score AS is equal to the sum of the negative logarithms of the probability values p_m returned by each model m that is part of the profile, that is, $AS = \sum_m -\log(p_m)$. To prevent $-\log(p_m)$ from getting too large when p_m is close to 0, we set a lower bound of 10^{-6} for p_m. The equation is chosen such that low probability values have a pronounced effect on the final score.

All detection models used by our system are implemented as part of a general library. This library, called *libAnomaly*, provides a number of useful abstract entities for the creation of anomaly-based intrusion detection systems and makes frequently-used detection techniques available. The library has been created in response to the observation that almost all anomaly-based IDSs have been developed in an ad-hoc way. Much basic functionality is implemented from scratch for every new prototype and also the authors themselves have written several instances of the same detection technique for different projects.

6 Experimental Results

This section details the experiments undertaken to evaluate the classification effectiveness and performance characteristics of our intrusion detection system.

The goal of our tool is to provide reliable classification of system call events in a performance-critical server environment. This requires that the system performs accurate detection while keeping the number of false alerts extremely low.

6.1 Classification Effectiveness

To validate the claim that our detection technique is accurate, a number of experiments were conducted.

For the first experiment, we ran our system on the well-known 1999 MIT Lincoln Labs Intrusion Detection Evaluation Data [13]. We used data recorded during two attack free weeks (Week 1 and Week 3) to train our models and then performed detection on the test data that was recorded during two subsequent weeks (Week 4 and Week 5).

The truth file provided with the evaluation data lists all attacks carried out against the network installation during the test period. When analyzing the attacks, it turned out that many of them were reconnaissance attempts such as network scans or port sweeps, which are only visible in the network dumps and do not not leave any traces in the system calls. Therefore, we cannot detect them with our tool.

Another class of attacks are policy violations. These attacks do not allow an intruder to elevate privileges directly. Instead, they help to obtain information that is classified as secret by exploiting some system misconfiguration. This class of attacks contains intrusions that do not exploit a weakness of the system itself, but rather take advantage of a mistake that an administrator made in setting up the system's access control. Such incidents are not visible for our system either, as information is leaked by 'normal' but unintended use of the system.

The most interesting class of attacks are those that exploit a vulnerability in a remote or local service, allowing an intruder to elevate her privileges. The MIT Lincoln Labs data contains 25 instances of attacks that exploit buffer overflow vulnerabilities in four different programs. Table 1 summarizes the results found for the attacks against these four programs: `eject`, `ps`, `fdformat`, and `ffbconfig`. In addition, we present results for interesting daemon and `setuid` programs to assess the number of the false alerts. The *Total* column shows the sum of all system calls that are executed by the corresponding program and analyzed by our system. The *Attacks* column shows the number of attacks against the vulnerable programs in the data set. *Identified Attacks* states the number of attacks that have been successfully detected by our system and, in parentheses, the number of corresponding system calls that have been labeled as anomalous. It is very common that attacks result in a series of anomalous system calls. The *False Alarms* column shows the number of system calls that have been flagged anomalous although these invocations are not related to any attack.

In the second experiment, we evaluated the ability of our system to detect a number of recent attacks. Four daemon programs and one `setuid` tool were installed to simulate a typical Internet server. After the test environment was prepared, the intrusion detection system was installed and trained for about one hour. During the training period, we attempted to simulate normal usage of the system. Then, the intrusion detection system was switched to detection mode and more extensive tests were conducted for five more hours. No malicious activity took place. After that, we carried out three actual exploits against the system, one against `wuftpd`, one against `linuxconf` and one against `Apache`. All of them were reliably detected. As our system is currently not able to automatically

Table 1. 1999 MIT Lincoln Labs Evaluation Results

Application	Total Syscalls	Attacks	Identified Attacks	False Alarms
`eject`	138	3	3 (14)	0
`fdformat`	139	6	6 (14)	0
`ffbconfig`	21	2	2 (2)	0
`ps`	4,949	14	14 (55)	0
`ftpd`	3,229	0	0 (0)	14
`sendmail`	71,743	0	0 (0)	8
`telnetd`	47,416	0	0 (0)	17
`Total`	127,635	25	25 (85)	39

Table 2. Detection Accuracy

Application	Total Syscalls	Attacks	Identified Attacks	False Alarms
`wuftpd`	4,887	1	1 (86)	1
`Apache`	17,274	1	1 (2)	0
`OpenSSH`	9,562	0	0 (0)	6
`sendmail`	15,314	0	0 (0)	5
`linuxconf`	4,422	1	1 (16)	3
`Total`	51,459	3	3 (104)	15

Controlled Environment

Application	Total Syscalls	False Alarms
`dhcpd`	431	0
`imapd`	418,152	4
`qmail`	77,672	11
`Total`	496,255	15

Real-World Environment

determine when enough training data has been processed, the duration of the training period was chosen manually.

The left part of Table 2 shows, for each application, the number of analyzed system calls, the number of detected attacks with their corresponding anomalous system calls and the number of false alerts. An analysis of the reported false alerts confirms that all alarms were indications of anomalous behavior that was not encountered during the training phase. Although the anomalous situations were not caused by malicious activity, they still represent deviations from the 'normal' operation presented during the learning process. While many useful generalizations took place automatically and no alerts were raised when new files were accessed, the login of a completely new user or the unexpected termination of processes were still considered suspicious.

The `7350wu` attack exploits an input validation error of `wuftpd` [1]. It was chosen because it was used by Wagner and Soto [26] as the basis for a mimicry attack to evade detection by current techniques based on system call sequences. Our tool labeled 86 system calls present in the trace of the `7350wu` attack as anomalous, all directly related to the intrusion. 84 of these anomalies were caused by arguments of the `execve` system call that contained binary data and were not structurally similar to argument values seen in the training data.

It should be noted that none of these anomalies would be missing were the exploit disguised using the mimicry technique suggested by Wagner and Soto [26]. Since each system call is examined independently, the insertion of intervening system calls to modify their sequence does not affect the classification of the others as anomalies. This shows that our technique is not affected by attempts to imitate normal system call sequences. Note that this does not imply that our IDS is immune to all possible mimicry attacks. However, by combining our system with a sequence-based approach the potential attack space is reduced significantly. This is due to the fact that the approaches are complimentary and an attacker would have to subvert both systems.

The attack against `linuxconf` exploits a recently discovered vulnerability [2] in the program's handling of environment variables. When the exploit was run, the intrusion detection system identified 16 anomalous `open` system calls with suspicious path arguments that caused the string length, the character distribution and the structural inference model to report an anomalous occurrence. Another example is the structural inference model alerting on `open` being invoked with the argument '`segfault.eng/segfault.eng`'. This is a path which is used directly by the exploit and never occurs during normal program execution.

The attack against `apache` exploits the `KEY_ARG` vulnerability in `OpenSSL v0.9.6d` for `Apache/mod_ssl`. When the attack is launched, our system detects two anomalous system calls. One of these calls, `execve`, is reported because `Apache` does not create a `bash` process during normal operation.

The third experiment was conducted to obtain a realistic picture of the number of false alerts that can be expected when the system is deployed on a real-world server. We installed our program on the group's e-mail server, trained the models for a period of two days and then performed detection on several important daemons (`qmail`, `imapd`, `dhcpd`) for the subsequent five days. The right part of Table 2 shows the number of analyzed system calls as well as the number of false alerts raised during the five days, listed for each of the monitored applications.

6.2 System Efficiency

To quantify the overhead of our intrusion detection system, we have measured its time and space performance characteristics.

The memory space required by each model is practically independent of the size of the training input. Although temporary memory usage during the learning phase can grow proportional to the size of the training data, eventually the models abstract this information and occupy a near constant amount of space. This is reflected in Table 3 that shows the memory used by our system for two different runs after it had been trained with data from normal executions of `wuftpd` and `linuxconf`, respectively. The results confirm that memory usage is very similar for both test runs, although the size of the input files is different by a factor of 2.5.

To obtain measures that can quantify the impact of our intrusion detection system on a heavily utilized server, we set up a small dedicated network consisting of three PCs (1.4 GHz Pentium IV, 512 MB RAM, Linux 2.4) connected via a 100 Mbps Ethernet. One server machine hosted the intrusion detection system and `wuftpd`. The remaining two PCs ran multiple FTP client applications which continuously downloaded files from the server. This experiment was run three

Table 3. IDS Memory Usage

Application	Training Data Size	Memory Usage
`wuftpd`	37,152K	5,842K
`linuxconf`	14,663K	5,264K

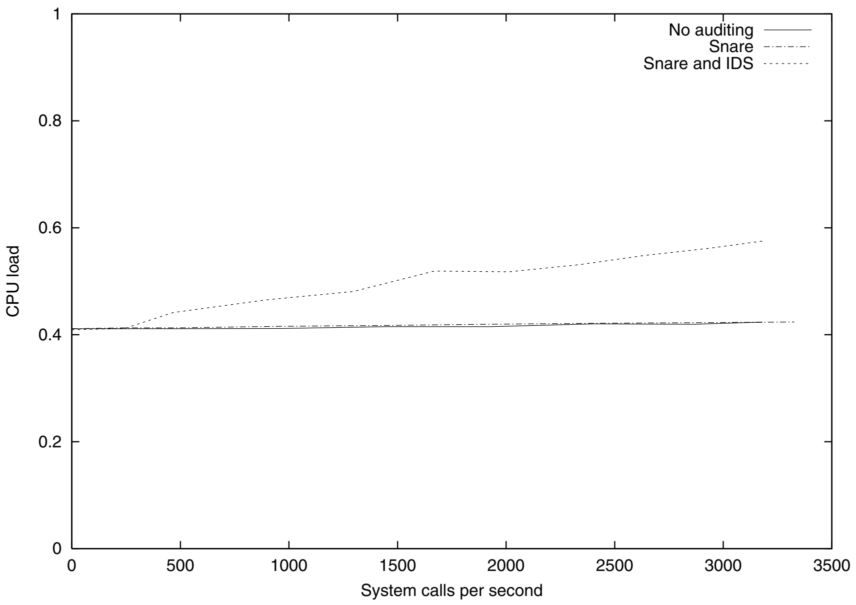


Fig. 1. CPU Load for different System Call Rates

times: once with `wuftpd` only, once with `wuftpd` and the auditing facility, and finally once with `wuftpd`, the auditing, and our intrusion detection system. In no cases were audit records dropped.

The server performance experienced by each client was virtually indistinguishable for all three cases. This indicates that the number of system calls that have to be analyzed every second by the intrusion detection system (210 on average in this case) is too low to be noticeable as performance degradation. Further analysis showed that the bottleneck in this experiment was the network.

To increase the system call rate to a point that would actually stress the system, we developed a synthetic benchmark that can execute a variable number of system calls per second at a rate that far exceeds the rate of system calls normally invoked by server applications. By measuring the resulting CPU load for different rates of system calls, we obtain a quantitative picture of the impact of our detection tool and its ability to operate under very high loads.

We ran the benchmark tool on an otherwise idle system for varying system call rates three times: once without any auditing, once with system call auditing (i.e., Snare), and finally once with system call auditing (i.e., Snare) and our intrusion detection system. Figure 1 shows the resulting CPU load observed on the system as an average of 10 runs.

The benchmark application used approximately 40% of the CPU on an idle system without auditing. As the number of system calls per second increased, a negligible impact on the CPU was observed, both with auditing turned completely off and with auditing in place. When our intrusion detection system was enabled, the CPU load increased up to 58%, when the benchmark performed about 3000 system calls per second. Note that this rise was caused by a nearly fifteen-fold increase of the number of system calls per second compared to the number that needed to be analyzed when `wuftp` was serving clients on a saturated fast Ethernet.

7 Conclusions

For a long time system calls and their arguments have been known to provide extensive and high quality audit data. Their analysis is used in security applications to perform signature-based intrusion detection or policy-based access control. However, learning-based anomaly intrusion detection has traditionally focused only on the sequence of system call invocations. The parameters have been neglected because their analysis has been considered either too difficult or too expensive computationally.

This paper presents a novel approach that overcomes this deficiency and takes into account the information contained in system call arguments. We introduce several models that learn the characteristics of legitimate parameter values and are capable of finding malicious instances. Based on the proposed detection techniques, we developed a host-based intrusion detection tool that monitors running applications to identify malicious behavior. Our experimental evaluation shows that the system is effective in its detection and efficient in its resource usage.

References

1. Advisory: Input validation problems in wuftpd. `http://www.cert.org/advisories/CA-2000-13.html`, 2000.
2. Advisory: Buffer overflow in linuxconf. `http://www.idefense.com/advisory/08.28.02.txt`, 2002.
3. M. Bernaschi, E. Gabrielli, and L. V. Mancini. REMUS: a Security-Enhanced Operating System. *ACM Transactions on Information and System Security*, 5(36), February 2002.
4. Patrick Billingsley. *Probability and Measure.* Wiley-Interscience, 3 edition, April 1995.
5. Suresh N. Chari and Pau-Chen Cheng. Bluebox: A policy-driven, host-based intrusion detection system. In *Proceedings of the 2002 ISOC Symposium on Network and Distributed System Security (NDSS'02)*, San Diego, CA, 2002.
6. D.E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

7. S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.
8. S. Forrest. A Sense of Self for UNIX Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.
9. A.K. Ghosh, J. Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions Against Programs. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'98)*, pages 259–267, Scottsdale, AZ, December 1998.
10. Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications. In *Proceedings of the 6th Usenix Security Symposium*, San Jose, CA, USA, 1996.
11. H. S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991.
12. C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, May 1997.
13. MIT Lincoln Laboratory. DARPA Intrusion Detection Evaluation. http://www.ll.mit.edu/IST/ideval/, 1999.
14. Sin Yeung Lee, Wai Lup Low, and Pei Yuen Wong. Learning Fingerprints for a Database Intrusion Detection System. In *7th European Symposium on Research in Computer Security (ESORICS)*, 2002.
15. W. Lee, S. Stolfo, and P. Chan. Learning Patterns from Unix Process Execution Traces for Intrusion Detection. In *Proceedings of the AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.
16. W. Lee, S. Stolfo, and K. Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. In *Proceedings of the* 5^{th} *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '99)*, San Diego, CA, August 1999.
17. U. Lindqvist and P.A. Porras. Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In *IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, May 1999.
18. V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.
19. Niels Provos. Improving host security with system call policies. In *Proceedings of the 12th Usenix Security Symposium*, Washington, DC, 2003.
20. SNARE - System iNtrusion Analysis and Reporting Environment. `http://www.intersectalliance.com/projects/Snare`.
21. Andreas Stolcke and Stephen Omohundro. HiddenMarkov Model Induction by Bayesian Model Merging. *Advances in Neural Information Processing Systems*, 1993.
22. Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. In *International Conference on Grammatical Inference*, 1994.
23. K. Tan and R. Maxion. "Why 6?" Defining the Operational Limits of Stide, an Anomaly-Based Intrusion Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 188–202, Oakland, CA, May 2002.

24. K.M.C. Tan, K.S. Killourhy, and R.A. Maxion. Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits. In *Proceedings of the* 5th *International Symposium on Recent Advances in Intrusion Detection*, pages 54–73, Zurich, Switzerland, October 2002.
25. D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001. IEEE Press.
26. D. Wagner and P. Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proceedings of the* 9th *ACM Conference on Computer and Communications Security*, pages 255–264, Washington DC, USA, November 2002.
27. C. Warrender, S. Forrest, and B.A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

Author Index