CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Subterfuge-safe trust management for delegation of permissions in open environments
Authors	Abdi, Samane
Publication date	2015
Original Citation	Abdi, S. 2015. Subterfuge-safe trust management for delegation of permissions in open environments. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2015, Samane Abdi. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2024-03-29 01:06:29
Item downloaded from	https://hdl.handle.net/10468/2097



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Subterfuge-Safe Trust Management for Delegation of Permissions in Open Environments

Seyedehsamane Abdigarmestani
(Samane Abdi)



NATIONAL UNIVERSITY OF IRELAND, CORK

Faculty of Science

Department of Computer Science

Thesis submitted for the degree of Doctor of Philosophy

January 2015

Head of Department: Professor Barry O'Sullivan

Supervisor: Dr. John Herbert

Contents

Abstract v
Acknowledgements vii
List of Figures viii
List of Tables x

1 Introduction **1**
1.1 Motivation 2
1.2 Contributions 4
1.3 Research Overview 6
1.4 Structure of Thesis 8

2 Background and Related Work **9**
2.1 Traditional Access Control Models 9
2.2 Distributed Access Control Models 12
2.3 Subterfuge Vulnerability 21
2.4 Delegation Subterfuge in Existing Trust Management Approaches 21
2.4.1 Delegation Subterfuge in KeyNote 22
2.4.2 Delegation Subterfuge in RT 24
2.4.3 Delegation Subterfuge In secPAL 25
2.4.4 Delegation Subterfuge In SPKI/SDSI 27
2.5 Summary 32

3 Subterfuge Safe Trust Management **33**
3.1 Overview on SSTM Infrastructure 33
3.2 Subterfuge Safe Authorization Language 34
3.2.1 Principals 34
3.2.2 Permissions 38
3.2.3 Delegation 43
3.2.4 Accountability 45
3.3 Formal Foundation for SSAL 46
3.3.1 Principal Names Relation 47
3.3.2 Permission Delegation 48
3.3.3 Permission Holding 49
3.3.4 Permission Ordering 50
3.3.5 Accountability for Permissions 51
3.3.6 Logical Properties 52
3.4 Threats and Mitigation 55
3.4.1 Key Expiration Threat 55
3.4.2 Key Refreshing Threat 55
3.4.3 Key Change Threat 55
3.4.4 Key Compromised Threat 56
3.4.5 Threat Mitigation Techniques 56
3.5 Certificate Chain Discovery 59
3.5.1 Related Work 59
3.5.2 Certificate Chain Discovery for SSTM 61

3.6 Discussion . . . 70
3.7 Summary . . . 71

4 Ontology-Based Implementation for SSTM 72
4.1 Preliminaries . . . 73
4.1.1 Definition of Ontology . . . 73
4.1.2 Ontology Languages and Reasoning Tools . . . 74
4.1.3 Methodology . . . 90
4.2 SSAL^O . . . 91
4.2.1 Design the TBox for SSAL^O . . . 92
4.2.2 Instantiating ABox with Individuals . . . 102
4.2.3 Policy Rules . . . 105
4.2.4 Integration of Policies within SSAL^O . . . 107
4.2.5 Queries for Trust Management . . . 109
4.2.6 Case study: Trust Management for a Selling Service by Brokers . . . 111
4.2.7 Run-Time Performance . . . 114
4.2.8 Related Work for Solving Heterogeneity Problem . . . 115
4.2.9 Discussion . . . 116
4.3 Summary . . . 118

5 Extending SSTM for Supporting Secure Cross Coalition Cooperation 119
5.1 Coalition Definition . . . 120
5.2 Coalition Features . . . 122
5.2.1 Membership Management . . . 122
5.2.2 Administration . . . 122
5.2.3 Subterfuge Safe Open Cooperation . . . 123
5.2.4 Formation and Evolution . . . 124
5.3 Existing Coalition Frameworks . . . 125
5.3.1 Systems Research Centre Model . . . 125
5.3.2 Coalition-Based Access Control Model . . . 126
5.3.3 Virtual Private Network Model . . . 127
5.3.4 Mäki-Aura Model . . . 127
5.3.5 Internet Services of Coalition Model . . . 128
5.3.6 Ellison-Dohrmann Model . . . 128
5.3.7 Distributed Authorization Language Model . . . 129
5.4 Secure Coalition Characteristics . . . 130
5.5 SSTM-Based Coalition Framework . . . 131
5.5.1 Forming a New Coalition . . . 133
5.5.2 Issuing Membership . . . 134
5.5.3 Local Policy for Cross Coalition Sharing Resources . . . 134
5.5.4 Coalition Split/Merge . . . 138
5.6 Discussion . . . 140
5.7 Case Study . . . 141
5.8 Summary . . . 144

6 Application of SSTM **145**
6.1 Secure Cloud Federation . . . 145
6.1.1 Breakdown in Permission Accountability in Cloud Federation . . . 147
6.1.2 Accountability for Delegated Permission . . . 149
6.1.3 Compliance Checking for Accountability . . . 150
6.1.3.1 Authorization Check . . . 152
6.1.3.2 Delegation Check . . . 152
6.1.4 Managing Cloud Federation Using SSTM . . . 152
6.1.5 Discussion . . . 155
6.2 Trust Management for Secure Federation of XMPP Servers . . . 156
6.2.1 Introduction to XMPP Servers . . . 156
6.2.2 SSTM for XMPP Servers Federation . . . 156
6.2.3 Case Study . . . 157
6.2.4 Checking for Subterfuge Safe Delegation . . . 164
6.2.4.1 Discussion . . . 165
6.3 Summary . . . 166

7 Conclusions and Future Work **167**
7.1 Overview . . . 167
7.2 Summary of Contributions . . . 168
7.3 Future work . . . 170
7.3.1 Threshold Structures . . . 170
7.3.2 Run-Time Optimization . . . 171
7.4 Summary . . . 172

Appendices **188**

A List of Abbreviations and Symbols **189**

B Proof of properties of SSAL Logic **191**
B.1 Proof for Property 1 . . . 191
B.2 Proof for Property 2 . . . 192
B.3 Proof for Property 3 . . . 193
B.4 Proof for Property 4 . . . 193
B.5 Proof for Property 5 . . . 194
B.6 Proof for Property 6 . . . 195
B.7 Proof for Property 7 . . . 196
B.8 Proof for Property 8 . . . 197
B.9 Proof for Property 9 . . . 198

I, Seyedehsamane Abdigarmestani (Samane Abdi), certify that this thesis is my own work and I have not obtained a degree in this university or elsewhere on the basis of the work submitted in this thesis.

Seyedehsamane Abdigarmestani
(Samane Abdi)

Abstract

Open environments involve distributed entities interacting with each other in an open manner. Many distributed entities are unknown to each other but need to collaborate and share resources in a secure fashion. Usually resource owners alone decide who is trusted to access their resources. Since resource owners in open environments do not have a complete picture of all trusted entities, trust management frameworks are used to ensure that only authorized entities will access requested resources. Every trust management system has limitations, and the limitations can be exploited by malicious entities. One vulnerability is due to the lack of globally unique interpretation for permission specifications. This limitation means that a malicious entity which receives a permission in one domain may misuse the permission in another domain via some deceptive but apparently authorized route; this malicious behaviour is called subterfuge.

This thesis develops a secure approach, Subterfuge Safe Trust Management (SSTM), that prevents subterfuge by malicious entities. SSTM employs the Subterfuge Safe Authorization Language (SSAL) which uses the idea of a local permission with a globally unique interpretation (localPermission) to resolve the misinterpretation of permissions. We model and implement SSAL with an ontology-based approach, SSAL^O, which provides a generic representation for knowledge related to the SSAL-based security policy. SSAL^O enables integration of heterogeneous security policies which is useful for secure cooperation among principals in open environments where each principal may have a different security policy with different implementation. The other advantage of an ontology-based approach is the Open World Assumption, whereby reasoning over an existing security policy is easily extended to include further security policies that might be discovered in an open distributed environment. We add two extra SSAL rules to support dynamic coalition formation and secure cooperation among coalitions. Secure federation of cloud computing platforms and secure federation of XMPP servers are presented as case studies of SSTM. The results show that SSTM provides robust accountability for the use of permissions in federation. It is also shown that SSAL is a suitable policy language to express the subterfuge-safe policy statements due to its well-defined semantics, ease of use, and integrability.

To Ehsan, Kian & My Parents

Acknowledgements

"The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man." (George Bernard Shaw)

During the course of my PhD project, I have been luckily supported by many people to whom I would like to express my gratitude.

First of all, I would like to express my sincere gratitude to my supervisor, Dr. John Herbert, for his support, technical discussions, and for giving freedom during the course of this work. I was very lucky to have John's support in helping me to regain my confidence to do a technical work at this level. I would also thank my thesis examiners, Prof. Joe Carthy and Dr. Sabin Tabirca, for taking time to read my thesis and for valuable discussions during the viva exam. I am also grateful to Dr. Simon Foley for initiating this project and providing financial support.

Thanks to Prof. Barry O'Sullivan my PhD advisor, Dr.Ian Pitt from the Department of Computer Science, Ms Michelle Nelson from the UCC Graduate Studies, and Ms Suzan Buckly from International Office for their support and assistance. I would also thank Science Foundation Ireland (SFI), FAME project for financial support of this work. My sincere appreciation to Prof. Barry O'Sullivan, the director of Insight Centre for Data Analytics in UCC for providing funding to attend and present talks at the conferences.

I also thank former and current members of Security Group: William, Wayne, Fatih, Olgierd, and Ultan. I am also grateful to former members of the Cork Constraint Computation Centre, current and former members of Insight Centre for Data Analytics, particularly Ena Tobin, Thuy Truong, Walid Trabelsi, Abdul Razak and John Horan. All my friends whom I shared a large part of my life in Cork with them. Big thank you to Caitriona Walsh, Eleanor O'Riordan, and Ann O'Brien in the Computer Science Department who provided me with necessary administrative supports during the last 4 years.

I take this opportunity to thank my parents not only for visiting me in Cork during my maternity period, but also for all their support in my life. Special thanks to my siblings in particular my lovely sister, Sanaz, who spent three months in Cork to provide the necessary support and help.

Finally, I would like to thank my husband and love of my life, Ehsan, whom I owe my sincerest gratitude. This PhD work would have not been easy to overcome without his continuous and endless support and love. The last word goes to Kian, my lovely son, who has been the light of my life for the last two years and who has given me the extra strength and motivation to get things done.

List of Figures

1.1 Centralized access control model 3

2.1 DAC: UNIX access control matrix 11
2.2 A role-based access control example 12
2.3 An overview on trust management system 15
2.4 Expected chain for downloading *AlbumX* 28
2.5 Unexpected chain for downloading *AlbumX* 29
2.6 Subterfuge scenario by establishing a bogus company 30
2.7 Subterfuge in the delegation of permission Atlantic.com/AlbumX 31

3.1 Overview of SSTM framework 34
3.2 Illustration of a local name, a group, and an extended local name. 35
3.3 Example of a public key 36
3.4 Direct and indirect delegation 44
3.5 "local cert" and "copy cert" in local repositories. 62
3.6 Initialization of certificates in distributed repositories 67

4.1 Concept hierarchy 79
4.2 Example of concept membership 81
4.3 Set of individuals 82
4.4 OWA without covering axiom 84
4.5 OWA considering covering axiom 84
4.6 An overview on SSAL^O 93
4.7 An overview on concept Principal in SSAL^O 95
4.8 An overview on concept LocalPermission in SSAL^O 99
4.9 An overview on concept Delegation in SSAL^O 101
4.10 Integrating the locally defined policy to SSAL^O 109
4.11 Implementation of SSTM 111
4.12 Delegation certificate 112
4.13 Name certificate 112
4.14 Permission certificate 112
4.15 Run time performance of reasoning over SSAL^O 114
4.16 Run time performance of reasoning over SSAL^O 114

5.1 A sample of coalition structure 120
5.2 Centralized access control for coalitions 121
5.3 Decentralized access control for coalitions 122
5.4 Subterfuge in open cooperation of coalitions 124
5.5 The SRC framework 126
5.6 Summary of coalition features of different frameworks 130
5.7 A coalition is formed by leader, issuing certificates to members . 137

6.1 Breakdown in accountability for permissions in cloud federation 146
6.2 Locally defined permission with global unique interpretation . . 149
6.3 Trust management for cloud federation 150

6.4 Inputs and output of a compliance checking system 151
6.5 Robust accountability for permissions in cloud federation 154
6.6 XMPP server, client, and service connections 157
6.7 Representation of the delegation statement 6.7 in SSAL^{O} 160
6.8 FACNAC/SSTM federation scenario 164

List of Tables

3.1 Permission *sell* and its definition in two different name spaces . 40

4.1 Concrete syntax of DL constructors 77
4.2 A comparison of *SHOIN(D)* and OWL-DL constructors 77
4.3 Domain and range of properties in SSAL^O 102
4.4 Terminological axioms and their syntaxes 103

Chapter 1

Introduction

Controlling access to protected data, resources, and services has been a major issue in information security since the beginning of the information security discipline [1]. Access control systems mediate access to a protected resource by only allowing authorized users to perform an operation on a particular resource. Traditional access control mechanisms are robust enough to address authorization control in closed environments. These systems are not sufficient to address authorization control in open environments. Trust management systems were introduced to address various concerns in decentralized access control. Although trust management systems have a major advantage over traditional access control approaches, they are vulnerable to subterfuge. *Subterfuge* is a deceptive behaviour with the goal of evading the actual intention of a security mechanism. This thesis explores the notion of subterfuge in current trust management frameworks, and then introduces a subterfuge safe trust and authorization model for addressing access control in open environments. This new approach for trust management supports subterfuge safe cooperation by unknown entities in open environments. In section 1.1 we discuss the motivation for this research. Section 1.2 covers the contributions of the thesis to the field of trust management. Section 1.3 gives an overview on the research. The structure of the thesis is outlined in section 1.4. Section 2 discusses background and related work. The subterfuge vulnerability is introduced in section 2.3. In section 2.4 we discuss delegation subterfuge in existing trust managements. Finally, a summary for the chapter is given in section 2.5.

1.1 Motivation

Traditional access control mechanisms focus on the protection of data in closed environments. A security administrator is familiar with all resources in the system and when a request for a resource is received, first determines who the requester is. It typically uses an authentication protocol in which the requester digitally signs the request. Then the administrator queries an internal database to decide whether the signer should be granted access to the resource and allowed to perform the requested action. Access control mechanisms for closed environments can be categorised as mandatory access control (MAC) [2, 3], discretionary access control (DAC) [4], and role-based access control (RBAC) [5–7]. All these access control models include *subject, object, action,* and *function*. Subjects are entities that can perform actions on the system; objects are the entities representing resources to which access may need to be controlled. An access control *function* is a matrix that maps each combination of *subject, object,* and *action* to an authorization decision. The authorization decision result is either the access request is granted, or the access request is denied. Figure 1.1 depicts a general model of access control for closed environments. In closed environments, granting permission for a resource is controlled by a security administrator. A security administrator is familiar with all resources that it controls and has a complete overview on subjects, objects, and actions in the system. Thus, a security administrator chooses a specific name schema for defining permissions for accessing resources. Therefore, permissions that a security administrator defines to grant access to a resource have a unique meaning in the system and can be only used for that particular resource. In this way, the possibility of deceiving a security administrator to define a permission specification that can be used to access two different resources is very low.

When environments become open and decentralized, the distributed users may make requests to access other users' resources. Each resource owner is familiar with its own resources and authorization is controlled by resource owners. The resource owners and requesters may not be known to each other in advance. Access control solutions need to decide which requester can be granted access to the protected resource, as well as which principal is qualified to provide this resource [8]. Certificate-based access control models have been proposed to address access control in open and decentralized environments. When a user makes a request to access a resource, it must provide a certificate to the resource owner proving that the requester has permission to access that resource. The resource

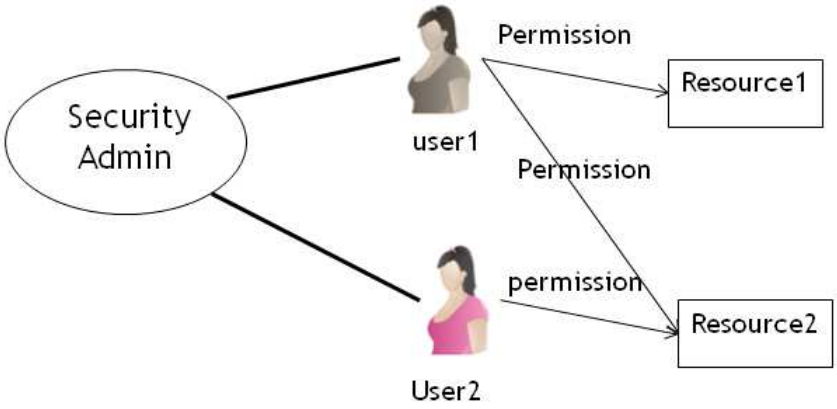


Figure 1.1: Centralized access control model

owner defines a permission specifying that the holder of that permission is allowed to access a particular resource. Therefore, defining a globally unique permission relies on the expertise of the resource owner who defines the permission specifications. In open environments, a resource owner does not have a complete overview on the name schema that other resource owners use for specifying permissions for their resources. One resource owner may define a permission specification that is the same as the permission specification that another resource owner specified for accessing its resources. This ambiguity means that a malicious requester may bypass the actual intention of a permission specification and deceive a resource owner into allowing access to its resources with an apparently legitimate permission. Preventing deception of a principal depends on the expertise of the resource owner who defines the permission specification, and the sources of vulnerability in the actual security mechanism that the resource owner uses. Many existing trust management systems are designed in an ad-hoc manner to prevent malicious behaviour. Their design follows best practice, and withstand only certain classes of known malicious behaviours. They lack a systematic way of specifying a globally unique interpretation for a permission. Without a globally unique interpretation for permissions, a principal that receives a permission in one domain, may misuse that permission in another domain via some deceptive, yet apparently authorized route, i.e. the behaviour called subterfuge. Many existing trust management frameworks such as [9, 10] are designed to specify arbitrary permissions. They assume unique and unambiguous permission specifications are provided by using a global name service. Although, global name services provide a unique interpretation for each specification, the principals participating in a federation may still define arbitrary specifications to represent their own resources. Specifying non-

ambiguous permissions depends on the expertise of the administrator who defines the permissions. However, the design of non-ambiguous permissions should not rely on ad-hoc methods; it should be formalized in a systematic way. Due to the lack of systematic design approaches, designing a well-founded security mechanism is a challenging task. This thesis explores the possibility of designing a well-founded security mechanism by answering the following research question:

Can we design a well-founded systematic method to avoid subterfuge for cooperation of distributed principals in open environments?

1.2 Contributions

This thesis contributes to the field of trust management by focusing on preventing subterfuge when managing trust and delegation relationships among distributed entities in open environments. The main contributions are as follows:

1. The notion of *localPermission* is introduced with the purpose of defining permissions locally, while also providing an automatic globally unique interpretation for that permission.

2. A logic-based authorization language, Subterfuge Safe Authorisation Language (SSAL), is introduced to support secure delegation of permissions in open cooperation. SSAL is a simple yet expressive language. SSAL can be used to prevent subterfuge without relying on a central authority and a pre-agreed global naming service.

3. An ontology for SSAL is developed. The ontology-based approach, SSAL^O, is also used as a technique for integration of heterogeneous security policies. In open environments, different organizations may define different security policies to meet their security requirements. These security policies may be implemented with different techniques and different policy languages. SSAL^O provides a mechanism for integration of these security policies. SSAL^O can be viewed as a query engine for Subterfuge Safe Trust Management (SSTM).

4. Using SSTM, a formal framework for establishing secure dynamic coalition and cross coalition cooperation is introduced. With this framework, a coalition can be dynamically established in a fully distributed manner without relying on a central authority. This framework can be used to merge

coalitions and split a coalition into further coalitions.

5. The applicability of SSTM in two real world examples (cloud federation, and federation of XMPP servers) is demonstrated. The results show that SSTM is a robust mechanism for subterfuge safe delegation of permissions in federation and also provides strong accountability for principals. The success of SSTM for these case studies indicates that it provides a general solution for subterfuge safe management of trust and authorization relationships in open collaboration of entities. It can be applied to other examples of open collaboration in open distributed environments.

The results in this thesis have been published in peer-reviewed publications as follows:

1 Simon N. Foley and **Samane Abdi**, "Avoiding delegation subterfuge using linked local permission names.," In Proceedings of the *8th international conference on Formal Aspects of Security and Trust* (FAST'11),Lueven, Belgium, September 2011.

2 Simon N. Foley, and **Samane Abdi**, "Avoiding Delegation Subterfuge Using Linked Local Permission Names," Formal Aspects of Security and Trust. (pp. 100-114). Springer Berlin Heidelberg, 2012.

3 **Samane Abdi**, "An Autonomic Trust Management Framework For Secure Dynamic Coalition Cooperation," In Proceedings of the *10th IEEE Conference on Autonomic and Trusted Computing (ATC 2013)*, Vietri sul Mare, Italy, December 2013.

4 **Samane Abdi**, "Integration of Heterogeneous Policies for Trust Management" In Proceedings of the *38th IEEE International Conference in Computer Software and Applications (COMPSACW)*, Västerås, Sweden, July 2014.

5 **Samane Abdi**, "I was confused: Robust accountability for permission delegation in cloud federations," In Proceedings of the *38th IEEE International Conference in Computer Software and Applications (COMPSACW)*, Västerås, Sweden, July 2014.

6 **Samane Abdi** and John Herbert, "An Algorithm for Distributed Certificate Chain Discovery in Open Environments," In Proceedings of the *30th ACM/SIGAPP Symposium On Applied Computing(SAC)*, Salamanca, Spain, April 2015.

7 **Samane Abdi**,"Federated Autonomic Trust Management," In Proceedings of the *1st Insight Student Conference*, Dublin, Ireland, September 2014.

The thesis is part of the Federated, Autonomic Management of End-to-end communication services (FAME) project ($http://www.fame.ie/$).

1.3 Research Overview

In this thesis, we introduce a logic-based authorization language to support open and subterfuge free delegation of permissions for secure federations. This can be used as a policy language to construct statements and manage authorization/delegation relationships, and to automate the decision making process for securely sharing resources among federated participants. This language also uses the notion of *localPermission* to eliminate ambiguity concerning the interpretation of a permission and thereby avoid subterfuge attacks. The notion of localPermission refers to permission specifications that are defined locally but have globally unique interpretation. The use of localPermissions means that a principal receiving two identical permission specifications cannot misuse the permissions for non-intended purposes, since the permissions have globally unique interpretations and clearly refer to a global context. Thus, localPermissions can prevent subterfuge when unknown entities cooperate in open environments. This new trust model can be used as a basis for the establishment and management of secure dynamic cooperation in open environments.

Subterfuge Safe Authorization Language (SSAL) is designed to support subterfuge safe delegation in open environments. SSAL provides the necessary expressiveness of an authorization language while supporting subterfuge safe delegation by using the notion of *localPermission* without requiring a pre-agreed global and unique name scheme for defining permissions. The logic of the language is robust enough in terms of providing a reliable formal way to determine whether delegation of a particular permission to principals is able to resist against a malicious principal's misbehaviour.

It is essential to establish a common vocabulary that specifies the structure and semantics of SSAL statements. Modelling SSAL within an ontology provides a standard and formal semantic for the SSAL language. An ontology provides a conceptual model of a domain of interest which is a formal description of concepts and their relationships in the domain of discourse, and is intended for sharing

and reusing knowledge. It provides the ability to make logical statements of shared information and infer new knowledge in the domain of discourse. A SSAL ontology (SSAL^O) supports reasoning and querying about the knowledge in SSAL statements without requiring an application side reasoning service. SSAL^O can be used to integrate heterogeneous policies (issued by different principals in open environments) and so supports well-founded access decisions that comply with the policies of all of the principals involved.

SSAL^O is modelled in the OWL-DL sublanguage whose expressive power relies on the expressiveness of Description Logic (DL) and the Web Ontology Language (OWL). Using OWL-DL supports maximum expressiveness while guaranteeing decidability and tractability. There is a rationale for using the ontology for encoding permissions. Permissions are structured data and ontologies are useful to model structured data. A permission specification's hierarchical structure can be expressed in more detail by capturing relations and constraints in the ontology. SSAL^O can be viewed as a Subterfuge Safe Trust Management (SSTM) engine. Any particular application or service can use SSTM to manage access control for its resources. A principal makes a request to an application or service to access some resources. A query interpreter then interprets the request and queries SSAL^O. SSAL^O returns the query results to the resource owner. Moreover, each principal may have its own local policy for access control for its own resources. The principal adds its local policy to SSAL^O, presenting a set of certificates. SSAL^O then decides whether to permit the request based on the SSAL statements, rules, and the local policy.

Every delegation certificate delegates some permission from its issuer to its subject. Chains of delegation certificates issued by different issuers may be formed, thus enabling permission to be granted in a decentralized manner. Certificate chain discovery refers to presenting a set of certificates relevant to a request. Delegation certificates can form chains of delegation, where permissions are delegated from one principal to another. In order to prove to the resource owner that the requester is authorized, the requester must present a set of certificates relevant to its request to the resource owner.

SSTM can be used as a basis to form secure coalitions. The design of secure coalitions allows for secure cooperation and sharing of resources within one coalition and across different coalitions.

1.4 Structure of Thesis

This thesis is divided into six chapters. The remainder of this chapter gives the reader a background on trust management and the problem of subterfuge in conventional trust management systems. Chapter 3 introduces a policy language that prevents the subterfuge problem in a systematic way, the Subterfuge Safe Authorization Language, SSAL. In chapter 4, an ontology-based approach is introduced to implement SSAL as a policy engine, and to address the integration of heterogeneous policies defined by different entities in open environments. A secure dynamic coalition framework that uses the subterfuge safe trust management model, SSTM, to support establishment of secure coalitions and safe cross-coalition delegation is described in chapter 5. The application of this approach is demonstrated through two real world examples in chapter 6. Chapter 7 concludes the thesis and shows directions for possible future work.

Chapter 2

Background and Related Work

An important requirement for information systems is managing how the protected data and resources must be secured against unauthorized principals, while making them available to authorized principals. This feature is called *"access control"* [11–13]. In access control, the principal that receives a request for access to its resources first checks who the requester is, by using an authentication mechanism, and then queries a centralized database to decide whether the access should be granted or denied. On the other hand, trust management, introduced in [14], is an approach for managing trust and authorization among distributed principals. The concept of trust management is closely related to that of distributed access control for open environments. The remainder of this chapter first reviews existing access control models, with a focus on trust management systems. Then, we investigate a vulnerability in current trust management frameworks that results in the violation of the intention of a security mechanism. The review starts from early work on access control models, and then continues with existing trust management frameworks.

2.1 Traditional Access Control Models

Early work on access control can be categorized as mandatory access control, discretionary access control, and role based access control. In the following sections we discuss these access control models.

Mandatory Access Control (MAC)

Access control models came from the study of security policies in the 70s [15, 16]. In hierarchical organizations the primary concern is confidentiality of data, where prevention of information leakage is the most important goal. In response to this need, Bell and LaPadula [15] introduced a security model that restricts flows of classified information. Their work led to the development of numerous multilevel security systems, and is arguably one of the most influential models in the history of computer security. Multilevel security was developed as a means to manage classified information in hierarchical organizations. Each document is labelled with a degree of sensitivity, known as a classification such as: *unclassified, confidential, secret,* and *top secret*. All the personnel of an organization are assigned a clearance level on the same labelled scale as the classification. This assignment may depend on a variety of factors, including professional rank, organizational unit. The access control policy states that a reader must have a clearance at least as high as the classification of the document he/she attempts to read. In MAC models [12, 17, 18], the security policy is determined centrally by a system administrator instead of by resource owners. In other words, the most important feature of mandatory access control is that users who create resources do not control these resources. The system security policy, set by the administrator, entirely determines the access rights.

Discretionary Access Control (DAC)

The main idea behind DAC is that the owner of a resource should be trusted to manage its security. More specifically, owners are granted full access rights to the resources under their control, and are allowed to decide whether access rights to their resources should be passed to other subjects or groups of subjects at their own discretion. Lampson formulates the first abstract model of DAC from the point of view of operating systems [19]. In his model, an access control matrix is a two-dimensional matrix with a row for each subject and a column for each object (resource). An element in the matrix specifies the access rights that a subject has for an object. An access matrix is a convenient abstraction for expressing discretionary access control polices. As a practical example, the UNIX file system implements discretionary access control. It defines three subjects in the access control matrix: the object owner, group, or everyone in the system. The user who creates an object is the owner and only the administrator can change the

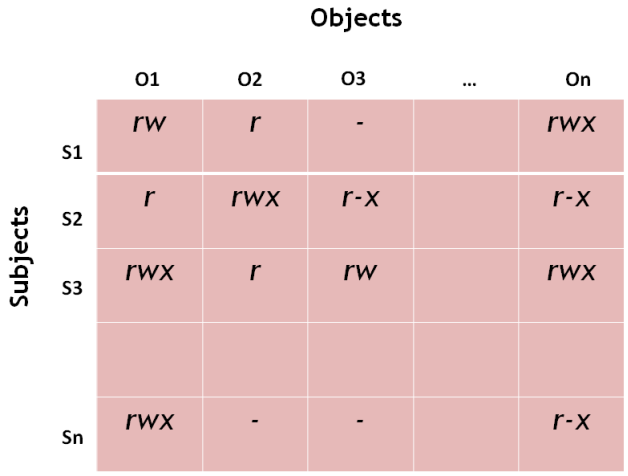


Figure 2.1: DAC: UNIX access control matrix

ownership of an object. There are three access modes: *read, write* and *execute*, and the access rights for each subject is represented as a 3-bit value,"rwx". Figure 2.1 is an illustration of UNIX access matrix. The key to UNIX access control is that the owner decides who is allowed to access the object and what kind of permission they should have. The controls are discretionary in the sense that a subject who holds an access permission may also delegate the permission to others. In DAC, although the resource owners determine their security policies but the super security administrator can change the whole security policy and control the system.

Role Based Access Control (RBAC)

In the beginning of the 90s, it was observed that resources are generally not owned by users, but rather by the organization or agency to which these users belong. Access requests are typically made by a user in the capacity of some role, and thus, access control decisions are often determined by the acting roles [6,7]. Over the years, many researchers have proposed models for RBAC [6,7,20–26]. While the differences in these models are quite significant, the core concept remains fairly consistent between them. In RBAC, the basic components are *users, permissions*, and *roles*. A user in RBAC typically refers to a human being, although this definition can be extended to include machines, computer processes or autonomous agents. Permissions are defined as an approval to execute an operation on one or more protected objects. An operation could be a simple access mode such as *read, write, update*, or a complex operation such as a method

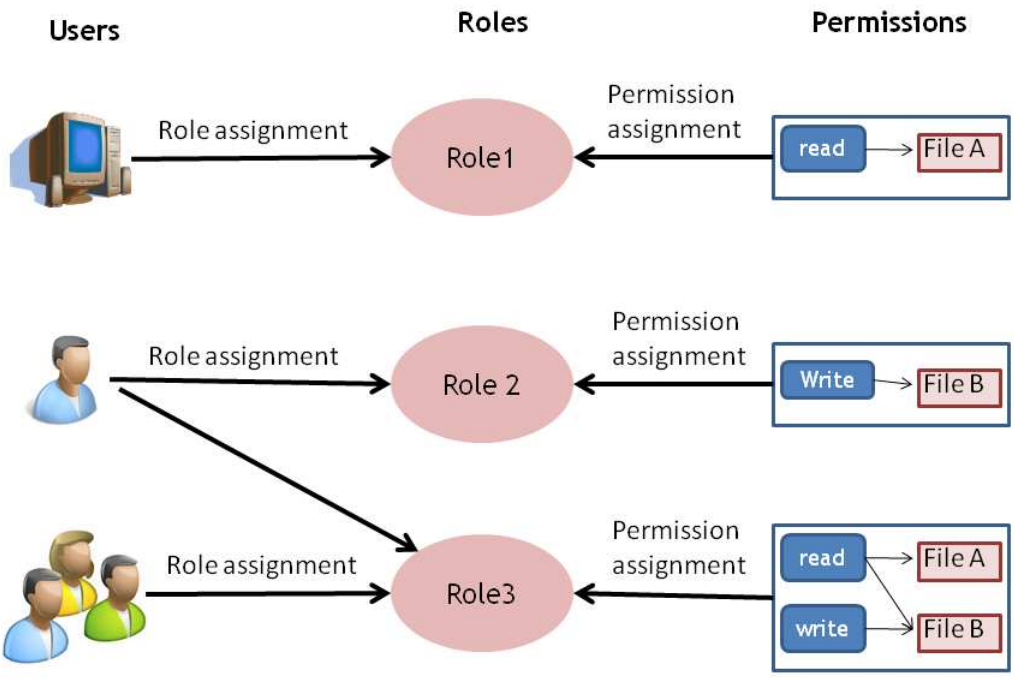


Figure 2.2: A role-based access control example

invocation in an object-oriented system [22, 23]. Central to RBAC is the notion of relation that connects permissions, users, and roles. Permissions assigned to a role represent organizational security policies and are granted to users only through roles. Suppose a user in a bank attempts to withdraw money from an account, she must be assigned to some role that permits money withdrawal, for instance, cashier. Note that, in RBAC it is possible to assign multiple roles to a single user. In RBAC, the security policy is determind centrally by a system administrator. Figure 2.2 demonstrates an RBAC model.

2.2 Distributed Access Control Models

The traditional access control models are insufficient in meeting the requirements for access control in open environments. Traditional approaches assume that a security administrator is familiar with all the system and manages the access control to all the protected resources. However, with the emergence of open networks, a central security administrator is not sufficient to manage available resources. New challenges exist for access control on resources owned by distributed principals. The open networks are generally heterogeneous, decentralized and large scale, with possibly millions of entities which may be individuals, agents, organizations

or other administrative domains. These entities wish to share their resources in a secure and controlled fashion. Collaborating entities may be mutually unknown to each other, thus access control cannot be based on a central administrator, as is the case in traditional approaches. Most modern distributed access control systems apply the ideas of cryptographic credentials as a proof of access rights in open and distributed environments [27–34]. In these approaches, any entity that can be authenticated is called a principal. The use of public keys for authentication and signing the credentials is widespread due mainly to the complexity of key management with symmetric key cryptography [35]. A principal and a service must share a secret key which is distributed over the internet. Therefore, it is desirable to constrain the use of a secret key to each individual service to limit the damage caused by disclosure of the key. Public key cryptography significantly simplifies key management because it is sufficient for a communicating party to know only public keys. A public key credential binds a public key to some attributes of the holder of the corresponding private key. From now on, in this thesis, we use certificate to refer to the cryptographic credentials. Based on how certificates are used, distributed access control may be grouped into two categories: the identity-based approach, and key-based approach.

Identity-based Approach

One common use of access control certificates is to bind the name of a subject with access permissions. The idea is that once the name of a requester has been verified by a reliable authentication mechanism, access control certificates with the matched name can then be used to make access decisions. This approach separates access control into two distinct stages: authentication and authorization. Authentication requires the binding of a public key to a name, while authorization is handled by the access control certificates which bind a name and a set of permissions. The security of this approach therefore depends on the reliability of both bindings. Standards exist for the binding of a public key to a name. Pretty Good Privacy (PGP) [36,37] and X.509 Public Key Infrastructure (PKI) [38–41] are the two most widely used standards today. The most prominent standard for binding public keys to names are the X.509, Privilege Management Infrastructure (PMI) with its support for attribute certificates [42], and Simple Distributed Security Infrastructure (SDSI) [43].

Key-based Approach

Another possible use of access control certificates is to directly bind a public key with permissions, thus avoiding the use of names completely. With this approach, the public key in an access control certificate effectively identifies a subject, and if possession of the corresponding private key can be proved, a service accepting this certificate can be sure of the identity of the subject and make access decisions simply by examining the access permissions specified in the certificate. Unlike the identity-based approach, the key-based approach integrates the problem of authentication and authorization into one step. Most of the key-based access control models are known as trust management systems. Examples are Simple Public Key Infrastructure (SPKI) [44,45], and KeyNote [46]. We describe conventional trust management systems and analyse a specific vulnerability associated with them in the next section.

Trust Management Systems

In open environments there can exist a number of principals who may send requests to the other principals to access their resources. These principals may be unknown to each other unless they have had interactions before. Therefore, an access control mechanism needs to be maintained securely in a distributed manner, and has to be stored across the entire network to make the appropriate access decision. Thus, a distributed and flexible process for establishment of trust and making authorization decisions has been proposed called *Trust Management*. Trust Management [10, 43, 44, 47–53] is an approach to establish trust and manage authorization relationships among distributed principals that mediates access control. It is divided into two main categories: reputation-based trust management systems, and certificate-based trust management systems. In reputation-based trust management systems, entities establish a trust relationship based on beliefs resulting from past interactions, which predict future behaviour [49–51, 54–58]. In certificate-based trust management systems, entities establish a trust relationship based on some evidence called certificates [10,43,47,48,59–63]. With a given set of certificates that the requester provides to the resource owner (recipient) for accessing specific resources, the recipient makes an access decision that complies with the security policy. This approach not only allows unknown requesters to prove their authorization for some resources, but also provides decentralized control to support delegation of authorization among unknown principals to prop-

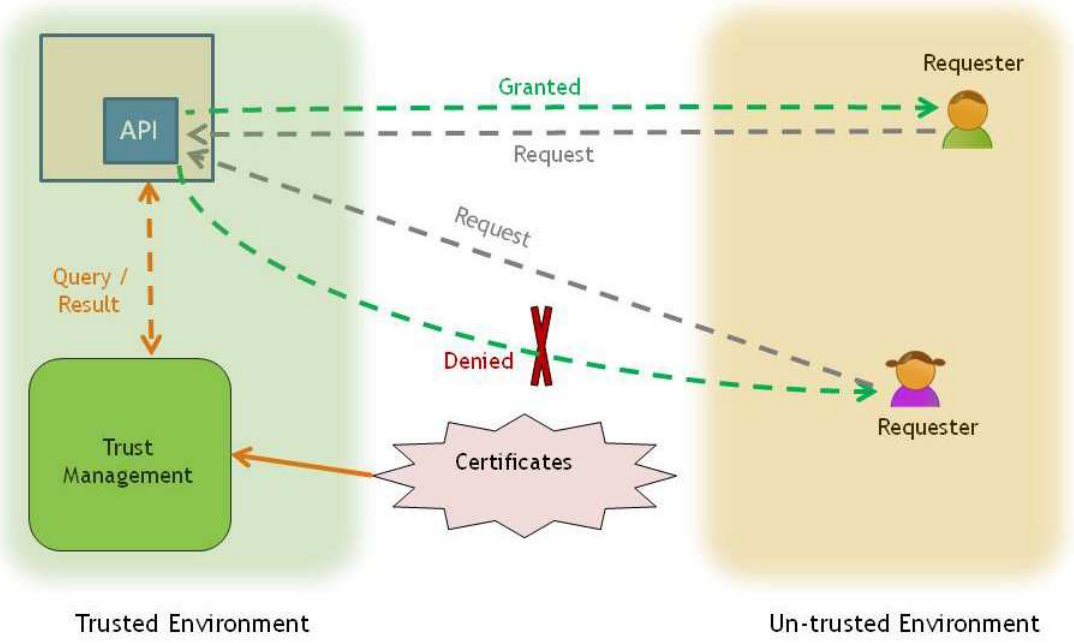


Figure 2.3: An overview on trust management system

agate access rights. In this thesis, when we use the term "trust management" we mean certificate-based trust management.

Trust management systems are more expressive and scalable than classical access control systems for distributed environments. Each principal can define its own access control policy for its resources without the need to rewrite and reinterpret its security mechanism when joining other principals in a distributed environment. It is also independent from the design of individual security policies in applications. The trust management approach is based on the notion of *delegation certificate*, where every delegation certificate delegates some permission from its issuer to its subject. Chains of delegation certificates issued by different issuers may be formed, thus enabling access to be granted in a decentralized manner. Figure 2.3 depicts a trust management system. A principal sends access requests for a resource to the trust management service via an API. The trust management system checks whether the requester has provided the necessary certificates for its request. The response determines whether access to the resource should be granted or denied.

Glossary

To clarify fundamental elements of trust management systems, we now provide a glossary of relevant terms. These terms occur throughout the rest of this thesis.

Principal An entity that can be authenticated via an authentication mechanism in distributed environments. Principals are entities that may be trusted and consequently authorized to perform an action.

Permission A permission specifies the access rights to some resources. Permissions imply a hierarchy of access rights and so a principal holding a permission dominating other permissions can also holds the dominated permissions.

Delegation Propagation of a permission, that a principal holds, to other principals is called delegation.

Policy A set of rules specifying the conditions under which a request may be granted.

Compliance Checker A compliance checker handles the access decisions based on a set of certificates and policies. For example, a bank trusts its employee *Alice* to create a bank account. The bank authorizes *Alice* by issuing a certificate for this purpose. *Alice* presents her request for creating a bank account along with the relevant certificates to the trust management system. The compliance checker checks *Alice*'s access for creating a bank account based on the certificates she provides to the trust management system.

Certificates Cryptographic assertions as proof of authorization for some actions or delegation of permissions to other principals.

Actions Operations on protected resources that need to be controlled by the trust management system.

Authorization The authority to perform an action on some resource by a principal is called authorization. Note that authorization is different from permission. A permission is a specification of some action on some resource. Authorization refers to binding a principal to a set of permissions.

Existing Approaches for Trust Management

Trust management frameworks may vary in terms of the infrastructure that they use to establish trust relationships. However, at a higher level, they all have an automatic way to manage the authorization by building up trust relationships among ordinary principals in open and distributed environments. The term *trust management* was coined when PolicyMaker was introduced by Blaze et al. [14] in 1998. The main idea in PolicyMaker is authorizing decentralized access by checking a proof of compliance. It provides a unified framework for managing security policies, certificates and their trust relationships. Blaze et al. argue that identity based approaches such as PGP [36, 37] and X.509 are limited and are appropriate to only one application, and do not support security for applications that are distributed. In PGP and X.509, a user's public key is linked to the user's identity within the X.509 certificate, and also the user's identity is linked to a set of actions that the user is authorized to perform. The linking of a user's identity to the set of authorized actions should be implemented for each application separately; thus, this approach is inefficient for open and distributed environments. PolicyMaker associates the user's identity with a set of authorized actions by binding a user's public key to the actions they are authorized to perform. The PolicyMaker engine takes the request, certificates, and local policy as input, and the compliance checker provides a proof of compliance, returning `True` or `False`. The `True` or `False` result determines whether or not, respectively, the requester is authorized for performing the action that he/she requested. Certificates are mainly used for trust delegations. A trusted principal issues a certificate to a non-trusted principal to become trusted and, in turn, the new trusted principal issues a similar certificate to another principal, and so on. The principal receives the permission for a resource through a delegation certificate. Delegation is an important feature that provides decentralized scalability for trust management and access control. Certificate structures in PolicyMaker are arbitrary and can be encoded within any programming languages. Moreover, permissions are specified arbitrarily and the global uniqueness of a permission specification depends on the expertise of the security administrator, who defines these permissions, to

take care of providing a globally unique interpretation for them.

KeyNote [46, 47], which is the second generation of PolicyMaker, is also based on certificates and local policies. In KeyNote, certificates and policies have the same syntax and are both referred to as assertions. The only difference between certificates and policies in either KeyNote or PolicyMaker is that certificates are signed assertions and policies are not signed because they are locally trusted by KeyNote trust management. A *Principal* is identified by a principal identifier which is a public key. A principal signs certificates and distributes them across an untrusted environment where they are used by other KeyNote trust management systems. A principal P may request some action while the other principal Q issues a certificate related to that action and delegates the permission for that action to the principal P. All assertions in KeyNote are expressed in a language managed by the KeyNote *compliance checker*. In receiving a query, the KeyNote trust management constructs a graph where each node corresponds to a principal's public key and an edge corresponds to the assertion that represents delegation of a permission. The root of the graph is the policy assertion and an algorithm tries to construct a path to the requester's public key, using certificates to satisfy the compliance checker for acceptance or rejection of a request. Despite the fact that PolicyMaker leaves the verification of the requester's signature key with the application, KeyNote accomplishes this task inside the trust management system. The compliance checker inside the trust management system evaluates the request and the result of the query is returned to the application. The KeyNote compliance evaluation value is flexible and depends on the application design, while in PolicyMaker compliance values are the binary values of `True` or `False`. KeyNote defines a specific language to construct certificates, compared with PolicyMaker where certificates can have arbitrary structure encoded within any programming languages. These features make KeyNote more flexible to use as a trust management system to integrate into applications rather than the previous trust management system, PolicyMaker. However, similar to PolicyMaker, in KeyNote permissions are specified arbitrarily and the global uniqueness of a permission specification depends on the expertise of the security administrator, who defines these permissions, to take care of providing a globally unique interpretation for them.

SPKI/SDSI is another certificate-based mechanism that can be viewed as a trust management system. This mechanism is a combination of two different approaches, SPKI (Simple public key infrastructure) and SDSI (Simple Distributed Security Infrastructure) [43]. SPKI/SDSI was combined in 1999 with the goal of

easy use of public key infrastructure (PKI) [64]. The combined SPKI/SDSI [60] uses the SDSI approach for name certificates and the SPKI approach for authorization certificates. It addresses the naming of principals, creation of groups of principals, associating permissions for actions with principals, and the delegation of permissions from one principal to another. A SPKI/SDSI name certificate is inherited from SDSI's name certificate [43], and provides a way of naming public keys that are meaningful in a principal's name space. A principal's name space is specified by its public key. Each principal chooses an arbitrary name for the other principal in its name space and binds that arbitrary chosen name to its public key to refer to that principal in a global manner. This binding is done by issuing a name certificate. The name N, bound to a principal's name space identified by its public key K is called a *local name*, denoted as $(K\ N)$. A name certificate $(K\ N) \longrightarrow P$ is a statement signed by the owner of public key K that principal P is defined to be N in K's local name space. A name certificate can also be issued to refer to another local name principal, called an *extended name*. Moreover, a name certificate can be issued to specify group membership. A SPKI/SDSI authorization certificate is inherited from the SPKI approach [60]. The SPKI mechanism identifies principals only as public keys but allows binding of permission to those keys and delegation of permissions from one key to another by issuing an *authorization certificate*. An authorization certificate denoted by: $P \stackrel{X}{\Longrightarrow} Q$ indicates that principal P delegates authority for permission X to principal Q. SPKI/SDSI inherits this approach for authorization and delegation of permissions among principals that are identified by either their public keys or local names.

PolicyMaker, KeyNote, and SPKI/SDSI have similar approaches where certain permissions are delegated from their issuers to the other principals. Other principals may delegate further those permissions.

Role-based Trust management (RT) [9, 65] and secPAL [10] are other approaches that are based on constrained Datalog [66, 67]. The RT is influenced by SDSI and Delegation Logic [48, 68, 69]. Local names in SDSI correspond to *roles* in RT. The notion of *roles* unifies concepts such as conditions in KeyNote, names in SDSI, and permission tags in SPKI. For example, if P is a principal, N and R are roles, $P.N$ in RT may correspond to the local name $(P\ N)$ in SDSI and $P.R$ may correspond to a permission which is defined in the condition field in Keynote and permission tags in SPKI. RT is a family of a number of languages with a common basic structure. The core basic language of RT is RT_0, and additional features were introduced as parametrized roles in RT_1, delegation of activation of roles in

RT^D, and manifold roles and role-product operators in RT^T. In general, in the RT family, principals are roles who also can define new roles, issue certificates, and make requests. Principals may be identified by a public key, or by a user account. For example, *Alice* is a principal that may be identified by her public key k_A and is a member of role "Bank1.Manager". Permissions are also represented by roles. Membership of a role assigns permissions that are defined to authorize the principal who is a member of that role. For example, the permission for opening an account on Bank1 can be represented by role "Bank1.openAccount". Hence, a principal that belongs to the role "Bank1.openAccount" is authorized to open an account in Bank1. In RT each principal may have a role and the traditional permissions that were mentioned in PolicyMaker, KeyNote, SPKI/SDSI and delegation of permissions among principals are unified to the role assignment concept.

secPAL is another approach for trust management which was proposed in [10,70]. It is a logic-based policy language that addresses establishment of trust and management of authorization in open and distributed environments with no predefining trust relationships. Policies and certificates are expressed by predicates, and constraints within logical clauses. Authorization and delegation to principals are defined in predicates. Further delegation is also supported to either fixed or arbitrary depth of delegation. Assuming the same scenario in the previous example for RT, the *BankManager* may decide to delegate permission for creating an account (*creatAccount*) in Bank1 to its employees and allow its employees to delegate of this permission to other principals in different domains. This can be expressed in secPAL as the following assertions:

BankManager says X can say$_\infty$ *createAccount*

if X is an employee

delegation is expressed by "*can say*$_\infty$" construct and the suffix "∞" means that the statement " *can say*$_\infty$ *createAccount*" can be delegated further with no depth limit. All assertions are signed by their issuers, so the above assertion is signed by *BankManager*. Returning back to the previous scenario, *Alice* can delegate *createAccount* to *Bob* if the following statement exists:

BankManager says Alice can say$_\infty$ *createAccount*

if Alice is an employee

then the statement *"createAccount is accepted"* can be derived. Note that the *"is accepted"* is an example of an attribute. secPAL supports expressing a wide range of policies with role hierarchies, parametrized roles, and threshold constraints using statements close to the natural English language.

2.3 Subterfuge Vulnerability

Subterfuge is a deceptive behaviour with the goal of evading the actual control intention of a security mechanism [71, 72]. Many existing trust management frameworks are explicit in their assumption that principals are uniquely identified, however the literature has generally not been as prescriptive regarding the uniqueness of permissions. Authorization certificates are used to specify delegation of permissions among principals. Permissions are bound to the certificate issuer's public key to facilitate access control in a decentralized approach. Since binding permissions to public keys is not particularly meaningful to principals, a series of delegation certificates, that a principal uses to prove authorization for some action, may not return a result that reflects the exact intention of all participants in the existing chain. A principal can somehow bypass the actual intention of a series of delegation certificates via some indirect but apparently authorized route. Delegation subterfuge refers to the inconsistency problem in delegation of a permission that can arise when there is ambiguity concerning the uniqueness and interpretation of a permission. In the following sections we investigate the subterfuge vulnerability in the trust management frameworks discussed in section 2.2 with examples and more details.

2.4 Delegation Subterfuge in Existing Trust Management Approaches

A trust management aids principals to carry out a task through an automated decision making process. Malicious principals are incompetent principals who are not expected to carry out an action based on the actual control intention of the security mechanism. They attempt to perform the action by using a vulnerability in a security mechanism via a sequence of deceptive but apparently legitimate behaviours. Subterfuge is one of these deceptive behaviours performed by incompetent and malicious principals with the goal of evading the intended control of

a security mechanism and misleading the accountability tracking for a permission. Although, existing trust management languages have uniquely identified assumptions for principals; they do not provide a systematic method for unique interpretation of permissions. We explain subterfuge by giving some examples of delegation in the mentioned trust managements. Although, they all rely on some form of global name providers to ensure that different parties get the right name for resources, still malicious principals may bypass the actual intention of a delegation.

2.4.1 Delegation Subterfuge in KeyNote

KeyNote relies on the Internet Assigned Number Authority (IANA) [73] as a name service provider to ensure that different parties get the right name for resources. However, global name providers are not security administrators; they only provide each name with a unique meaning and have no control over how names are used. Principals from different name spaces may still use arbitrary specifications to represent their own resources. Assume that an electronic banking system in *Bank1* uses KeyNote trust management to build and manage trust relationships. The *Bank1Manager*, owner of public key k_{BM}, issues a certificate that states *Alice*, owner of public key k_A, is trusted to *create account* for *Bank1*. This is represented by the following delegation certificate:

```
KeyNote-Version:  "2"
Comment:  Certificate(1)
Authorizer:  k_BM
Licensees:  k_A
Condition:  Action=="create account";
signature:  by k_BM
```

Alice may decide to delegate permission for *create account* in *Bank1* to *Bob*, owner of public key k_B, who is an employee of *Bank3*. She issues the following certificate:

KeyNote-Version: "2"
Comment: Certificate(2)
Authorizer: k_A
Licensees: k_B
Condition: Action=="create account";
signature: by k_A

The electronic banking system in *Bank2* also uses KeyNote as its trust management system. *Eve*, the owner of public key k_E, is an employee of *Bank2*. *Eve* delegates the permission for *create account* in *Bank2* to *Bob* who is an employee of *Bank3*. *Eve* issues the following certificate:

KeyNote-Version: "2"
Comment: Certificate(3)
Authorizer: k_E
Licensees: k_B
Condition: Action=="create account";
signature: by k_E

Bob is willing to delegate the permission *create account* in *Bank2* that he obtained from *Eve* to *Dave*, the owner of public key k_D to access resources of *Bank2*. *Bob* issues the following certificate:

KeyNote-Version: "2"
Comment: Certificate(4)
Authorizer: k_B
Licensees: k_D
Condition: Action=="create account";
signature: by k_B

Dishonest *Dave* obtains certificates (1) and (2) and represents the chain of certificates (1),(2),(4) as proof of his authorization to create account in *Bank1*. However, *Bob*'s intention by issuing certificate (4) to *Dave* was to allow *Dave* to

present the certificates (3), (4) as a proof of authorization to *Bank2*.

2.4.2 Delegation Subterfuge in RT

The RT family [65] uses Application Domain Specification Documents (ADSDs) [9] to ensure the globally unique naming. Although ADSDs provide a unique interpretation for each name, the principals of different domains may still use arbitrary names to represent their own resources. For example, an electronic banking system enables *Alice* to activate the role "Bank1.employee" to use in a session. The role "Bank1.employee" is authorized for permission "Bank1.createAccount". *Alice* delegates this role to *Bob* by issuing the delegation certificate for this purpose denoted as:

$$Alice \xrightarrow{act\ as\ Bank1.employee} Bob$$

"*act as Bank*1.*employee*" is called "role activation". Suppose that malicious *Eve* obtains this role activation "*act as Bank*1.*employee*". She first intercepts the above delegation certificate and then delegates this role to *Bob* by issuing the certificate:

$$Eve \xrightarrow{act\ as\ Bank1.employee} Bob$$

Bob may further delegate this role activation to *Dave* by issuing the certificate:

$$Bob \xrightarrow{act\ as\ Bank1.employee} Dave$$

Dave requests to create an account in Bank1 in the capacity of "*act as Bank*1.*employee*" that can be represented by the following statement:

$$Dave \xrightarrow{act\ as\ Bank1.employee} Bank1.creatAccount$$

This request is granted because there is a chain of delegation from *Alice* to *Dave* for the role activation "*act as Bank*1.*employee*". However, *Bob*'s intention by delegating the role activation "*act as Bank*1.*employee*" to *Dave* was that *Dave* present the delegation chain from *Eve* to *Dave* for his request. This happens because *Bob* does not understand that *Eve* has no authority over "*act as Bank*1.*employee*" activation role. Therefore, the intercepted delegation statement can be used by *Dave* to create an account for *Bank*1.

2.4.3 Delegation Subterfuge In secPAL

We consider the previous scenario, the electronic bank system, and using the secPAL policy language to investigate the subterfuge vulnerability in secPAL. In the following example, we show how using the secPAL policy language results in delegation subterfuge. We identify principals by their names as *Bank1Manager, Alice, Bob, Eve, Dave,...* where a public key is associated to each of these names in the implementation of secPAL as $k_{BM}, k_A, k_B, k_E, k_D, ...$ respectively. *Bank Manager* issues a delegation statement for all of its employees to permit them to create an account in *Bank1*. Note that, the constructor "say_∞" means that the permission in that statement can be delegated further. This delegation statement is expressed as the following assertion in secPAL:

$$\begin{aligned} &\textit{Bank1 Manager says } X \textit{ can say}_\infty \textit{ createAccount} \\ &\textit{if } X \textit{ is an employee} \\ &\textit{Bank1 Manager says Alice is an employee} \end{aligned}$$

and therefore these statements can be inferred:

$$\textit{Bank1 Manager says Alice can say}_\infty \textit{ createAccount}$$

Alice delegates this permission to *Bob* by issuing the following assertion:

$$\textit{Alice says Bob can say}_\infty \textit{ createAccount}$$

The manager of *Bank*2 issues a similar statement for permitting its employees to create an account. Thus, *Bank2 Manager* issues the following assertion:

$$\begin{aligned} &\textit{Bank2 Manager says } X \textit{ can say}_\infty \textit{ createAccount} \\ &\textit{if } X \textit{ is an employee} \\ &\textit{Bank2 Manager says Eve is an employee} \end{aligned}$$

and the following statement can be satisfied for $X = Eve$:

$$\textit{Bank2 Manager says Eve can say}_\infty \textit{ createAccount}$$

Malicious *Eve* delegates the permission for creating an account to *Bob* for further delegation in the following assertion:

$$Eve\ says\ Bob\ can\ say_{\infty}\ createAccount$$

Bob unwittingly delegates the *createAccount* permission, that he obtained from *Eve*, to *Dave* to present to *Bank2* for creating an account. Note that, the constructor "say_0" means that the following statement is only accepted if it is directly asserted by *Dave* and is not deduced by a chain of delegation assertions.

$$Bob\ says\ Dave\ can\ say_0\ createAccount$$

However, *Dave* can present the following collection of assertions along with his request to *Bank1*:

$$Bank1Manager\ says\ Alice\ can\ say_{\infty}\ createAccount$$
$$Alice\ says\ Bob\ can\ say_{\infty}\ createAccount$$
$$Bob\ says\ Dave\ can\ say_0\ createAccount$$

The intention of *Bob* by delegating the permission for creating account was to allow *Dave* to use the following chain of delegation assertions:

$$Bank2Manager\ says\ Eve\ can\ say_{\infty}\ createAccount$$
$$Eve\ says\ Bob\ can\ say_{\infty}\ createAccount$$
$$Bob\ says\ Dave\ can\ say_0\ createAccount$$

This subterfuge happens because of poor expression in the statements by the issuers. Good assertion construction depends on the issuers' expertise. However, the issuer of assertions does not have a clear view of all available naming schemes across other domains. So, in the above scenario, *Bob* was confused because of misinterpretation of permission *creatAccount* that he received from two different principals.

2.4.4 Delegation Subterfuge In SPKI/SDSI

In SPKI/SDSI permissions are identified by tags or S-expressions. Authorization certificates specify either authorization or delegation of permissions to principals. Permissions are bound to the public key of the certificate's issuers. Since binding permissions to public keys does not provide a meaningful interpretation for principals, a series of delegation certificates that a principal uses to prove its authorization for some resources may not reflect the exact intention of those delegation certificates. This issue is described in more detail in the following examples. In the following examples, the permission tag:

$$T = (tag(download\ X))$$

refers to the permission specification defined in the SPKI/SDSI framework for downloading AlbumX. We refer to this tag simply as *AlbumX*.

Example 1 *Atlantic Records* signs a music publishing contract with a *Music Broker Service* (owner of public key k_B), that by their agreement, the broker service would be able to issue permission for its customers to download from *Atlantic Records* website by paying $\$10$ for *AlbumX*. *Atlantic Records* (owner of key k_A) adds *Music Broker Service* to its contract list which is identified by the SDSI name ($k_A\ Contracts$), that all members of SDSI group ($k_A\ Contracts$) are authorized to download *AlbumX* by paying $\$10$. Principal k_E registers in *Music Broker Service* and is identified by the local name ($k_B\ Customers$). k_E sends the request to download the *AlbumX* from *Atlantic Records* and presents the following certificates as proof of its authorization (notations are explained in 2.4):

$$k_A \overset{AlbumX}{\Longrightarrow} (k_A\ Contracts)$$

$$(k_A\ Contracts) \rightarrow k_B$$

$$k_B \overset{AlbumX}{\Longrightarrow} (k_B\ Customers)$$

$$(k_B\ Customers) \rightarrow k_E$$

The expected certificate chain between *Atlantic Records* company and *Music Broker Service* is depicted in Figure 2.4.

On the other hand, *Motown Records* is another music company that also signs a publishing contract with the same *Music Broker Service* (owner of public key

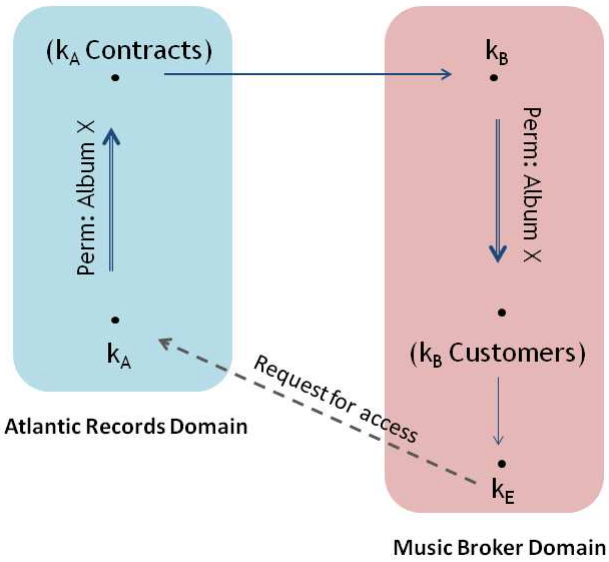


Figure 2.4: Expected chain for downloading *AlbumX*

k_B), which by their agreement the *Music Broker Service* would be able to issue permission to its customers to download from *Motown Records* by paying $\$1$ for *AlbumX*. *Motown Records*, the owner of public key k_M, adds k_B to its contracts list specified by SDSI group (k_M *Contracts*), where all the members of this group are authorized for *AlbumX*. Dishonest k_E can collect all the certificates and present the following certificates to *Motown Records* to pay less for *AlbumX*:

$$k_M \overset{AlbumX}{\Longrightarrow} (k_M \ Contracts)$$

$$(k_M \ Contracts) \rightarrow k_B$$

$$k_B \overset{AlbumX}{\Longrightarrow} (k_B \ Customers)$$

$$(k_B \ Customers) \rightarrow k_E$$

However, k_B's intention when delegating permission *AlbumX* to the SDSI group (k_B *Customers*) is that k_E uses the expected chain as proof of authorization to download from *Atlantic Records* and make purchases by paying $\$10$. Unknown to k_B, dishonest k_E collects all other certificates and uses the above unexpected chain to make a cheaper purchase by paying $\$1$ to *Motown Records* rather than $\$10$ to *Atlantic Records*. The unexpected certificate chain between *Motown Records* company and *Music Broker Service* is depicted in Figure 2.5.

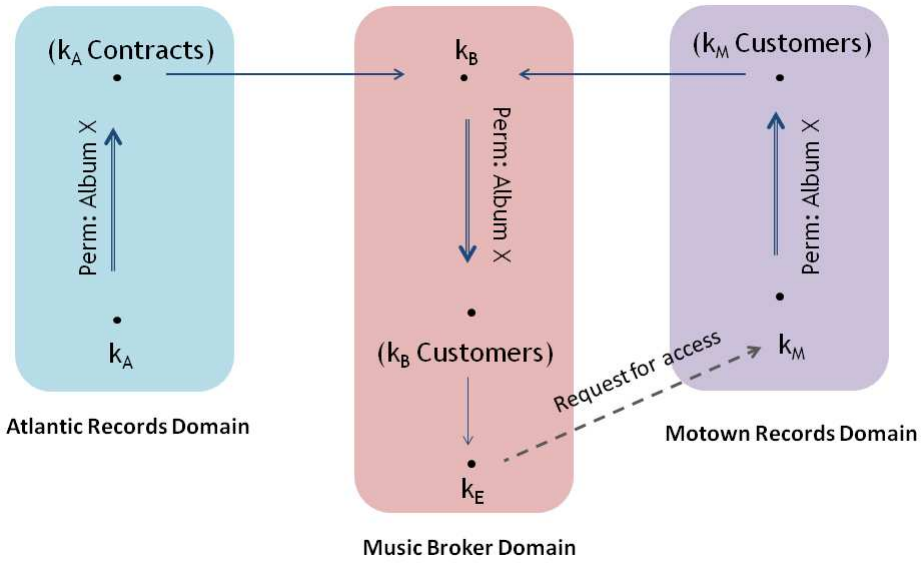


Figure 2.5: Unexpected chain for downloading *AlbumX*

Example 2 *Eve* (the owner of public key k_E) sets up a bogus company and masquerades as k_M, the owner of the bogus company. Note that Eve is also the owner of public key k_M. *Eve* encourages the *Music Broker Service* (owner of public key k_B) to join its company and consequently to its group (k_M *Contracts*). *Eve*, masquerading as k_M, delegates permission on downloading *AlbumX* which is the same as the permission that k_B already holds from a different company. k_B does not realize this and delegates the permission on downloading *AlbumX* to (k_B *Customers*) and consequently to k_C as a customer of *Music Broker Service*. k_C normally would not be expected to hold this permission. This confusion occurs as k_B might have too many certificates to manage and does not track which permission should be associated to which company.

In this case, k_B's intention by registering in k_M's company was purchasing minimum quality and cheaper services for its ordinary customers. However, k_C as an ordinary customer of *Music Broker Service* can present the following certificates to access *Atlantic Records* and download the higher quality and more expensive product:

$$k_A \overset{AlbumX}{\Longrightarrow} (k_A\ Contracts)$$

$$(k_A\ Contracts) \rightarrow k_B$$

$$k_B \overset{AlbumX}{\Longrightarrow} (k_B\ Customers)$$

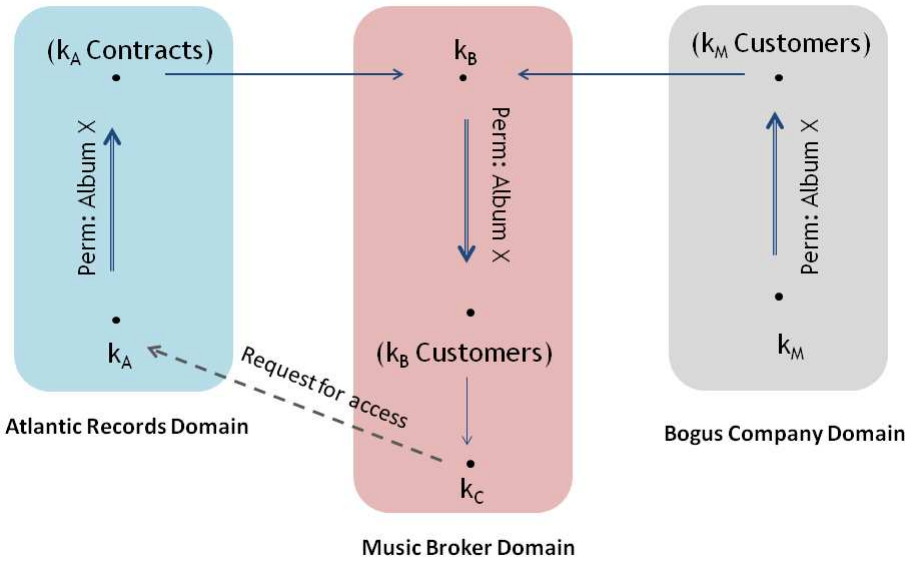


Figure 2.6: Subterfuge scenario by establishing a bogus company

$$(k_B\ Customers) \rightarrow k_C$$

Note that k_C pays the *Music Broker Service* and the *Music Broker Service* pays its contract companies. Hence, k_B presumes that it should pay k_M's company, but *Atlantic Records* charges k_B for principal k_C's purchase. This scenario is depicted in Figure 2.6.

Example 3 In Figure 2.7, *AtlanticRecords* delegates the permission *AlbumX* to its group with the local name $(k_A\ Contracts)$. k_C registers in the group $(k_A\ Contracts)$ and is not aware of the permissions that this group has.

If k_C delegates the permission *Atlantic.com/AlbumX* that she received from *Atlantic Records* to another principal such as k_M, the expected chain that k_M presents for requesting access to *AlbumX* on *Atlantic Records* would be the following chain:

$$k_A \overset{Atlantic.com/AlbumX}{\Longrightarrow} (k_A\ Contracts)$$

$$(k_A\ Contracts) \rightarrow k_C$$

$$k_C \overset{Atlantic.com/AlbumX}{\Longrightarrow} k_M$$

On the other hand, k_E and k_M are principals that can use the vulnerability in permission naming to perform the following attack on *Atlantic Records* to access *AlbumX* illegitimately. However, if k_C is not willing to give the permission

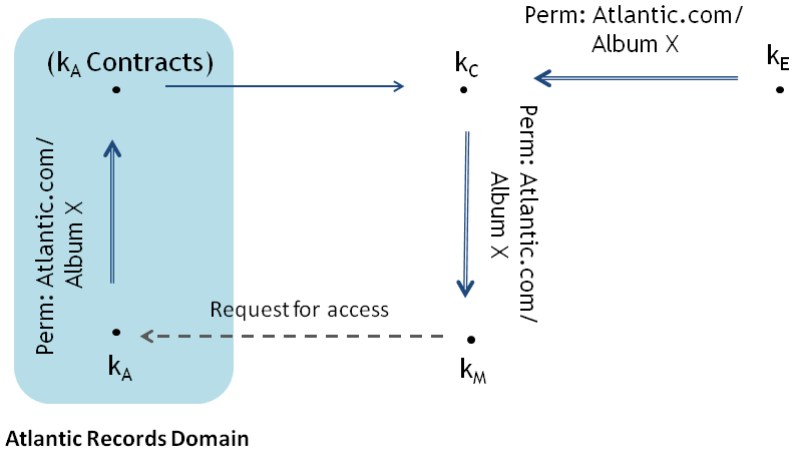


Figure 2.7: Subterfuge in the delegation of permission Atlantic.com/AlbumX

Atlantic.com/AlbumX that she obtained from *Atlantic Records* to k_M, k_E can collude with k_M and intercepts $k_A \overset{Atlantic.com/AlbumX}{\Longrightarrow} (k_A\ Contracts)$. k_C has no idea about the permissions that the group $(k_A\ Contracts)$ has. In addition, k_C does not realize k_E has no authority over permission *Atlantic.com/AlbumX*, and delegates the permission *Atlantic.com/AlbumX* to k_M which it received from k_E. Therefore, k_C's intention when delegating permission on *Atlantic.com/AlbumX* to k_M is that k_M uses the expected chain:

$$k_E \overset{Atlantic.com/AlbumX}{\Longrightarrow} k_C$$

$$k_C \overset{Atlantic.com/AlbumX}{\Longrightarrow} k_M$$

as proof of authorization to k_E. However, dishonest k_M obtains all the certificates in the chain and presents the following collection of certificates to *Atlantic Records* to access *AlbumX* without k_C's knowledge.

$$k_A \overset{Atlantic.com/AlbumX}{\Longrightarrow} (k_A\ Contracts)$$

$$(k_A\ Contracts) \rightarrow k_C$$

$$k_C \overset{Atlantic.com/AlbumX}{\Longrightarrow} k_M$$

Note, even if k_E obtains permission *Atlantic.com/AlbumX* legitimately, they can also collude with k_M to perform the above attack.

2.5 Summary

This chapter has presented an overview of the research in access control models. It has also briefly reviewed the research on distributed access control, starting from the identity-based approaches to the modern key-based approaches. The identity-based approach uses certificates as assertions for the binding of a public key with a name. The representative work in this area is the X.509 Public Key Infrastructure (PKI), Privilege Management Infrastructure (PMI), and Simple Distributed Security Infrastructure (SDSI). The newer key-based approach associates a public key directly with permissions, thus avoiding the use of names. Simple Public Key Infrastructure (SPKI), and PolicyMaker/KeyNote are representative work in this area. In particular, PolicyMaker/KeyNote with an integrated approach to the specification of security policies and trust relationships underlies trust management systems.

Trust Management like many other protection techniques, provides operations that are used to control access. As with any protection mechanism the challenge is to make sure that the mechanisms are configured in such a way that they ensure some useful and consistent notion of security. We showed how poorly characterized permission specifications within cryptographic certificates can lead to authorization subterfuge during delegation operations. This subterfuge results in a vulnerability concerning the accountability of the authorization provided by a delegation chain. The delegation operations in the chain may not reflect the actual intention of the security mechanism. The challenge here is to ensure that permissions can be referred in such a way that properly reflects their context. Since permissions are intended to be shared across local name spaces their references must be global. We discussed some ad-hoc strategies to ensure global permissions. In particular, we consider the use of global name services and public keys as the sources of global identifiers. However, the design of a security mechanism should not rely on ad-hoc methods, rather, should be formalized in a systematic way to prevent subterfuge.

Chapter 3

Subterfuge Safe Trust Management

This chapter introduces Subterfuge Safe Trust Management (SSTM). SSTM represents and expands the works that were introduced in [74, 75]. We begin with an overview of the SSTM infrastructure, outlining the basic concepts and key features in section 3.1. Section 3.2 introduces the *Subterfuge Safe Authorization Language* (SSAL), a policy language for specifying trust-related policies. In section 3.3, a formal foundation for SSAL is given. Section 3.4 outlines the potential threats to SSTM. Section 3.5 addresses certificate chain discovery for SSTM. Finally, a summary of the chapter is presented in section 3.7.

3.1 Overview on SSTM Infrastructure

SSTM is a framework for specifying, expressing, and managing trust and authorization relationships among distributed principals in open environments. It provides a trust model for subterfuge safe delegation of permissions in large scale distributed systems without relying on a pre-agreed global naming service or a super security administrator. In SSTM, a principal may be a person, an organization, a computer process, or any other entity authenticated by some authority. Principals are identified by either public keys or locally defined names that are identified uniquely in global environments. Permissions are considered to be specified locally in some principal's name space, however, they have globally unique interpretations. A principal can delegate a permission to others. A principal describes its own security policy in its name space, therefore, a language is designed

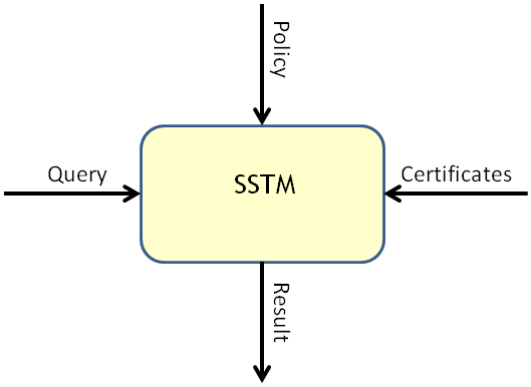


Figure 3.1: Overview of SSTM framework

for policy specification, called Subterfuge Safe Authorization Language (SSAL). Figure 3.1 shows a general overview of the SSTM framework.

3.2 Subterfuge Safe Authorization Language

Subterfuge Safe Authorization Language (SSAL) is intended to be used by principals to specify their policies regarding subterfuge safe trust/delegation relationships. SSAL introduces usage of localPermissions instead of using conventional specification of permissions in existing trust management systems. This section describes the syntax of the language and provides a formal semantics using formal methods. The three main concepts in the language: principals, permissions, and delegation will be discussed in the following sections.

3.2.1 Principals

There are two types of principals in SSAL, both of which are uniquely identifiable in a global manner. A principal can be specified by a *public key*, and therefore is globally unique. A principal can also be an arbitrarily chosen name associated with a public key called *local name*, therefore is globally unique as well. Local names are inherited from Simple Distributed Security Infrastructure (SDSI) [43] and are formed by linking principals to arbitrary chosen names. Each principal has its own name space, identified by its global unique identifier (either public key or local name), and can choose independently an arbitrary name to refer to another principal in its own name space. Binding a principal that is recognized by its local name to a name identifier provides an *extended local name*. An

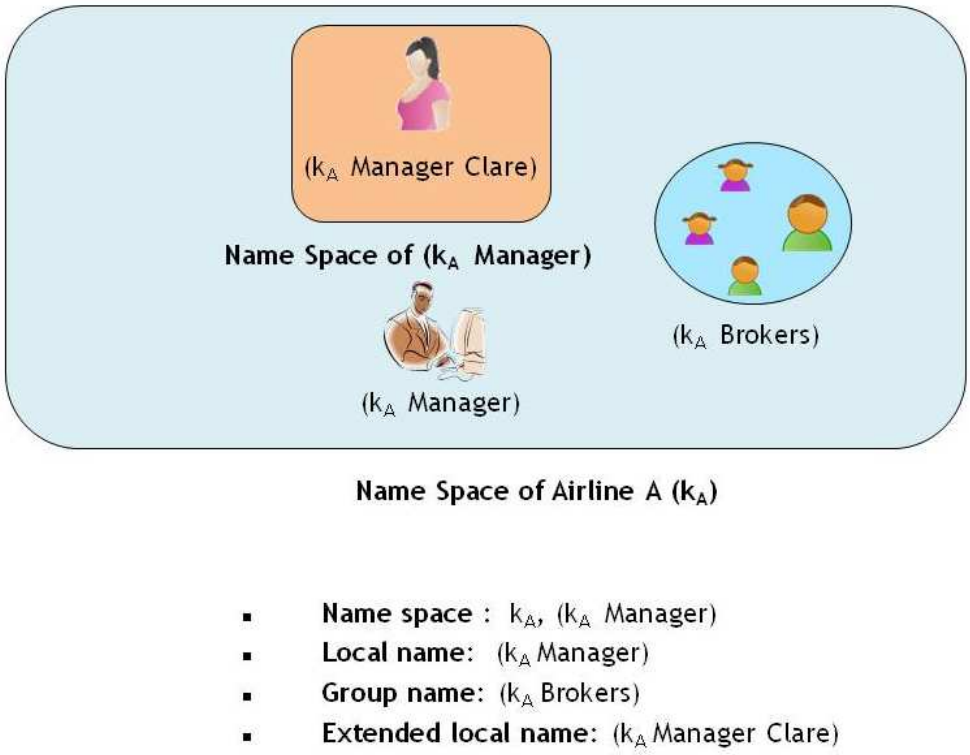


Figure 3.2: Illustration of a local name, a group, and an extended local name.

extended local name is a public key followed by two or more names. Note that the arbitrarily chosen name can be a name for a group of principals called a *group*. A group is a public key followed by a name. For example, the airline A, identified by its public key k_A, can choose an arbitrary name for its manager and define the local name $(k_A\ Manager)$ as a unique reference for its manager in global environments. The airline A can also define a group of brokers as $(k_A\ Brokers)$. Assuming that the airline A's manager has an employee called $Clare$, which is unique in the name space of the manager, then $Clare$ can be identified by the extended local name $(k_A\ Manager\ Clare)$. Figure 3.2 illustrates this example. Extended local names and groups do not have a separate definition; their meaning is defined in terms of the meaning of the local names. In this thesis, we refer to local names, extended local names, and groups simply as local names.

Public key

Public key cryptographically enables entities to securely communicate in insecure open distributed environments, and reliably verifies the identity of an entity via digital signatures. The public key is meant to be available for anyone in an open

```
---- BEGIN ... PUBLIC KEY ----
Comment: Public key for Alice
AAAAB3NzaC1yc2EAAAABIwAAAIEA1on8gxCGJJWSRT4uOrR13mUaUk0hRf4RzxgE
SZ1zRbYYFw8pfGesIFoEuVth4HKyF8k1y4mRUnYHP1XNMNMJl1JcEArC2asV8sHf
6zSPVffozZ5TT4SfsUu/iKy9lUcCfXzwre4WWZSXXcPff+EHtWshahu3WzBdnGxm
5Xoi89zcE
---- END ... PUBLIC KEY ----
```

Figure 3.3: Example of a public key

distributed environment. The private key is meant to be confidential, no one but the owner is allowed to access it. The private key can be used to sign documents such as statements, certificates, etc. Since it is kept privately only the owner can use the private key. The public key can be used to verify a signature. Therefore, once a document is signed by a private key anyone in the open environment can verify the signature with the corresponding public key. Given these properties of a public and private key pair, a principal can be represented by its public key as a global unique identifier. We assume that no two entities share the same public key and thus are globally unique and verifiable by others. Otherwise, if two or more entities have the same public key they could impersonate each other. While it is theoretically possible to generate the same public key, given the sizes of the keys (1024 bits, about 309 digits), it is extremely unlikely to generate the same public key in reality. Figure 3.3 depicts a Secure Shell (SSH) Public Key File Format [76].

For simplicity in this thesis, we use k followed by a small number, letter or word to refer to the public keys. For example, k_A might represent the public key in Figure 3.3.

Local Name

The syntax of the local name is the public key followed by a sequence (one or more) of arbitrarily chosen names. A local name is an arbitrary name N that principal P (identified by either a public key or a local name) chooses for principal Q in its (P's) name space. Principal P refers to principal Q in its name space as N and in the global environment as $(P\ N)$. $(P\ N)$ is called the local name for Q and their relationship is represented using the *speaks for* relation whereby the statement principal Q *speaks for* local name $(P\ N)$ is denoted as:

$$(P\ N) \longrightarrow Q$$

Any statement that is signed by Q can be viewed as originating from $(P\ N)$. For example, an airline owner such as A, represented by its public key, k_A, signs a statement that *Bob*, the owner of public key k_B, is acting as its broker:

$$(k_A\ Brokers) \longrightarrow k_B$$

which means that a message signed by the broker k_B can be considered to be originated from $(k_A\ Brokers)$.

Name Certificate

The local name and its relationship with other principals is identified by issuing a *name certificate*. The principal P which signs the certificate is the issuer of the name certificate, N is the locally chosen name in the name space of P, and Q is the subject of the certificate that the issuer P refers to with the locally chosen name N. The subject of the name certificate can be either a public key, or local name. A name certificate is represented as the following where s_K means the certificate is signed by the owner of public key K (i.e. P):

$$\{\!|N, Q|\!\}_{s_K}$$

It states that principal Q is referred to by the name N in a principal's name space that is identified by public key K. For example, the certificate $\{\!|\mathsf{Bob}, k_B|\!\}_{s_{k_A}}$ specifies that *Bob* is the name that the owner of public key k_A arbitrarily chose to refer to (the owner of) k_B in her name space. The *speaks for* relation can be inferred from the name certificate whereby the name certificate $\{\!|N, Q|\!\}_{s_K}$ implicitly states that principal Q *speaks for* the principal $(K\ N)$. It means that a message signed by principal Q can be considered to be originated from N in the name space of a principal identified by public key K.

Definition *"B speaks for A" if and only if there exists a certificate where B is the subject and A is the certificate issuer's public key followed by an arbitrary chosen name (local name). When Principal B makes a request, it can be interpreted that the request came from A.*

The following rewrite rule provides a *speaks for* interpretation for name certificates. Given Principal (local name or public key) Q, name N, public key K, the following rule will be interpreted as *speaks for* relation, that Q *speaks for* $(K\ N)$:

$$\frac{\{\!|N, Q|\!\}_{s_K}}{(K\ N) \rightarrow Q}$$

In addition to a single name, a name certificate may specify a group name, where N represents the name for that group and can be bound to several principals that are members of the group. Every member of the group *speaks for* the entire group. For example, an airline owner, Alice, represented by her public key k_A, specifies a group of brokers. All she needs to do is give each member of the group a name definition ($k_A\ Brokers$) and issue one certificate. It is not necessary to issue a new certificate for each individual member. If she decides to add some members, she needs only issue a new name certificate for that principal. k_A signs a statement that k_B is a member of its brokers (i.e. *speaks for* ($k_A\ Brokers$)):

$$\frac{\{\!|Brokers, k_B|\!\}_{s_{k_A}}}{(k_A\ Brokers) \longrightarrow k_B}$$

This means that a message signed by the broker k_B can be considered as originating from ($k_A\ Brokers$). The *speaks for* relation between k_B and ($k_A\ Brokers$) is denoted by the following:

$$(k_A\ Brokers) \longrightarrow k_B$$

3.2.2 Permissions

Permissions are a set of rights that a principal may issue or obtain in a network. They are specifications that define access rights to specific principals and/or groups of principals. They are defined as an approval to perform an action on one or more protected resources. A permission could be a simple action on a file system such as read, write, update, or a complex specification such as a method invocation in an object-oriented system. As discussed in chapter 1, existing trust management systems such as SPKI/SDSI, and KeyNote allow specifying arbitrary permissions. In these systems it depends on the experience of the principal who defines the permission for protected resources. However, in distributed environments, none of the principals have a complete overview of the name schema that other principals use for specifying arbitrary permissions for their resources; therefore, they are vulnerable to the subterfuge problem. SSAL introduces a systematic way of specifying subterfuge-safe permissions (rather than relying on an ad-hoc method) called *localPermission* [74].

localPermission

A new notion for a globally unique permissions called *localPermission* is introduced in SSAL. A localPermission is a permission specification that is bound to its originator name space and has therefore a globally unique interpretation in open environments. A localPermission is formed by a principal followed by a permission specification. The reason for introducing localPermissions in SSAL was to provide a globally unique interpretation for permission specification to prevent ambiguity, and therefore subterfuge, in delegations. In other words, a principal receiving two identical permission specifications cannot misuse the permissions for non-intended purposes, since the permissions have globally unique interpretations and refer clearly to a global context. localPermissions provide a reliable scheme for naming permissions relative to their originator's public key or local names. In localPermission definition, permissions are arbitrarily chosen specifications. By binding permissions to their originator's public key or local name, they will have a global unique interpretation while preserving their locality to their originator's name space.

Definition *A localPermission is an arbitrary permission specification that a principal defines to access its resources in its name space. The arbitrary defined permission specification is bound to the originator's name space and refers to a global unique context.*

Each principal can define its own permission specification referencing a global unique context. The global unique context is the signed value of that permission specification signed by the permission originator P (where K is the public key of P), represented as $\{\!|Perm|\!\}_{s_K}$. This represents a permission specification $Perm$ that originates from a principal which is the owner of public key K. Therefore, a principal by signing the statement $\{\!|Perm, \{\!|Perm|\!\}_{s_K}|\!\}_{s_K}$ binds its arbitrary defined permission specification $Perm$ to the self signed permission $\{\!|Perm|\!\}_{s_K}$ to introduce a globally unique interpretation of $Perm$ in its name space denoted as $\langle P\ Perm\rangle$.

For example, an airline owner *Alice*, identified by her public key k_A, specifies permission for selling flights for her airline as *sell*, and delegates this permission to the brokers. On the other hand, *Dave*, identified by his public key k_D, the owner of another airline defines the same permission specification *sell* for selling flights for his airline by his brokers. Using the *localPermission* scheme provides a systematic way of globally and uniquely interpreting these two identical per-

mission specifications relative to their originators' name spaces without relying on a global name service. Table 3.1 depicts permission *sell* in two different name spaces of airline owners k_A and k_D with the globally unique interpretations.

Table 3.1: Permission *sell* and its definition in two different name spaces

Name Space	Permission Specification	Global Context	localPermission
k_A	*sell*	$\{\!\lvert sell \rvert\!\}_{s_{k_A}}$	$\langle k_A\ sell \rangle$
k_D	*sell*	$\{\!\lvert sell \rvert\!\}_{s_{k_D}}$	$\langle k_D\ sell \rangle$

The localPermission $\langle k_A\ sell \rangle$ clearly references the global unique context $\{\!\lvert sell \rvert\!\}_{s_{k_A}}$, therefore it is globally unique.

Similarly, the localPermission $\langle k_D\ sell \rangle$ clearly references the global unique context $\{\!\lvert sell \rvert\!\}_{s_{k_D}}$ which is different from the interpretation of $\langle k_A\ sell \rangle$.

localPermission Characteristics

We assume that a localPermission has the following characteristics in its definition.

Reference to a Globally Unique Context A permission's global unique context is the signed value of a permission specification by the originator of the permission. Each localPermission references a global unique context.

Local to a Name Space Although a localPermission has a globally unique interpretation, it also preserves its locality to its issuer's name space. localPermissions in different name spaces are distinct from each other even if they have the same specification. In the example described in Table 3.1, $\langle k_A\ sell \rangle$ is different from $\langle k_D\ sell \rangle$.

Associated with Their Originator The localPermissions are associated with their originator's either public key or local name. The localPermission of the form $\langle P\ Perm \rangle$ indicates that the principal P originated the permission $Perm$.

Globally Unique Interpretation Each localPermission references a global unique context (self signed permission specification), and therefore has a unique interpretation across all name spaces in open environments.

Long-Lived Data It is assumed that permissions are considered as long-lived data which do not need a specific validity period. When a principal originates a permission in its name space, it is considered that the permission specification is valid until the corresponding resource is unavailable. In other words, permission specifications can be defined for a long period of time and are always valid. Associating a validity period with permission certificates states that permission specifications are not considered to be valid for all time [77], which contradicts our assumption. This will be discussed in more detail in section 3.4.

Permission Certificate

There is also a permission certificate which binds a local specified permission in an originator's name space to a global context. The global context is the signed permission by the permission originator $\{|Perm|\}_{s_K}$ and represents a permission specification $Perm$ that originates from a principal owning public key K. Based on the assumption that a public key is considered to be globally unique, a permission signed by the key can be considered to have a globally unique permission interpretation and is assumed unambiguous. By signing $\{|Perm|\}_{s_K}$, its originator is the owner of key K, and has just one interpretation for $Perm$ in its name space. When a principal originates a new permission to allow access to its resources, that principal signs a self-signed certificate that binds the permission specification $Perm$ to the globally unique value $\{|Perm|\}_{s_K}$. The permission certificate is represented as the following:

$$\{|Perm, \{|Perm|\}_{s_K}|\}_{s_K}$$

Permission Global Ordering

localPermissions originate in some principal's name space, and therefore a principal must explicitly define how its locally specified permission relates to other permissions either in its name space or other name spaces. For example, assume *Alice* authorizes *Bob* to access all her resources and the global context for this permission is $\{|all|\}_{s_{k_A}}$. When *Bob* requests write permission for *Alice*'s documents, the trust management (policy) engine must infer that permission $\langle k_A\ all\rangle$ dominates permission $\langle k_A\ write\rangle$. Both permissions $\langle k_A\ all\rangle$, and $\langle k_A\ write\rangle$ are globally defined values as $\{|all|\}_{s_{k_A}}$, and $\{|write|\}_{s_{k_A}}$ respectively. There is implicit ordering relations between these two unique values. Thus, we define an explicit

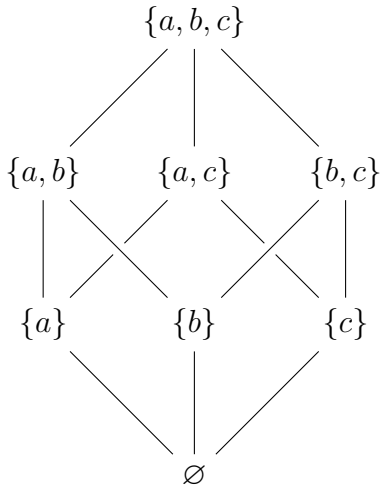
ordering relationship among localPermissions for further inferences to make access decisions. A binary relation is defined between localPermissions to introduce a meaningful ordering between them such that $X \rightsquigarrow Y$ represents permission Y is *"no less authoritative than"* permission X. Therefore, a principal that either issues or obtains permission Y is implicitly considered to hold permission X. A principal may issue permission certificates to define permission orderings that specify how a permission in its name space is related to permissions in other name spaces, where:

$$\{\!|Perm, X|\!\}_{s_K}$$

is a statement by principal P who is the holder of public key K stating that the permission $Perm$ in its name space (denoted as $\langle P\ Perm\rangle$) is *no less authoritative than* the permission X. The above permission certificate is denoted as:

$$X \rightsquigarrow \langle P\ Perm\rangle$$

It indicates that permission X is dominated by permission $\langle P\ Perm\rangle$. In other words, each principal that holds permission $\langle P\ Perm\rangle$ also holds permission X. Principal P must hold the permission X to define the global ordering relation with X relative to permission $Perm$ in its name space. In general, the set of all localPermissions S provides a lattice $(S, \sqsubseteq)$. A lattice is a partially ordered set in which every two elements X and Y have a least upper bound U and a greatest lower bound L. An upper bound U of S is said to be its least upper bound if $U \sqsubseteq W$ for each upper bound W of S. A set has zero or no more than one least upper bound. Dually, L is said to be a grater lower bound of S if $E \sqsubseteq L$ for each lower bound E of S. A set may have many lower bounds, or none at all, but can have at most one greatest lower bound. The following diagram depicts a set of permissions $\{a, b, c\}$, where the greatest lower bound of two elements $\{a, b\}$ and $\{a, c\}$ is $\{a\}$. Therefore, permissions $\{a, b\}$ and $\{a, c\}$ dominate permission $\{a\}$.



Definition *The greatest lower bound of localPermissions X and Y is denoted as:*

$$(X \sqcap Y)$$

$(X \sqcap Y)$ dominates any localPermission Z that is a lower bound of localPermissions X and Y. For example, localPermissions $\langle k_A\ \{read, write\}\rangle$ and $\langle k_A\ \{read, execute\}\rangle$ dominate localPermission $\langle k_A\ read\rangle$; that is:

$$\langle k_A\ read\rangle \rightsquigarrow (\langle k_A\ \{read, write\}\rangle \sqcap \langle k_A\ \{read, execute\}\rangle)$$

3.2.3 Delegation

Delegation refers to the act of a principal to propagate the permissions that either it originates or obtains from other principals or group of principals. This process can continue and form a chain of delegation. In delegation, the principal who propagates permissions to others is called a *delegator* and the principal who receives permissions is called a *delegatee*. A principal may be delegated a permission directly from another principal called *direct delegation*, or may be delegated a permission via a chain of direct delegations called *indirect delegation*. The example depicted in Figure 3.4 illustrates the delegation between *Alice* and *Bob*. It means that *Alice* allows *Bob* to propagate permission X to other principals. *Bob* then delegates permission X to *Dave*, and then *Dave* has been delegated X from *Alice* indirectly. The delegation between *Bob* and *Dave* is a direct delegation. The delegation between *Alice* and *Dave* is an indirect delegation.

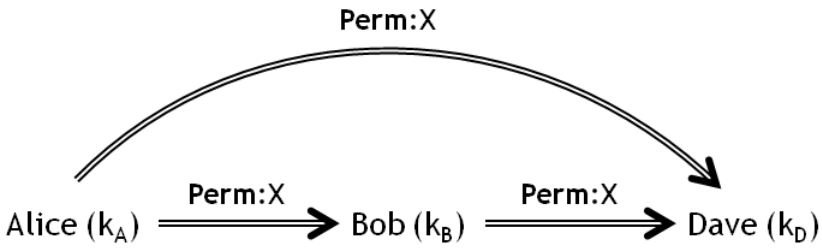


Figure 3.4: Direct and indirect delegation

Delegation Certificate

Delegation certificates are represented by the following syntax:

$$\{\![Q, X, D, V]\!\}_{s_K}$$

A principal P (owner of public key K) signs the certificate and delegates the authority for permission X (where X is a localPermission) to Q which is the subject of the delegation certificate. Q is a principal or group of principals that is identified by either its/their public key(s) or local name(s). For ease of exposition, we ignore the delegation bit D and validity period V. The delegation certificate is denoted as the statement $P \stackrel{X}{\Longrightarrow} Q$ and indicates that principal P delegates authority for permission X to principal Q. For example, the airline A, holder of public key k_A, delegates permission for selling flights ($\langle k_A\ sell \rangle$) to a group of its brokers ($(k_A\ Brokers)$). This delegation is accomplished by issuing the following delegation certificate:

$$\{\![(k_A\ Brokers), \langle k_A\ sell \rangle, d_1, v_1]\!\}_{s_{k_A}}$$

denoted as:

$$k_A \stackrel{\langle k_A\ sell \rangle}{\Longrightarrow} (k_A\ Brokers)$$

Note that, in SSAL, delegation of a permission does not imply that the recipient of the permission holds it,

First, it depends on whether the permission was originated from the principal that the permission specification was defined in that principal's name space. When a principal originates a permission specification, the permission specification occurs inside a permission certificate that is assumed to be in the certificate issuer's name space. A general localPermission is of the form $\langle K\ Perm \rangle$ where K is the issuer's

public key and $Perm$ is an arbitrary chosen permission specification. Principal P, the owner of public key K, originates permission specification $Perm$ in its name space and binds permission $Perm$ to the self signed permission $\{\!|Perm|\!\}_{s_K}$ as a reliable reference to the globally unique context. Then, it is implicitly inferred that P *holds* $\langle P\ Perm\rangle$, denoted as $P \ni \langle P\ Perm\rangle$. In the above example, the delegation of $\langle k_A\ sell\rangle$ implies that the $(k_A\ Brokers)$ holds it, since k_A is the originator of permission $\langle k_A\ sell\rangle$.

Second, the principal who is willing to delegate a permission must finally hold the permission in order to propagate it further. This prevents malicious principals delegating permissions that they do not hold, and consequently are not expected to delegate. For example, the malicious broker k_M may delegate permission $\langle k_A\ sell\rangle$ to $Clare$, holder of public key k_C. In the presence of the delegation statement $k_M \overset{\langle k_A\ sell\rangle}{\Longrightarrow} k_C$, k_C does not know whether $k_M \ni \langle k_A\ sell\rangle$ and mistakenly thinks that she holds the permission $\langle k_A\ sell\rangle$.

3.2.4 Accountability

Accountability refers to the obligation of a principal to be responsible for its activities based on the permission that is delegated to it. Delegation subterfuge is possible when one cannot precisely specify how a permission is held by a principal. It leads to breakdown in tracking accountability. A principal who is concerned about subterfuge checks whether other earlier principals in the chain are responsible for the permissions they delegate. When a principal originates a permission it is implicitly held accountable for that permission, denoted as $P \rhd \langle P\ Perm\rangle$. In general, a principal is considered to be accountable for a permission if it accepts responsibility for the result of the actions done by other principals using the delegated permission. For example, in issuing permission $\langle k_A\ sell\rangle$ for airline A, the principal k_A is considered to be accountable for the use of permission $\langle k_A\ sell\rangle$ by any principal that holds this permission. Note that, holding a permission by a principal may not result in that principal's accountability for that permission. In other words, a principal may not be considered to be accountable for a permission that it holds, unless it clearly states that it *accepts accountability* for that permission. This prevents malicious principals deceitfully delegating a permission to other principals to make them accountable by holding a permission. On the other hand, a principal must hold the permission to assert accountability. This ensures that a non-trusted malicious principal cannot assert accountability by issuing an *accepts accountability* statement for a permission for which they are not trusted

to hold. A principal who is accepting accountability for the permission issued by a group of principals implicitly accepts accountability for the similar permission specification referring to the name space of each member of the group. For example, if *Emily* is a member of group $(k_A\ Brokers)$ and accepts accountability for permission that is originated by $(k_A\ Brokers)$ group as $\langle (k_A\ Brokers)\ sell \rangle$, she implicitly asserts being accountable for $\langle k_E\ sell \rangle$. Therefore, in delegating $\langle k_E\ sell \rangle$ to *Bob* for further delegation, *Emily* accepts accountability for how $\langle k_E\ sell \rangle$ is handled by *Bob*.

Moreover, permission ordering relations may not result in accountability, where one cannot infer accountability for any permission in one name space dominated by another permission originated in a different name space. If this were permitted then *Bob* may not be aware of this and specifies his own permission $\langle k_B\ sell \rangle$ and asserts $\langle k_M\ sell \rangle \rightsquigarrow \langle k_B\ sell \rangle$. Then since *Bob* is by default accountable for all the permissions he originates then he would be inferred as accountable for the $\langle k_M\ sell \rangle$ permission which results in subterfuge for k_M.

3.3 Formal Foundation for SSAL

In this section we present a logic for the SSAL language which provides subterfuge-freedom in delegation of permissions in open environments. When a set of statements and certificates is defined, and an authorization request is formulated, the logic rules can be used to check whether the request is valid as a consequence of the existing SSAL statements. The SSAL logic uses the following notations and formulae where P, Q, R represent principals; X, Y, Z represent localPermissions; N is an arbitrary chosen name for a principal; $Perm$ is an arbitrary chosen specification of a permission, and A, B are formulae:

- $P \ni X$: principal P *holds* permission X.
- $P \longrightarrow Q$: principal Q *speaks for* principal P.
- $P \overset{X}{\Longrightarrow} Q$: principal P *delegates* permission X to principal Q.
- $P \rhd X$: principal P is *accountable* for permission X in the delegation chain.
- $X \rightsquigarrow Y$: localPermission Y is *no less authoritative than* localPermission X.
- $(P\ N)$: local name N in the name space of principal P.

- $\langle P\ Perm \rangle$: locally defined permission $Perm$ in the name space of principal P.
- $A \Rightarrow B$: the statement A implies statement B.

Note that, by mentioning principal k_A we mean that there is a principal A that is the owner of public key k_A. The main focus of the SSAL logic is a set of new axioms, expressed as rewrite/inference rules that extends the SPKI/SDSI logic by incorporating localPermissions and demonstration of how localPermission names can avoid subterfuge in delegation of permissions. The idea behind the logic is to allow a principal to make a subterfuge-safe decision, whether it is safe to delegate a permission based on a collection of statements, and also whether there is any principal accountable for the action performed based on that permission. The SSAL logic is comprised of 21 axioms including 6 axioms (N1, N2, N3, D1, D2, and D4) that come directly from SPKI/SDSI.

3.3.1 Principal Names Relation

Each name certificate denotes a *speaks for* relation between the subject of a certificate and the arbitrary chosen name in the certificate issuer's name space.

N1

Given principal (local name or public key) P, name N, and public key K, the following rule will be interpreted as *speaks for* relation, that P *speaks for* $(K\ N)$:

$$\frac{\{\!|N, P|\!\}_{s_K}}{(K\ N) \longrightarrow P}$$

N2

This reduction combines two linked *speaks for* statements. Given local names (or public keys) P, Q, R and an arbitrary chosen name N then:

$$\frac{(Q\ N) \longrightarrow P; R \longrightarrow Q}{(R\ N) \longrightarrow P}$$

This rule indicates that principal Q may define an arbitrary name N for principal P in its (Q's) name space; if principal Q *speaks for* principal R, then R also refers to principal P in its (R's) name space with the same name identifier N.

N3

Given principals P, Q and R the *speaks for* relation is transitive, that is:

$$\frac{P \longrightarrow Q; Q \longrightarrow R}{P \longrightarrow R}$$

3.3.2 Permission Delegation

D1

The following rule provides an interpretation for delegation.

$$\frac{\{\!|P, X, D, V|\!\}_{s_K}}{K \stackrel{X}{\Longrightarrow} P}$$

D2

Given principals P, Q, R, S and permission X then the direct delegation reduction rule is:

$$\frac{P \stackrel{X}{\Longrightarrow} Q; Q \longrightarrow R}{P \stackrel{X}{\Longrightarrow} R}$$

This rule indicates that the permission X which is delegated to principal Q is also delegated to any principal (R) that *speaks for* the delegatee.

D3

In direct delegation of permissions, if principal P delegates a permission Y to principal Q, principal P implicitly delegates any permission X dominated by Y.

$$\frac{P \stackrel{Y}{\Longrightarrow} Q; X \rightsquigarrow Y}{P \stackrel{X}{\Longrightarrow} Q}$$

D4

The permission that a principal is delegated via indirect delegation is the intersection of all permissions delegated in the chain.

$$\frac{P \overset{X}{\Longrightarrow} Q; Q \overset{Y}{\Longrightarrow} R;}{P \overset{X \sqcap Y}{\Longrightarrow} R}$$

3.3.3 Permission Holding

H1

Given a public key K and arbitrary chosen specification N for permission, we define the following Holding rule, that the owner of public key K holds localPermission $\langle K\ N \rangle$:

$$\frac{\{\!|N, \{\!|N|\!\}_{sK}|\!\}_{sK}}{K \ni \langle K\ N \rangle}$$

H2

Q is authorized for X, if P holds X in the first place and delegates it to Q:

$$\frac{P \ni X, P \overset{X}{\Longrightarrow} Q}{Q \ni X}$$

H3

A member of a group holds any permission that is held by the whole group:

$$\frac{P \ni X, P \longrightarrow Q}{Q \ni X}$$

Note that the group of principals does not hold the permissions that each of its members holds in their own name spaces.

H4

A principal holding permission Y, holds all permissions, X, dominated by Y:

$$\frac{P \ni Y; X \rightsquigarrow Y}{P \ni X}$$

3.3.4 Permission Ordering

P1

The following inference rule is the ordering interpretation for permission certificates. Given public key K, permission X and permission specification $Perm$ then $\langle K\ Perm\rangle$ is *no less authoritative than* X:

$$\frac{\{\!|Perm, X|\!\}_{s_K}, K \ni X}{X \rightsquigarrow \langle K\ Perm\rangle}$$

P2

The ordering relationship $\rightsquigarrow$ between permissions is, by definition, reflexive:

$$\frac{P \ni X}{X \rightsquigarrow X}$$

P3

If a localPermission X dominates locally defined permission $Perm$ in the name space of principal P, it implicitly dominates the similar permission specification $Perm$ in the name space of all principals that have *speak for* relation with P:

$$\frac{\langle P\ Perm\rangle \rightsquigarrow X; P \longrightarrow Q}{\langle Q\ Perm\rangle \rightsquigarrow X}$$

P4

localPermission reduction is defined by the following rule, whereby, given principals P and Q, localPermissions X and Y, and an arbitrary chosen permission specification $Perm$, then:

$$\frac{X \rightsquigarrow \langle P\ Perm\rangle; P \longrightarrow Q; \langle Q\ Perm\rangle \rightsquigarrow Y; Q \rhd \langle P\ Perm\rangle}{X \rightsquigarrow Y}$$

This rule indicates that if a permission $Perm$ originated in the name space of principal P, and dominates permission X, and there exists the same permission specification $Perm$ in the name space of Q; where Q *speaks for* P, then Q must explicitly provide an accountability for permission $\langle P\ Perm\rangle$ so that any permission Y that dominates $\langle Q\ Perm\rangle$ also dominates X.

P5

A permission that is a lower bound of permission X and Y is dominated by the greatest lower bound of X and Y:

$$\frac{Z \rightsquigarrow X; Z \rightsquigarrow Y}{Z \rightsquigarrow (X \sqcap Y)}$$

3.3.5 Accountability for Permissions

A1

The owner of public key K that originates the permission $\langle K\ Perm \rangle$ is considered to be accountable for any actions permitted by that permission. We have:

$$\frac{K \ni \langle K\ Perm \rangle}{K \rhd \langle K\ Perm \rangle}$$

A2

A principal K that holds a permission may accept accountability for a valid permission X by signing a statement indicating acceptance of accountability. The following rule denotes this:

$$\frac{\{\!|\mathsf{accept_accountability}(X)|\!\}_{s_K}; K \ni X}{K \rhd X}$$

A3

A principal P is considered to be accountable for a permission X if a principal Q that is accountable for that permission X *speaks for* P:

$$\frac{Q \rhd X; P \longrightarrow Q}{P \rhd X}$$

A4

For a principal referenced within a localPermission, the following accountability may be deduced, that is:

$$\frac{R \rhd \langle P\ Perm\rangle; P \longrightarrow Q}{R \rhd \langle Q\ Perm\rangle}$$

A5

A principal that is accountable for a permission is considered to hold the permission, that is:

$$\frac{R \rhd X}{R \ni X}$$

3.3.6 Logical Properties

The following properties are derived from the previous axioms. Their reasonableness provides confidence in the correctness of the SSAL logic. The formulae $A \Rightarrow B$ indicates that the statement A implies statement B.

Property 1

The following property can be inferred from combining rule P5, D3, and the SPKI-delegation rule D4 in section 3.3.2. This allows a principal to reason about indirect delegation based on a collection of delegation statements and localPermission relations. Given principals P, Q, R and permissions X, Y, Z then we infer:

$$((P \stackrel{X}{\Longrightarrow} Q) \wedge (Q \stackrel{Y}{\Longrightarrow} R) \wedge (Z \rightsquigarrow X) \wedge (Z \rightsquigarrow Y)) \ \Rightarrow (P \stackrel{Z}{\Longrightarrow} R) \qquad (3.1)$$

Property 2

This follows from Holding rule H2 and Permission Ordering rule P3 that, in delegation of a localPermission Y by a principal that holds it, the recipient also holds any localPermissions dominated by Y:

$$((P \ni X) \wedge (P \stackrel{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \qquad (3.2)$$

Property 3

Any principal that *speaks for* the recipient of a localPermission also holds all dominated localPermissions. It follows from combining Permission Delegation rule P4, and Holding Permission rule H3:

$$((P \ni X) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y) \wedge (Q \longrightarrow R)) \Rightarrow (R \ni X) \tag{3.3}$$

Property 4

If a principal is delegated some permissions and the delegator holds any dominated permissions, that delegatee also holds the dominated permissions.

$$((P \ni X) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X)$$

Property 5

If a principal P originates a permission $Perm$ in its name space, then it dominates the same permission name $Perm$ in the name space of all principals that have *speaks for* relation with P.

$$((P \ni \langle P\ Perm \rangle) \wedge (P \longrightarrow Q)) \Rightarrow (\langle Q\ Perm \rangle \rightsquigarrow \langle P\ Perm \rangle) \tag{3.4}$$

Property 6

If we consider well-defined (held by principals) localPermissions X, Y and Z then by reflexivity of *speaks for* and *no less authoritative than*, it follows that permission ordering is transitive in the sense that:

$$((X \rightsquigarrow Y) \wedge (Y \rightsquigarrow Z)) \Rightarrow (X \rightsquigarrow Z) \tag{3.5}$$

Proposition (3.5) can be considered in the context of a conventional trust management system whereby some *"super security authority"* effectively asserts a global pre-order over permissions $(PERM, \sqsubseteq)$ and thus the set $(PERM, \rightsquigarrow)$ forms a pre-order.

Property 7

If a principal Q accepts accountability for permission $Perm$ in the name space of a principal P and principal Q also *speaks for* P, any permission, X, that dominates permission $Perm$ referring to its name space, also dominates the permission $Perm$ referred to by principal P that is:

$$((\langle Q\ Perm\rangle \rightsquigarrow X) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle P\ Perm\rangle)) \Rightarrow (\langle P\ Perm\rangle \rightsquigarrow X) \quad (3.6)$$

Property 8

If a principal Q accepts accountability for the permission $Perm$ in the name space of principal P, and Q *speaks for* P, any permission that is dominated by permission $Perm$ in P's name space, is also dominated by the same permission $Perm$ in the name space of Q.

$$((X \rightsquigarrow \langle P\ Perm\rangle) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle Q\ Perm\rangle)) \Rightarrow (X \rightsquigarrow \langle Q\ Perm\rangle) \quad (3.7)$$

Property 9

By reflexivity $\langle P\ Perm\rangle \rightsquigarrow \langle P\ Perm\rangle$ on well-defined localPermissions, it follows from proposition (3.6) that if a principal Q *speaks for* P and is accountable for permission $Perm$ referred to within principal P's name space, then the permission $Perm$ referred to by principal Q dominates permission $Perm$ originated by P:

$$((P \longrightarrow Q) \wedge (Q \rhd \langle P\ Perm\rangle)) \Rightarrow (\langle P\ Perm\rangle \rightsquigarrow \langle Q\ Perm\rangle) \quad (3.8)$$

3.4 Threats and Mitigation

Here, we describe the threats that can affect the global unique interpretation of a localPermission. A localPermission is an interpretation of a permission in a specific principal's name space. A name space is identified by a principal's public key; therefore, any threat on that principal's public key may affect the localPermission's global unique interpretation. A principal's public key may become compromised, changed, or expired and the localPermission interpretation for that public key may no longer be valid. In this section, we review these threats and propose a solution to mitigate the threats.

3.4.1 Key Expiration Threat

Each public key has an expiration date indicating the public key is invalid and should not be trusted after the expiration date. When a principal's public key has expired, any localPermission interpretation that refers to that key will no longer be valid. Assuming airline A's public key is k_A, after originating localPermission $\langle k_A\ sell \rangle$ and delegating to Bob expires, then, when k_A expires, the global unique interpretation for $\langle k_A\ sell \rangle$ is invalid.

3.4.2 Key Refreshing Threat

A public key's expiration date can be extended. When a key holder finds out that its public key (consequently private key) is expiring, they request the key provider to extend the expiration date and retrieve the updated public key (actually key pair). Thus, the localPermission interpretation that refers to that key would not be changed. However, the originator of a permission needs to issue the permission certificate with the public key that has a new expiration date.

3.4.3 Key Change Threat

A principal may change its public key (key pair); therefore, the localPermission and its interpretation that is bound to that key would be affected. The localPermission of the form $\langle K\ Perm \rangle$ emphasizes that permission $Perm$ originates from a principal with public key K and belongs locally to the name space of holder of

key K. If any change is applied to the originator's public key, then, the localPermission interpretation needs to be redefined with the new public key. Consider the scenario that airline A changes its public key from k_A to $k_{A'}$. Assuming that it already has originated permission *sell* in its name space as $\langle k_A\ sell \rangle$, after changing its public key, it has to sign all the existing permission certificates with its new private key that corresponds to the new public key.

3.4.4 Key Compromised Threat

The use of a private key (corresponding to a public key) that an attacker has stolen to sign the permission on behalf of the resource owner leads to referring to that key pair as compromised [78]. Although obtaining a private key is a difficult process for an attacker, it is possible. After an attacker obtains a private key, that key pair is compromised. An attacker can use the stolen key to originate or modify permissions and pretend that the permissions are from the legitimate resource owner without the knowledge of the resource owner. Based on our assumptions, permissions in our model are globally unique and long-lived data; therefore, they do not have a validity period or at least are considered to have a long term validity period. Thus, a malicious principal such as *Eve* with the public key k_E has enough time to obtain the compromised key and originate a new permission, say $\langle k_A\ all \rangle$ by using the compromised key k_A and pretending that this permission is from airline A.

3.4.5 Threat Mitigation Techniques

In this section, we discuss the solutions for threat mitigation and finally introduce a reliable scheme for preventing the existing threats on localPermissions.

Setting Expiration Date for Permissions

Despite the characteristics in the definition of localPermission, we assume a validity period for a localPermission when a principal signs a permission certificate. This validity period is independent of the validity period of the public key of the originator of the permission, and also independent of the validity period of the delegation certificate. The permission certificate with validity period is repre-

sented as:

$$\{\!|Perm, \{\!|Perm|\!\}_{s_K}, V|\!\}_{s_K}$$

where $Perm$ is a permission specification, V is a validity period of the localPermission that references the permission value $\{\!|Perm|\!\}_{s_K}$, and K is a public key of the principal who originates the permission. In this scheme, when the validity period ends, the localPermission interpretation is not valid any more. In addition, a principal can issue a permission certificate, within the validity period V_1, that dominates the other localPermission within the validity period V_2, if V_2 contains V_1. For example, k_A may issue a localPermission $\langle k_A\ sell\rangle$ within validity period $V_1 = (02/06/2014, 04/06/2014)$, and define its global ordering relation to permission $sell$ in k_B's name space, $\langle k_B\ sell\rangle$, within the validity period of $V_2 = (01/06/2014, 05/06/2014)$, as $\langle k_B\ sell\rangle \rightsquigarrow \langle k_A\ sell\rangle$). Validity period V_2 contains the validity period V_1 and indicates a principal must issue a permission ordering certificate to any other permissions in the interval of validity period of the dominated permission. Note that, if $V_2 \leq V_1$, the permission ordering cannot be effective after the time that the dominated permission expires. Moreover, a delegation certificate that contains a localPermission is not effective outside of the interval that the localPermission is valid. In the delegation of a permission there is the complication of dealing with two levels of intervals, the interval at which a delegation certificate is valid, and the interval at which a permission is alive. A principal can issue a delegation certificate with validity period not greater than the permission certificate validity period. For example, if we have V_1 validity period for the delegation certificate, and validity period V_2 for the permission certificate, the effective interval that the delegation certificate is valid is $V_1 \sqcap V_2$. The above shows how validity periods for permission certificates can be incorporated to the model. However, later discussions show the use of an ephemeral key and a separated key for signing permission certificates makes it non essential to use a validity period for permission certificates. Therefore, we assume localPermissions to be long lived data without validity period.

Using an Ephemeral Key for Signing Permission Certificates

The signed permission as a global reference for a localPermission can be decrypted by the holder of a public key to verify that the permission is originated by the owner of the private key. Using a different key pair for signing permissions (called permission key), rather than the permission originator's identity public/private key, will prevent the threats of expiration, refreshing, or changing the identity

key. However, even by using a different key for signing the permissions rather than the permission originator's identity key, there might exist enough time for a malicious principal to run a compromised key attack for a permission key. To prevent a compromised key attack, we propose using ephemeral keys [79–81] to handle the security of localPermissions over a long period of time. An ephemeral key is a private or public key that is unique for each signing scheme. An ephemeral private key is to be destroyed as soon as computational need for it is completed. It enables a principal to sign a permission in a way that ensures the private key cannot be re-used after a finite period of time, and eventually prevents the compromised key attack. Before the expiration time, the principal that aims to encrypt the permission specification uses the ephemeral private key for signing the permission and binds the permission global unique context to the locally defined permission. The provider of an ephemeral key destroys the ephemeral private key to prevent key recovery after a short time. In this way, an attacker cannot obtain the principal's ephemeral private key, since that is valid only for a short period of time. A principal that originates the localPermission references its permission key to its identity key by issuing a *speaks for* relation between the permission key and identity key. Consider k_e as permission key of the principal that is the owner of the public key k, we have the following inference by rule N1 in SSAL (Note that in the name certificate the name can be null):

$$(\{\!|null, k|\!\}_{s_{k_e}}) \Rightarrow (k_e \longrightarrow k) \ (1)$$

A principal with public key k, originates permission $Perm$ in its name space and issues the permission certificate with the permission key k_e, that is:

$$(\{\!|Perm, \{\!|Perm|\!\}_{s_{k_e}}|\!\}_{s_{k_e}}) \Rightarrow (k_e \ni \langle k_e \ Perm\rangle) \ (2)$$

consequently by rule H3, statements (1), and (2) we deduce that $k \ni \langle k_e \ Perm\rangle$, that is:

$$((k_e \ni \langle k_e \ Perm\rangle) \wedge (k_e \longrightarrow k)) \Rightarrow (k \ni \langle k_e \ Perm\rangle)$$

3.5 Certificate Chain Discovery

In distributed and open environments when a principal requests access to a resource, in order to prove to the resource owner that the requester is authorized, the requester must present to the resource owner a set of certificates relevant to its request. The requester may not know what set of certificates it should present to the resource owner to be granted access to that resource, or even may not know about the existence of such a set of certificates. It is assumed all relevant certificates to a request are stored with the requester. If the requester maintains a database of a small number of certificates, it would be straightforward to extract the certificates related to the request and send them to the resource owner. However, if stored by distributed principals and each principal maintains a large number of certificates, it is not obvious which set of certificates the requester must present to the resource owner to prove authorization for accessing a particular resource. Therefore, an automated process is required for discovering and generating the sequences, if a valid chain of certificates exists. The process of discovering the set of certificates that proves authorization for a resource is called *certificate chain discovery*. In a centralized environment, a certificate chain discovery algorithm addresses how to discover a set of certificates relevant to a request from a large number of certificates that are stored in a centralized manner. However, in decentralized environments certificates are issued by distributed principals. In order to discover a set of certificates related to a request, a requester or even a resource owner does not have all certificates in one location. In this section, we present a certificate chain discovery algorithm for SSAL that discovers the full set of certificates related to a request when certificates are stored with distributed principals.

3.5.1 Related Work

Certificate Chain Discovery in SPKI/SDSI

SSAL builds on some of the features of SPKI/SDSI. However, SPKI/SDSI does not contain the notion of *"localPermission"* and specifies permissions in tags (s-expressions). Several certificate chain discovery algorithms have been proposed for SPKI/SDSI. Most of these [59, 82] assume that all of the potential certificates related to a request are centralized in one place, and they do not address how these certificates are gathered. This assumption is unusual for trust management

systems, since they establish and manage trust and authorization relationships between decentralized principals. Moreover, SPKI/SDSI uses an ACL (Access Control List) that controls access to the resource belonging to the principal which holds the ACL. A certificate chain discovery algorithm for SPKI/SDSI was proposed in [59, 83]. In this approach the set of certificates are stored centrally, so the proof of authorization is constructed in a centralized manner. Another certificate chain discovery algorithm based on the theory of push-down automata was proposed in [82] for SPKI/SDSI. However this algorithm also assumes that certificates are stored centrally and proof of authorization requires finding a certificate chain among a set of certificates that are stored centrally.

Certificate Chain Discovery in PolicyMaker and Keynote

While PolicyMaker and Keynote [14,47] support trust relationships in distributed environments, they do not address mechanisms for certificate chain discovery. They include a mechanism for checking the existence of a particular trust relationship called a *compliance checker*. The compliance checker service determines how an operation requested by a principal should be handled, given a policy and a set of certificates.

Certificate Chain Discovery in RT

Li et al. [84] proposed a certificate chain discovery algorithm for Role-based Trust management (RT) that discovers the certificate chain when the storage of certificates is either centralized or decentralized. The scheme uses three algorithms: forward search algorithm, backward search algorithm, and bidirectional search algorithm. The bidirectional search algorithm is an alternative for both forward and backward search algorithms when some chains cannot be found by neither forward nor backward search algorithms individually. Although the certificate chain discovery algorithm for RT addresses certificate discovery from distributed principals, the proof of authorization is constructed in a centralized manner [84]. In our approach, certificate storage is distributed and various principals summarize their part of the proof of authorization before sending it to other principals. In addition, our approach uses a simple and effective algorithm to discover certificates without requiring centralized data storage such as the ACL in SPKI/SDSI.

3.5.2 Certificate Chain Discovery for SSTM

Certificate Storage Strategy

In our model, distributed storage is provided by the repositories that are running on different principals' servers. When a principal issues a certificate, it stores that certificate in its own repository called a "local cert". The issuer publishes its certificate in another principal's repository, a so called "copy cert". The purpose of publishing the certificates to another principal's repository is that those certificates can be distributively located and used by other principals. Each repository supports insertion and storage of name, permission, and delegation certificates to perform authorization queries and discovery of proof of authorization. Certificates, after being issued, are inserted in the issuer's local repository. Copies of these certificates are published to the repository of the subject of the certificates. Consider airline A is a resource owner and delegates the permission $\langle k_A\ sell\rangle$ to a group of its brokers denoted as $(k_A\ Brokers)$. Bob is a member of this group. The airline A issues the following delegation, name, and permission certificates:

$$C1 : k_A \stackrel{\langle k_A\ all\rangle}{\Longrightarrow} (k_A\ Brokers)$$

$$C2 : (k_A\ Brokers) \longrightarrow k_B$$

$$C3 : \langle k_A\ sell\rangle \rightsquigarrow \langle k_A\ all\rangle$$

Airline A stores these certificates in its local repository. Copies of these certificates are also stored with Bob, the owner of public key k_B. Bob originates a new permission in his name space as $\langle k_B\ all\rangle$ and states that this permission dominates the permission that he received from airline A. He then delegates the permission $\langle k_B\ all\rangle$ to $Dave$. These assertions are defined by issuing the following certificates:

$$C4 : \langle k_A\ all\rangle \rightsquigarrow \langle k_B\ all\rangle$$

$$C5 : k_B \stackrel{\langle k_B\ all\rangle}{\Longrightarrow} k_D$$

When $Dave$ requests authorization for sell at airline A, he has a copy of certificates $C4$ and $C5$ (obtained from Bob and denoted as $C4'$ and $C5'$) in his local repository. $Dave$ contacts Bob's repository to collect all the certificates related to permission $\langle k_A\ sell\rangle$, where Bob is the subject of those certificates. Therefore, $C1'$, $C2'$, and $C3'$ will be found. $Dave$ presents all those certificates to Airline A. Hence, the set of certificates from different locations are successively discovered.

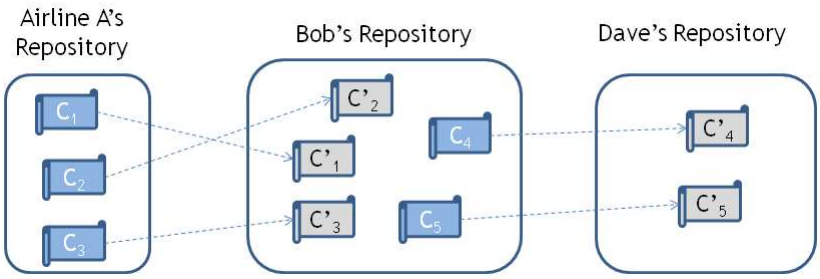


Figure 3.5: "local cert" and "copy cert" in local repositories.

The requester collects all related certificates from different locations to discover a delegation chain from the resource owner (airline A) to the requester ($Dave$) if one exists. Figure 3.5 depicts the local repositories that contain the certificates which support the trust relationship between airline A, Bob, and $Dave$.

Algorithm

The certificate chain discovery algorithm is used by the requester to discover a set of certificates that provide a proof of the requester's authorization. The following protocol demonstrates the requesting process by requester A for an action a on a resource r where principal B is the owner of resource r. The notation $A \twoheadrightarrow B : \{a, b, c, ...\}_{s_A}$ indicates that the entity A sends a message to entity B. The message $\{a, b, c, ...\}$ is signed by the sender (s_A).

$$Msg1 : A \twoheadrightarrow B : \{(a, r)\}_{s_{k_A}}$$

$$Msg2 : A \twoheadrightarrow B : \{S\}_{s_{k_A}}$$

$$Msg3 : B \twoheadrightarrow A : Authorization\ Decision$$

1. The requester sends a request to the resource owner to access resource r for action a.
2. The requester sends a set of certificates S related to the permission $\langle k_B\ (a, r)\rangle$.
3. If the resource owner verifies the proof of authorization, it grants the requester access to the requested resources.

The certificate chain discovery algorithm is used for step 2. The algorithm takes the requester's public key k_A, permission related to the requested action and resource $\langle k_B\ (a, r)\rangle$, a set of certificates, and current time as input, and returns

a certificate chain if one exists. The certificate chain provides proof that the requester (public key k_A) is authorized to perform the action a on resource r at the time specified in the time interval I. If the time I is not specified in a certificate it is assumed to be valid forever ($-\infty$ to $+\infty$). The set of certificates in S includes all type of SSAL certificates, i.e. name certificates, permission certificates, and delegation certificates.

The algorithm first excludes irrelevant certificates. Irrelevant certificates are certificates that are useless in deriving the proof of authorization. The following are the certificates that need to be removed before the discovery process:

1. The certificates that either their signatures cannot be verified or are outside of the valid time period.
2. Delegation certificates that do not include the permission that was requested in the request.
3. Permission certificates which are not a superset of the requested permission.

Before the certificate chain discovery process, the permissions that are not a superset of the requested permission are removed. For example, in the case that there is the following permission ordering:

$$\langle k_A\ read\rangle \rightsquigarrow \langle k_A\ write\rangle \rightsquigarrow \langle k_A\ all\rangle$$

when a principal requests *write* on k_A's resources, the permission certificate for $\langle k_A\ read\rangle$ will be removed before the certificate chain discovery process starts. This is because the algorithm first checks that the requested permission is a subset of the permissions in each of the delegation certificates. If there is a delegation certificate whose permission is not a superset of the requested permission, the certificate sequence is invalid, and is useless in providing a proof of authorization. These certificates are useless in trying to derive the desired set of certificates. The certificate chain discovery algorithm checks if there is a chain of delegation certificates from resource owner to the requester. The complexity of discovering a chain of delegation certificates depends on the subject of the delegation certificate. We explain the discovery process based on the subject of delegation certificates: public keys, local names, and extended local names.

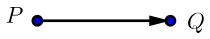
Subjects as Public Keys The subject of a delegation certificate can be specified by its public key. The issuer of the delegation certificate is the resource

owner who originates the permission for access to its resources. The delegation certificate grants the permission to the subject for the resources in the name space of the issuer of the certificate. Considering P and Q as public keys, $\langle P\ Perm \rangle$ denotes the permission specified as $Perm$ in the name space of P. The following denotes a delegation certificate where P grants permission $Perm$ in its name space to Q

$$P \stackrel{\langle P\ Perm \rangle}{\Longrightarrow} Q$$

Assuming all the delegation certificates are as above and all the useless certificates are removed before the process, the algorithm proceeds for the scenario with principals as public keys as follows:

1. A directed graph is set up where each public key is represented by a vertex and the delegation relation between public keys is represented by an edge between the associated vertices. The following is a directed graph with two vertices and one edge representing the delegation certificate:



2. A Depth-First Search (DFS) algorithm [85] is used to determine if a path exists from vertex k_O (the public key of the resource owner) to vertex k_R (public key of the requester).

3. If the path exists the path is returned, and if there is not a path, it terminates with failure.

Subjects as Local Names In this scenario, we assume that the subjects of the delegation certificates are local names. To proceed with certificate chain discovery the local names must be reduced to their public key values. Given public keys P and R, local name Q, and permission $\langle P\ Perm \rangle$, the following rule (1) reduces local names to their public key values:

$$\frac{P \stackrel{\langle P\ Perm \rangle}{\Longrightarrow} Q; Q \longrightarrow R}{P \stackrel{\langle P\ Perm \rangle}{\Longrightarrow} R}$$

The permission $Perm$ originating in the name space of the issuer of the delegation certificate P is delegated to a principal identified by its local name Q. The result of this computation is a set of delegation certificates with the subjects as public keys only. Recalling the scenario with only public keys, we can now run the

algorithm described for that scenario over the remaining delegation statements.

Subject as Extended Name The subject of a delegation certificate can be an extended name. In most applications the use of extended names is not needed. However, for those applications that need extra expressiveness for naming the principals, extended local names can be used. For instance, *Insight* is a research centre in University College Cork (UCC), and UCC is a member of the National University of Ireland (NUI), then (k_{NUI} *UCC Insight*) is an extended name that identifies the unique name *Insight* in the name space of the principal that is identified by its local name (k_{NUI} *UCC*). Extended names can be reduced to local names and further reduced to public keys using the following reduction rule. The following rule (2) states that an extended local name ((*Q N*) *M*) can be reduced to the local name (*R M*). Given local names (or public keys) Q, R, and public key P and arbitrary chosen names N and M then:

$$\frac{((Q\ N)\ M) \longrightarrow P; R \longrightarrow (Q\ N)}{(R\ M) \longrightarrow P}$$

Principal (*Q N*) may define an arbitrary name *M* for principal *P* in its name space; if principal (*Q N*) speaks for principal *R*, then (*R M*) has the same public key value. For example, the delegation certificate denoted $k_1 \stackrel{X}{\Longrightarrow} (k_2\ N_1\ N_2)$, and the name certificates denoted as: $(k_2\ N_1) \longrightarrow k_3$ and $(k_3\ N_2) \longrightarrow k_4$ can be reduced to the delegation statement $k_1 \stackrel{X}{\Longrightarrow} k_4$. The result of this computation is a set of delegation certificates with the public key as subject. Recalling the scenario with only public keys, the algorithm described for that scenario can now be run over the remaining delegation statements.

Distributed Chain Discovery

In this section, we demonstrate the certificate chain discovery through an example. Recall that the issuer of each certificate stores the original certificate in its local repository and sends a copy of that certificate to the repository of the subject of the certificate. The distributed certificate chain discovery process starts from the requester. The requester may hold a copy of the delegation certificate that is related to the permission that it (the requester) needs for its request. Therefore, the chain discovery algorithm looks up the local repository of the issuer of the certificate to check if the requester holds a copy of that certificate. In the following, we explain the procedure that will be applied to the local repository of the

principals:

1. Remove useless delegation certificates.
2. Convert the remaining name, permission, and delegation certificates to name, permission, and delegation statements, respectively.
3. Compute the reduction rules (1) and (2) on these statements. This computation results in the subjects of all delegation statements being converted to their public key values.
4. Extract all statements of the form ${k_i}^T \Longrightarrow k_j$.
5. Form a directed graph, where the vertices are ${k_i}^T$ and k_j, and the edges are the delegation relations between k_i and k_j (k_i and k_j are public keys).
6. Use a DFS to determine if there is a path from resource owner k_o to the requester k_r.
7. Output the desired set of certificates if there is a path, otherwise terminate with failure.

This procedure is explained in detail in the following example. Suppose that company A, identified by public key k_A, sets up a group of brokers, (k_A $flightBrokers$), to sell flights on its behalf, by issuing the name certificate c_1, and sets up a group (k_A $hotelBrokers$), to book its hotel rooms, by issuing the name certificate c_2. A allows its brokers to delegate further their permission for selling flights and booking hotels. The other principals in this scenario are k_B, k_C, k_D, k_F, k_S, k_T and (k_T $employees$). Note that k_S is not only a hotel broker for A, but also is an employee of another company which is identified by public key k_T. A also issues the following delegation certificates c_3, c_4, and c_5. Certificates c_6, c_7, and c_8 are further delegations of the permissions that are originated by k_A. Certificate c_9 defines k_S is a member of the group of principals,

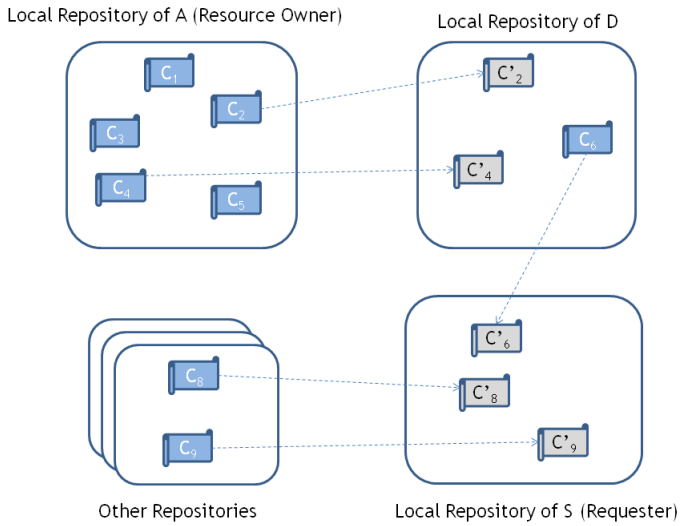


Figure 3.6: Initialization of certificates in distributed repositories

$(k_T\ employees)$.

$$c_1 : (k_A\ flightBrokers) \longrightarrow k_C$$

$$c_2 : (k_A\ hotelBrokers) \longrightarrow k_D$$

$$c_3 : {k_A}^T \stackrel{\langle k_A\ sell\rangle}{\Longrightarrow} (k_A\ flightBrokers)^T$$

$$c_4 : {k_A}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} (k_A\ hotelBrokers)^T$$

$$c_5 : {k_A}^T \stackrel{\langle k_A\ sell\rangle}{\Longrightarrow} {k_B}^T$$

$$c_6 : {k_D}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} {k_S}^F$$

$$c_7 : {k_C}^T \stackrel{\langle k_A\ sell\rangle}{\Longrightarrow} {k_F}^T$$

$$c_8 : {k_F}^T \stackrel{\langle k_A\ sell\rangle}{\Longrightarrow} {k_S}^F$$

$$c_9 : (k_T\ employee) \longrightarrow k_S$$

Figure 3.6 depicts the initialization of each certificate in each local repository. Assume that all certificates are valid for the period $(15/04/2014, 17/04/2014)$. For simplicity we omitted the validity period in defining the certificates. However, the validity period can be added simply to the above delegation certificates. At time $(16/04/2014)$, k_S (a hotel broker) requests k_A for permission to book k_A's hotel rooms and runs the certificate chain discovery algorithm for permission $\langle k_A\ book\rangle$.

The process is as follows:

In local repository of principal S

Step 1S. Remove the useless certificates: certificates $c_8{}'$ and $c_9{}'$ grant permission $\langle k_A\ sell\rangle$, therefore are useless in discovering the certificate chain for permission $\langle k_A\ book\rangle$. The set of certificates that remains as a result of this step is:

$$c'_6 : {k_D}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} {k_S}^F$$

Step 2S. Compute the reduction rules over the set of certificates in step 1S which results in the following delegation statement:

$$k_D \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} k_S$$

Step 3S. Collect the original version of certificate c_6. The result of this step is the following certificate if found and stored in the local repository of S:

$${k_D}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} {k_S}^F$$

In local repository of principal D:

Step 1D. Remove the useless certificates: all of the certificates in repository D are related to granting permission $\langle k_A\ book\rangle$. The set of certificates from this step is:

$$c'_2 : (k_A\ hotel\ brokers) \longrightarrow k_D$$

$$c'_4 : {k_A}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} (k_A\ hotel\ brokers)^T$$

$$c_6 : {k_D}^T \stackrel{\langle k_A\ book\rangle}{\Longrightarrow} {k_S}^F$$

Step 2D. Compute the reduction rules over the set of certificates in step 1D. Comput-

ing the reduction rule 1 over certificates c_2 and c_4 results in the delegation statement:

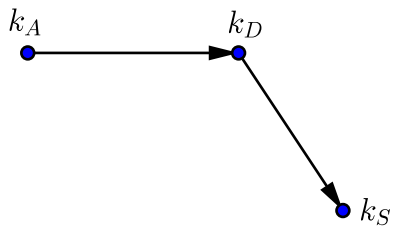
$$k_A \stackrel{\langle k_A\ book \rangle}{\Longrightarrow} k_D$$

Step 3D. Collect the original delegation certificates c_2 and c_4 from their issuers (k_A). The original version of these certificates will be sent to the local repository of S.

$$k_A{}^T \stackrel{\langle k_A\ book \rangle}{\Longrightarrow} k_D{}^T$$

In local repository of principal S

Step 4S. Set up a directed graph and do a Depth First Search to determine if there is a path from k_A to k_S. The result of this step is as follows:



The DFS algorithm returns the path: k_A, k_D, k_S. This shows that a certificate chain from k_A to k_S exists.

Step 5S. Produce the certificate chain: reconstruct and output the desired set of certificates from the information computed in the previous steps, consisting only of the input set of certificates. The resulting set of certificates to present to the resource owner k_A is as follows:

$$c_2 : (k_A\ hotel\ brokers) \longrightarrow k_D$$

$$c_4 : k_A{}^T \stackrel{\langle k_A\ book \rangle}{\Longrightarrow} (k_A\ hotel\ brokers)^T$$

$$c_6 : k_D \stackrel{\langle k_A\ book \rangle}{\Longrightarrow} k_S$$

Therefore, the requester k_S, signs an access request for $\langle k_A\ book \rangle$, along with the above set of certificates (result of step 5S) to the resource owner k_A. In the next chapter, we show how the policy engine determines that this set of certificates complies with the security policy to grant a request.

3.6 Discussion

Signed permissions are an effective approach to avoiding ambiguity in permission names. We followed SDSI's rationale for local names and introduced an extension to SPKI/SDSI that uses localPermissions in order to provide support signed permissions and thereby provide an authorization language that incorporates localPermission for subterfuge safe delegation. The logic supports truly decentralized access control whereby a principal may define, without rely on any central authority, its own permission locally and define a global orderings relative to permissions in other name spaces. Typical trust management systems make the implicit assumption that there exists a super security administrator that defines the global unique permissions. Many existing trust management systems such as PolicyMaker, KeyNote [46], SPKI/SDSI [43, 60], RT [86], and secPAL [10] are designed to specify arbitrary permissions. They assume unique and unambiguous permission names are provided by using a global name provider's services. For example, X.509 [40] uses X.500 naming service, the KeyNote uses the Internet Assigned Number Authority (IANA) [73]. In addition RT uses Application Domain Specification Documents (ADSDs) [86] to ensure the globally unique naming. Although, global name providers provide a unique interpretation for each name, the administrators may still use arbitrary names to represent their own resources. It depends on the expertise of the administrator who creates the permissions to specify non ambiguous permissions. Moreover, we introduced a simple certificate chain discovery algorithm that takes a set of certificates and returns a chain of certificates if one exists.

3.7 Summary

In this chapter, we introduced and developed the concept of localPermission, and introduced the SSTM framework to support subterfuge safe delegation of permissions in open environments. SSTM follows SDSI's rationale for choosing arbitrary names for principals and proposes an extension to SPKI/SDSI in order to provide support for signed permissions. An authorization language, SSAL, is provided to support truly decentralized and subterfuge safe trust management. A principal may define, without reference to any central authority, a permission in its own name space and define the global ordering relation of its localPermission with other permissions. Typical trust management systems make the implicit assumption that there exists a super security authority that defines the permission name space and ordering. In [87] a role-based distributed authorization language is described that provides subterfuge-freedom by constraint delegation to permissions that have an associated *"originating"* public key. While effective, this approach suffers the challenge of reliably referencing public keys and relies on a globally-defined function to define permission relationships (corresponding to an ordering). The FRM distributed policy management framework [88, 89] permits principals to locally define their permissions and orderings, and while it does permit a principal to define permission relationships with local policies of other principals, it is limited to permission orderings that form tree hierarchies. FRM also uses signed permissions to avoid subterfuge, but effectively relies on using public key values/X.509 certificates as principal identifiers.

The proposed logic comprises 15 axioms in addition to the original six axioms that describe SPKI/SDSI. 9 properties derived from these axioms provide some degree of confidence in the logic. Finally, we introduced a simple certificate chain discovery algorithm for SSTM. First, we addressed how to store certificates in a distributed manner. Then, we used an efficient algorithm to discover the certificate chain between a resource owner and the requester when certificates are stored with distributed principals. It has been shown that a DFS algorithm returns the chain if one exists.

Chapter 4

Ontology-Based Implementation for SSTM

In this chapter, we demonstrate an ontology-based approach, SSAL^O, that was introduced in [90]. SSAL^O represents the SSAL using a Description Logic (DL) [91] subset of the Web Ontology Language [92] (OWL-DL), and Semantic Web Rule Language (SWRL) [93]. SSAL^O is used as the SSTM policy engine where an incoming authorization request is evaluated by using a DL reasoner. Thus, the implementation provides a trust engine that enforces subterfuge-safe access to the protected resources of distributed parties. SSAL^O also provides a common domain model for integration of heterogeneous security policies. This approach is useful for secure cooperation and interoperability among principals in open environments, where each principal may have a different security policy with different implementation. We discuss the characteristics of SSAL^O in capturing SSAL and providing a framework for secure, automatic and dynamic integration of heterogeneous security policies specified by distributed principals in different domains. We employ various tools such as *Protégé* [94], *Pellet* [95], the Java programming language [96], and Jena framework [97] to implement our model.

This chapter is organized as follows: first we give the preliminaries in section 4.1, then we demonstrate the ontology model, SSAL^O in section 4.2. We later show how SSAL^O is used for integration of heterogeneous policies that may be implemented in different languages with different techniques. An example of using SSAL^O as policy engine is given. The work is then discussed in terms of its characteristics for automatic integration of locally defined policies to capture subterfuge safe trust management for cooperation of principals in open environments.

Finally, we summarize the chapter in section 4.3.

4.1 Preliminaries

4.1.1 Definition of Ontology

The term *Ontology* is borrowed from philosophy where ontology means a systematic account of existence [98]. An ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences. Ontology as *"an explicit specification of a conceptualization"* is its first definition in the context of computer science and information systems, defined by Gruber [99]. A more formal definition of ontology used by many researchers in the field [100–103], is as follows:

"An ontology is a formal, explicit specification of a shared conceptualization" [99]

Sharing of knowledge is one of the more common goals in developing ontologies. This facilitates organizations, programs, and humans to share and reuse knowledge. A *conceptualization* is an abstract model that consists of the relevant concepts and relationships that one wishes to represent for some purpose [104]. There must be a general accepted conceptualization of the specifications to be able to reuse the ontology by a community who have an interest in the corresponding knowledge. The core of an ontology is conceptualization which consists of the concepts, for example: *General Practitioner, Patient, Disease*; and the relationships that are assumed to be relevant to the concepts, for example: *"General Practitioner visits Patients", "Patient has Disease"*. Without conceptualization words have no precise semantics where the word *"Patient"* has two different meaning as a noun or adjective. Ontologies can be used for a range of applications such as information retrieval [105–107], and information integration [108–112]. In recent years, the use of ontologies for computer security, especially in the area of information security [113,114], access control [115–127], and trust management [128–133] has increased significantly.

4.1.2 Ontology Languages and Reasoning Tools

We describe the ontology languages and reasoning tools that we used in modelling SSAL^O. A sub-piece of the web ontology language that is based on description logics is called OWL-DL and is used to model the knowledge base for SSAL^O. SWRL is used for expressing a set of axioms introduced in SSAL. A DL reasoner is used for classification, realization and consistency checking of the knowledge base. The reasoning tool is also used to reason over the asserted knowledge and for inferring new knowledge.

Web Ontology Language

An ontology language is a formal language used to construct semantics for terms and syntax. OWL is a World Wide Web Consortium (W3C) recommendation to express meanings and semantics of an existing entity. OWL includes the definition of three variants with different levels of expressiveness as OWL-Lite, OWL-DL, and OWL-Full. The OWL-DL language is most closely related to the $SHOIN(D)$ description logic [134]. In $SHOIN(D)$, S stands for ALC (Attributive Concept Language) [135] plus role transitivity, H stands for role hierarchy, O stands for nominals, I stands for inverse role, N stands for cardinality restrictions, and D stands for data types. In order to encode knowledge in OWL-DL, an understanding of the constructors for $SHOIN(D)$ is necessary. This is given in the following sections.

Description Logic

The Description Logic (DL) [91] is a decidable fragment of First Order Logic (FOL) [136] and constitutes the formal basis for OWL-DL, a very expressive and yet decidable subspecies of OWL. Building an ontology requires the use of a logic as a means of axiomatizing [137]. Description Logics (DLs) [91] are a family of concept based knowledge representation formalisms. They are characterized by the use of constructors to describe complex concepts and relations among concept instances which form a decidable subset of First Order Logic (FOL) [136]. This decidability is very convenient for reasoning about ontologies. FOL is not decidable, which means that it is not possible to know in advance the validity of a formula and the computation can run forever without giving an answer [138]. However, although the DLs have some limitations in expressiveness

such as the absence of variables, they ensure decidability. Knowledge in DLs is represented in a hierarchical structure of concepts (or classes). These concepts are defined in terms of some specified properties (or roles) that individuals must satisfy in order to belong to those concepts. In other words, concepts represent sets of instances, and properties represent binary relations between instances. Making an individual, an instance of a concept, is called *instantiation* [139]. Concepts are either atomic (those identified by a name) or complex (those derived from atomic concepts using a set of constructors). DL basics include *Concepts* (unary predicates, corresponding to classes in OWL), *Roles* (binary predicates, corresponding to properties in OWL), *Individuals* (constants, corresponding to instances in OWL), and *Operators* (corresponding to constraints in OWL).

A DL-based ontology consists of a set of terminological axioms (called TBox) and assertional axioms (called ABox). Concepts and properties are separated from instances by partitioning the knowledge base into the TBox and the ABox. The TBox is constructed through declarations that describe properties for concepts. The ABox contains extensional knowledge that is specific to instances of concepts of the domain of interest.

Constructors Description logic can be used to represent much more than just concepts, properties and instances. A description logic also offers a formal syntax which specifies how to construct well formed statements. It also provides formal semantics for relating those statements to a model. Statements are formulas to represent concepts, properties, and instances. Thus, the constructors are used to derive well formed formulas. *SHOIN(D)* as a family of DLs uses several kinds of constructors and axioms. These constructors and axioms allow building complex concepts and property relationships from atomic concepts, and atomic properties. Note that, atomic concepts correspond to the unary predicates in FOL, atomic properties correspond to the binary predicates in FOL. An OWL-DL knowledge model consists of the Tbox and the Abox; where Tbox contains axioms relating to concepts and properties, while the ABox contains axioms relating to individuals. Both ABox and TBox utilize a comprehensive set of constructors (such as intersection, union, universal quantifier, or existential quantifier) to derive well formed formulas. The following constructors are used in this thesis.

Intersection ($\sqcap$): It is interpreted as the intersection of sets of individuals. For example, the intersection of concepts *Patient* and *Doctors* is expressed as the

following:

$$Patient \sqcap Doctors$$

This denotes those instances of concept *Patient* that are shared with concept *Doctors*.

Union ($\sqcup$): It is interpreted as the union of sets of individuals:

$$Patient \sqcup Doctors$$

This denotes those individuals that belongs to either concept *Patient* or *Doctors*.

Existential quantifier ($\exists$): Constrains those individuals that have a relationship to instances of some concept. The following expression:

$$\exists visits.Doctors$$

is the set of individuals, each of which has the property relation *visits* to some instances of concept *Doctors*.

Universal quantifier($\forall$): Constrains those individuals that have a relationship to only instances of some concept. The following expression:

$$\forall hasPatient.Patient$$

is the set of individuals that have the relationship via *hasPatient* property to only instances of concept *Patient*. Note that, universal quantification does not ensure that there will be a property that satisfies the condition, but it guarantees that if there is such a property, its range has to be constrained to the given concept or type. In the table 4.1, we list some of very common constructors in *SHOIN(D)*, their notations, and their semantics as they are used in the ontology.

Table 4.1: Concrete syntax of DL constructors

Constructor	DL Syntax	Example
Intersection	$C_1 \sqcap ... \sqcap C_n$	Patient $\sqcap$ Doctors
Union	$C_1 \sqcup ... \sqcup C_n$	Patient $\sqcup$ Doctors
Complement	$\neg C$	$\neg$ Patient
Universal quantifier	$\forall P.C$	$\forall$ hasPatient.Patient
Existential quantifier	$\exists P.C$	$\exists$ visits.Doctors
MaxCardinality	$\leq_n P$	$\leq_1$ hasPatient
MinCardinality	$\geq_n P$	$\geq_1$ hasDoctor
ExactCardinality	$=_n P$	$=_1$ patientRoom

Notation: C and D are concepts, P is property.

Table 4.2: A comparison of *SHOIN(D)* and OWL-DL constructors

Constructor	$SHOIN(D)$	OWL-DL
Intersection	$C_1 \sqcap C_2$	intersectionOf(C_1, C_2)
Union	$C_1 \sqcup C_2$	unionOf(C_1, C_2)
Complement	$\neg C$	complementOf(C)
Universal quantifier	$\forall P.C$	allValuesFrom(C) on Property(P)
Existential quantifier	$\exists P.C$	someValuesFrom(C) on Property(P)
MaxCardinality	$\leq_n P$	maxCardinality(n) on Property(P)
MinCardinality	$\geq_n P$	minCardinality(n) on Property(P)
ExactCardinality	$=_n P$	exactCardinality(n) on Property(P)

In addition, the OWL-DL constructors along with the corresponding $SHOIN(D)$ constructors, are listed in the table 4.2.

Description Logic Axioms Axioms in an OWL- DL ontology can be classified according to the knowledge they describe as TBox entities or ABox entities. A description logic knowledge base KB may be defined as the tuple consisting of a TBox T and an ABox A. $KB = (T, A)$, where T is the union of the set of concepts with the set of property relations in the domain, and A is the set of individuals in the domain. Furthermore, the TBox also contains axioms relating to concepts and properties, while the ABox contains axioms relating to individuals. Based on this categorization various DL axioms will be explained in the following.

Terminological Axioms Terminological axioms are statements related to TBox entities as concepts and properties, but not individuals. These axioms can be classified as either *inclusion* or *equality* axioms.

Inclusion An inclusion ($\sqsubseteq$) states a necessary but not sufficient condition for being an instance of some concept. The first type of inclusion is simply a *concept inclusion*. The inclusion statement $C \sqsubseteq D$ is interpreted that for a TBox entity to be included in the concept C, it is necessary to have the condition D. However this condition D alone is not sufficient to conclude that the individual (or any object) is in the concept C. In other words, if a random TBox entity satisfies condition D, it does not necessarily belong to the concept C. An example of this kind of axiom is:

$$Surgeon \sqsubseteq (Doctors \sqcap Consultant)$$

where an individual who is a doctor and a consultant, is not necessarily a surgeon.

The second type of inclusion axiom is the *specialization*, which has the abstract form $C \sqsubseteq A$. This is similar syntax to that of the inclusion axioms, but the right hand side of a specialization must be atomic (hence A). It indicates that having properties of concept C is necessary for an entity in order to be included in concept A. An example of specialization is:

$$Patient \sqsubseteq isRegisteredWith.GP$$

A specialization axiom is useful when some concepts cannot be defined completely. The third type of inclusion is *taxonomy*. Concepts can be organized as a hierarchy which is also known as a taxonomy. A concept can have sub-concepts that represent the concepts that are more specific than the super-concept. For example, the concept *Doctors* has the sub-concepts *GP* and *Surgeon*.

$$GP \sqsubseteq Doctors$$

$$Surgeon \sqsubseteq Doctors$$

Being a member of concept *GP* or *Surgeon* implies membership of its super-concept *Doctors*. In addition, either *GP* or *Surgeon* membership constraints will

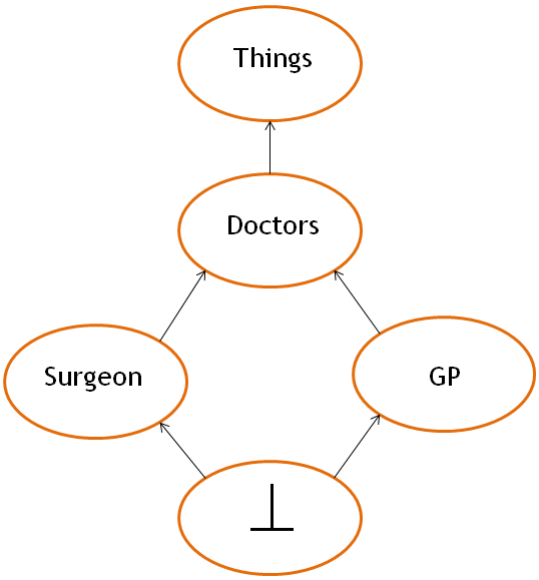


Figure 4.1: Concept hierarchy

be inherited from their super-concept *Doctors*. Concept inclusion axioms are very important in the structure of the knowledge base as they are used to generate a taxonomy from a set of assertions in a TBox. Figure 4.1 depicts the concept hierarchy.

Equation A concept equation of the form $(C \equiv D)$ states necessary and sufficient conditions that a TBox entity must hold in order for it to be included in some concept. By including sufficient conditions, then any random TBox entity that satisfies conditions D must be included in concept C. An example of this kind of axiom is:

$$Doctors \equiv hasPatient.Patient$$

This states that an individual is a member of concept *Doctors* if and only if it has the property relation *hasPatient* to members of concept *Patient*. Furthermore, this axiom is a *concept definition*. A special kind of equation is a concept definition of the form $A \equiv C$ where the left hand side is an atomic concept. A concept definition states the necessary and sufficient conditions that must hold in order for a TBox entity to be included in some other concept. Having property C is necessary and sufficient for a TBox entity to be included in concept A. Another form of concept equation is a *covering axiom* that will be discussed later.

Assertional Axiom Assertional axioms are statements related to ABox entities. These axioms can be classified as either *concept assertion* or *property assertion* axioms. A concept assertion is of the form $C(i)$, where C is some concept from the TBox and i is an individual, representing i as an instance of concept C. An example is:

$$Doctors(\texttt{DrAlice})$$

A property assertion is of the form *P(i,j)*, where P is some property from the TBox and `i` and `j` are individuals; where *P(i,j)* means that individual `i` has a P relation to individual `j`. An example of a property assertion axiom is given in the following:

$$hasPatient(\texttt{DrAlice,Bob})$$

Modelling in OWL-DL

OWL-DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will be finished in finite time). OWL-DL includes all OWL language constructs, but they can be used only under certain restrictions of DLs. For instance, while a concept may be a sub-concept of many concepts, a concept cannot be an instance of another concept.

Concept Concepts are sets of instances that are described using formal descriptions. The formal description specifies the conditions and requirements that must be satisfied by an individual to be a member of a concept. Being an instance of a concept includes either explicitly asserting an individual as an instance of a concept or implicitly inferring an instance as a result of reasoning over the asserted knowledge (for example, subsumption constraints) [140]. There is a general built-in concept named *Thing* which includes all individuals and is the super-concept of all concepts. We may choose to make a new sub-concept of the concept *Thing* named *Doctors*. We may also create a new concept named *Surgeon* that is a sub-concept of *Doctors*. From this, a reasoner can deduce that any instance of

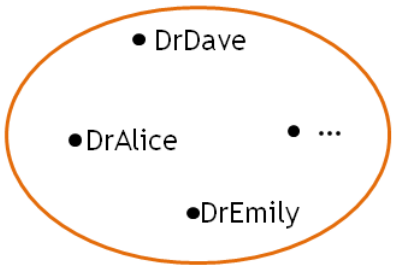


Figure 4.2: Example of concept membership

the concept *Surgeon* is also an instance of the concept *Doctors*. Note that there is no limitation on cycle creation in sub-concept hierarchies. For example, the following is an instance assertion for concept *Surgeon*.

$$Surgeon(\texttt{DrAlice})$$

Moreover, the following DL notation is the result of reasoning over the subsumption constraint that infers `DrAlice` as the instance of concept *Doctors*.

$$Doctors(\texttt{DrAlice})$$

Figure 4.2 is an example of concept membership.

Property Properties are binary relations that instances of concepts can have between one another. For example, the property *hasPatient* might link the instance `DrAlice` of concept *Doctors* to the instance `Bob` of concept *Patient* as:

$$hasPatient(\texttt{DrAlice,Bob})$$

Each property can have an inverse which provides an inverse of a given relation. For example, the property *hasPatient* relates the instances of concept *Doctors* to the instances of concept *Patient*. The inverse of the *hasPatient* property can be the *hasDoctor* property which relates the instances of concept *Patient* to the instances of concept *Doctors*. This is shown in the following:

$$hasPatient \equiv hasDoctor^{-}$$

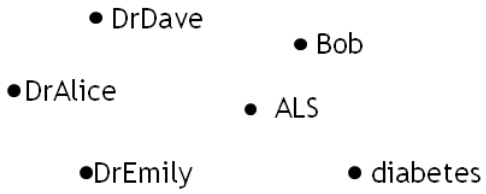


Figure 4.3: Set of individuals

Individual A concept can be instantiated by individuals. The instantiation of concepts with individuals makes an individual be an instance of some concepts. Properties may also be used to relate one individual to another. For example, an individual named `Bob` may be described as an instance of the concept *Patient* and the property *hasDoctor* may be used to relate the individual `Bob` to the individual `DrAlice`. Figure 4.3 illustrates individuals.

In this thesis, concepts and properties are written in italic font; a *Concept* begins with an Upper case letter and a *property* begins with a lower case letter. Note that `instances` are written in a lower case typewriter font.

Domain and Range It is possible to add domain and range restrictions to properties thereby restricting a given property to taking particular concepts as its domain and particular concepts as its range. Therefore, by defining a domain restriction for a property, the former individual specified with that property is assumed to belong to the concept(s) specified in the property domain. Furthermore, the latter individual specified with that property must belong to the specified range concept. For example, the following DL fragment limits the domain and range of property *hasPatient* to only the instances of concept *Doctors* (as domain) and *Patient* (as range).

$$Doctors \sqsubseteq hasPatient.Patient$$

Open World Assumption Open World Assumption (OWA) is the opposite of Closed World Assumption (CWA). The Closed World Assumption (CWA) is the assumption that what is not known to be true must be false. On the other hand, OWA is the assumption that what is not known to be true is simply unknown. For example, consider the following statement: "Bob is a patient". Now, what

if we were to ask “Is Bob a doctor?" Under a CWA, the answer is "no". Under the OWA, the answer is "it is not known". The CWA applies when a system has complete information. CWA is the logical basis for traditional database systems. OWA applies when a system has incomplete information. This is the case when we want to represent knowledge and discover new information. In the knowledge base (ontology) absence of information means that the information has not been made explicit and further knowledge may make it explicit. Hence, OWA is an essential aspect of knowledge base systems. Some of the OWA closures in description logic are: *disjointness, unique name assumption, covering and closure axioms.*

Disjointness In DLs, not asserting an individual as an instance of a certain concept does not mean it is not an instance of that concept. It must be asserted that two concepts do not share any instances. This relation between concepts is called *disjoint concepts.* For example, consider the concept *GP* versus concept *Surgeon.* These two concepts may never share individuals since a given general practitioner (GP) can never be interpreted as a surgeon, i.e.

$$GP \sqcap Surgeon \equiv \bot$$

where the bottom concept $\bot$ is the special concept with no individuals as instances. The above axiom thus says that the intersection of the two concepts *GP* and *Surgeon* is empty.

Covering Axioms In addition to disjointness, it is important to consider whether some set of sub-concepts fully covers the super-concepts. The Open World Assumption makes it possible that being instances of a super-concept without being also instances of its sub-concept. Figure 4.4 indicates that the super-concept *Doctors* may have other instances that are not an instance of its sub-concepts *GP* and *Surgeon.*

However, by defining covering axioms, it is necessary to explicitly state if an individual is an instance of a super-concept, then it must be an instance of at least one of its sub-concepts. Moreover, if the sub-concepts are defined as disjoint concepts then an instance of the super-concept must be an instance of one of its sub-concepts. Figure 4.5 illustrates the covering axiom for concept *Doctors* and its sub-concepts *GP* and *Surgeon* that are disjoint with each other. The following DL assertion defines a covering axiom for concept *Doctors* that guarantees it will only have two sub-concepts.

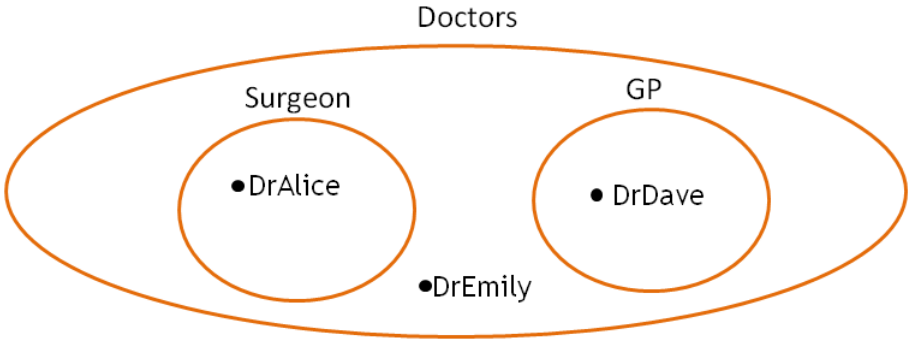


Figure 4.4: OWA without covering axiom

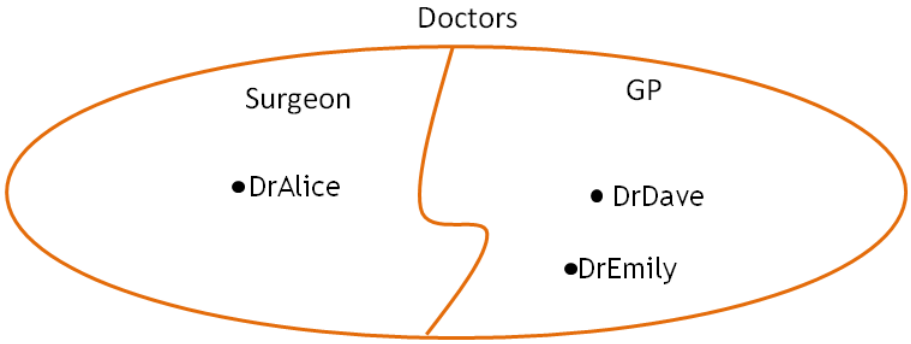


Figure 4.5: OWA considering covering axiom

$$Doctors \equiv GP \sqcup Surgeon$$

Closure Axiom Remembering the Open World Assumption, we need to define a closure axiom if we want to restrict the possibility of further additions for a given property. For example, the following fragment of DL assertion represents the closure axiom on the *hasDoctor* property for the concept *Patient* :

$$Patient \sqsubseteq \exists_{=1} hasDoctor.Doctors$$

where the existential quantifier ($\exists$) acts on the property *hasDoctor* having the concept *Doctors* as property range. This closure states that to be an instance of concept *Patient*, an individual must have exactly one *hasDoctor* property relation to the instances of concept *Doctors*.

Unique Name Assumption The Unique Name Assumption (UNA) refers to a case where if two individuals have different names they are, by default, different. DLs do not usually make the Unique Name Assumption, and indeed our formal definition allows two individual names to be interpreted as the same individual. Therefore, we have to make it explicit whether two names denote the same or distinct individuals. For example, an instance of concept *Patients*, say `Bob` may have only one relationship to exactly one instance of concept *Doctors* with property *hasDoctor*. Assuming individuals `DrAlice` and `DrClare` are both inferred as instances of concept *Doctors*, we can have the following individual instantiation:

$$Patient\ (\texttt{Bob}) \longleftarrow hasDoctor(\texttt{Bob}, \texttt{DrAlice}) \sqcap$$
$$hasDoctor(\texttt{Bob},\ \texttt{DrClare})$$

`DrAlice` and `DrClare` must be explicitly asserted as distinct individuals. Otherwise, the cardinality restriction will consider the individuals `DrAlice` and `DrClare` as the same individuals. Moreover, if the individuals `DrAlice` and `DrClare` have been asserted as distinct individuals, the above instantiation would be inconsistent, since `Bob` must have the *hasDoctor* relationship to exactly one instance of concept *Doctors*.

Reasoning in Description Logic

The term *reasoning* refers to automatic inference of further implicit facts from explicitly asserted statements within the ontology. An ontology contains knowledge as structured data. The users of an ontology are typically interested in obtaining information about relationships between concepts described in the ontology and querying about the knowledge existing in the ontology. Both tasks require reasoning tools; tools that can derive new knowledge from the ontology's explicit knowledge. A DL reasoner provides a set of description logic inference services such as *Consistency checking, Classification* and *Concept satisfiability* [95]. Consistency checking ensures that the ontology model does not contain any contradictory facts. For example, given individual `DrAlice` is an instance of concept *Surgeon*, it can be easily inferred that `DrAlice` is an instance of concept *Doctors* if one can Figure out that the concept *Doctors* subsumes the concept *Surgeon*. Classification refers to creating a concept hierarchy by computing the subsump-

tion relations. By reasoning over an ontology a new concept or individual can be assigned automatically to the correct taxonomy classification. Concept satisfiability refers to the possibility for a concept to have some instances. If a concept is un-satisfiable, then creating an instance of that concept causes the entire ontology to become inconsistent. A knowledge base modelled in OWL-DL can be reasoned over by a DL reasoner such as *Pellet* [95].

Semantic Web Rule Language

The Semantic Web Rule Language (SWRL) [93, 141] is a rule language that extends the semantic information described in an OWL (consequently in an OWL-DL) ontology. Since some policy rules may not be expressed in OWL-DL, therefore, we use SWRL. SWRL is based on a combination of the OWL-DL sublanguage of the Web Ontology Language and the Unary/Binary Datalog RuleML sub-language of the Rule Mark up Language [142]. It includes a high-level abstract syntax for Horn-like rules in OWL, that enables extra knowledge expressibility and the use of decidability tools (www.w3c.org). SWRL is based on OWL and all rules are expressed in terms of OWL entities as concepts, properties, individuals, literals and so on. SWRL allows one to write rules expressed in terms of OWL concepts and properties. The rules can be used to infer new knowledge from existing OWL knowledge bases. A SWRL rule syntax is as follow:

$$(a_1 \wedge ... \wedge a_n) \longrightarrow b$$

where a_i $(i = 1..n)$ is an antecedent; and b is a consequence, which both consist of one or more atoms [141]. The conjunction constructor $\wedge$ is interpreted as "and". Atoms (a_i $(i = 1..n)$, and b) can be of the form:

- $C(i)$, where i is an instance of concept C.
- $p(i, j)$, where instance i is related to instance j via the object property p.
- *sameAs(i,j)*, where determines two instances i and j are interpreted as the same instances.
- *differentFrom(i,j)*, where determines the two instances i and j are not the same instances.
- *Built-in*$(b, a_1, ..., a_n)$, represents a Built-in specification b is satisfied if a set of data type variables or data type values $a_1, ..., a_n$ within a particular

built-in are true.

- $D(a)$, where a is an instance of data type D.
- $u(i, a)$, where instance i is related to data type variable or data type value a via the data type property u.

i and j represent object variables or individuals. It is denoted with a "?" prefix or `typewriter` font, respectively.

The SWRL antecedent is satisfied, if it is empty (trivially true) or every atom of it is satisfied. The SWRL atom in the consequent is satisfied if it is not empty and atoms in the antecedent are satisfied. For example, a SWRL rule to express that all surgeons (expressed by variable $?d$) in OWL-DL are to be inferred as both being a doctor and a PhD, can be expressed with the following assertion:

$$Doctors(?d) \wedge PhD(?d)$$
$$\longrightarrow Surgeon(?d)$$

It is possible to express some rules using only DLs depending on the number of variables shared between consequent and antecedent. However in practice, SWRL rules are used to express what is not expressible in DL (when two or more variables are shared between antecedent and consequent).

Modelling in Semantic Web Rule Language The SWRL language includes support for user-defined built-ins that are common to most programming and scripting languages [140, 143]. SWRL built-ins are predicates that accept several arguments. All built-ins in SWRL must be preceded by the namespace qualifier"*swrlb:*". SWRL built-ins are categorized as: comparison operators, mathematical operators, boolean values, strings, date, time, URIs, TBox, Abox, and list operators [143, 144]. Using built-ins will provide the flexibility for expressing complex rules. The following SWRL rule:

$$Doctors(?d) \wedge numberOfPatient(?d, ?n) \wedge swrlb : lessThanOrEqual(?n, 50)$$
$$\longrightarrow GP(?d)$$

states that any *Doctors(?d)* that has a a number of patients *(?n)* less than or equal to 50 (swrlb :lessThanOrEqual) is to be classified as an instance of the *GP* concept. SWRL supports the Open World Assumption. This means that the SWRL does not support negation as failure, cardinality restrictions, and Unique Name Assumption. It supports *sameAs* and *differentFrom* clauses. *sameAs* atom can determine if two individuals are the same individual. Similarly, the *differentFrom* atom can be used to express that two individuals are not the same. In the following, we explain these in some concrete examples (Note that, invalid SWRL syntax is denoted between the "<" and ">" symbols):

Example1 With SWRL, it is not possible to retract or remove facts from an ontology. The following SWRL rule states that any doctor, *?d*, having a greater than or equal to 50 patients will have the range of the property isRegisteredDoctor set to `true`.

$$Doctors(?d) \wedge numberOfPatient(?d, ?n) \wedge$$
$$swrlb : greaterThanOrEqual(?n, 50)$$
$$\longrightarrow isRegisteredDoctor(?d, \texttt{true})$$

However, careful consideration is needed when adding new facts to the ontology as these new facts may conflict with existing facts. For example, if a doctor for whatever reason had its isRegisteredDoctor property previously initialized with a relationship to the individual `false`, then that doctor may result in the isRegisteredDoctor property having both boolean values. As the isRegisteredDoctor is intended to be functional, the DL reasoner will indicate an inconsistency with the newly added facts inferred by SWRL inferencing.

Example 3 Negation as failure is not supported by SWRL. This is because previously unknown facts yet to be discovered may invalidate a SWRL rule's conclusion. For example, it is not possible to state that individuals of concept *Doctors*, not being an instance of concept *Surgeon*, should be classified as instances of concept *GP*.

$$Doctors(?d) \wedge < \textbf{ not } \textbf{\textit{Surgeon}} >$$
$$\longrightarrow GP(?d)$$

Example 4 Both *differentFrom* and *sameAs* clauses are used by SWRL rules to determine whether individuals that are identified by different names are in fact the same individual or distinct individuals. These clauses are used in association with DL axioms that define a particular set of individuals to be mutually distinct or refer to the same individual. For example, stating that two independent instances of concept *Patient* which have the same disease belong to the same department (sameDept property relationship), requires explicitly defining both instances to be distinct (*differentFrom*). This is represented by the following SWRL rule.

$$Patient(?p1) \wedge Patient(?p2) \wedge differentFrom(?p1, ?p2) \wedge$$
$$hasDisease(?p1, ?ds1) \wedge hasDisease(?p2, ?ds2) \wedge$$
$$swrlb : equal(?ds1, ?ds2)$$
$$\longrightarrow sameDept(?p1, ?p2)$$

Semantic Query-Enhanced Web Rule Language

Semantic Query-Enhanced Web Rule Language (SQWRL) [145] is an expressive OWL query language that uses the SWRL semantics and serialization and supports comprehensive querying of OWL, and is defined based on the SWRL for retrieving knowledge from OWL [145, 146]. It takes a SWRL rule antecedent as a pattern specification for a query and replaces the consequent with a retrieval specification. For example, the most common SQWRL consequent is the *sqwrl:select* operator, which takes one or more arguments (variables) that correspond to those already specified in the antecedent. It builds a table where arguments form columns of the table and the retrieval knowledge corresponding to the arguments form each row. SQWRL queries can operate in conjunction with SWRL rules in an ontology and can be used to retrieve knowledge inferred by those rules. Note, all valid SWRL rule antecedent built-ins are valid within SQWRL. SQWRL queries do not modify the knowledge within the ontology. The following SQWRL query returns pairs of doctors and their associated patients :

$$Doctors(?d) \wedge hasPatient(?d, ?p)$$
$$\longrightarrow sqwrl : select(?d, ?p)$$

SQWRL built-ins include basic counting (*sqwrl:count*) and aggregation (*sqwrl:min*, *sqwrl:max*, *sqwrl:sum*, *sqwrl:avg*). These built-ins operate on the query results and not on the underlying ontology. The *sqwrl:count* built-in, for example, keeps track of the number of relevant items matched in a query, not the number of such items in the ontology being queried. Similar to SWRL, the open world assumption and the unique name assumption are supported in SQWRL. SQWRL queries can also operate in conjunction with SWRL rules to retrieve knowledge inferred by those rules. These inferences can be used by other rules and queries. In addition to the query functionality, queries with more complex closure requirements can not be expressed using these built-ins. For example, queries with negation as failure, complex aggregation, or disjunction are not expressible. The set operators are added to support these requirements. These operators include *sqwrl:makeSet, sqwrl:groupBy, sqwrl:union, sqwrl:difference,* and *sqwrl:intersection*.

4.1.3 Methodology

Building an ontology is essentially a three stage process. First, design the TBox for the knowledge base; second, populate the ABox with individuals; and third, relate TBox and ABox. Below lists these steps specifically for building an ontology.

1. Design the TBox for the knowledge base. a TBox consists of Concepts, Properties, and Individuals.
    - 1.1 Classify entities as concept, property, or individual.
    - 1.2 Add concepts to TBox.
        - 1.2.1 Declare atomic concepts.
        - 1.2.2 Define non-atomic (constructed) concepts.
        - 1.2.3 Create concept taxonomy.
        - 1.2.4 Partition the concept taxonomy.
    - 1.3 Add properties to TBox.
        - 1.3.1 Declare transitive and symmetric properties.
        - 1.3.2 Declare inverse properties.
        - 1.3.3 Declare functional properties.

1.3.4 Add domain and range restrictions to properties.

1.3.5 Add cardinality restrictions to properties.

1.4 Add other axioms to further refine concepts and properties.

2 Populate the ABox with individuals.

2.1 Enumerate and classify each individual according to available concepts.

2.2 Relate individuals via available properties in the ontology.

3 Relate TBox and ABox.

3.1 Create enumerated concepts.

3.1 Relate individuals to concepts via properties.

The details of designing TBox, ABox, and relating them will be showed in next section.

4.2 SSAL^O

The Subterfuge Safe Authorization Language (SSAL) was introduced in chapter 3 as a policy language with the purpose of subterfuge safe delegation of permissions among principals [147]. In addition to the design of SSAL for trust management, it is also important to capture a common vocabulary that is understandable by different distributed principals. Having a common vocabulary allows information sharing and reuse, and therefore facilitates integration of security policies defined by individual principles for access to their own resources. Using an ontology facilitates sharing and reuse of knowledge and interoperability in the domain of security policies. For a policy language, the ontology can express the policy statements and certificates in a conceptual way. In addition, the ontology provides a common vocabulary for integration of heterogeneous policies defined by different principles in distributed environments. In this section, we demonstrate an ontology-based approach that implements SSAL as well as allowing integration of heterogeneous security policies for subterfuge safe trust management. This approach, SSAL^O, represents SSAL using OWL-DL and SWRL. This implementation provides a policy engine that enforces subterfuge safe authorization of requests for accessing the protected resources of distributed principals. SSAL^O also provides a common domain model for integration of heterogeneous security policies. This approach is

useful for secure cooperation and interoperability among principals in open environments (such as coalitions) where each principal may have a different security policy and different implementation. We discuss the characteristics of SSAL^O in capturing SSAL and providing a framework for secure and dynamic integration of heterogeneous security policies specified by distributed principals in different domains. We employ various tools such as *Protégé*, *Pellet*, the Java programming language, Eclipse workbench (www.eclipse.org), and Jena framework to implement our model.

The objective of this section is to develop an ontology for SSAL, with which to reason about a set of assertions and SSAL rules. An ontology-based approach models the SSAL policy language using the OWL-DL [92] and the *Protégé* [94] knowledge-modelling tool. This section is a revised and extended version of the work presented in [90]. SSAL^O is intended to be used as a policy engine. To build the SSAL^O, we follow the methodology described in section 4.1.3 which consists of designing the TBox, filling the ABox with individuals, and relating TBox and ABox. Figure 4.6 depicts an overview of the SSAL^O.

4.2.1 Design the TBox for SSAL^O

SSAL^O includes five concepts as: *Principal, Key, LocalName, LocalPermission* and *Delegation*. It also includes sixteen object properties: *asAuthAs, delegatesPermission, hasDelegator, hasDelegatee, hasNameSpace, hasOriginator, holds, isHeld, speaksFor, isSpokenBy, isAccountable, isAccountableBy, impliesTarget, impliesAction, hasTarget,* and *hasAction*. The property *isHeldBy* is the inverse of property *holds*, property *speaksFor* is the inverse of property *isSpokenBy*, and the property *isAccountable* is the inverse of property *isAccountableBy*. In addition, SSAL^O includes a data type (string) property: *hasName*. All these concepts and properties correspond to the entities in SSAL. For instance, the *speaksFor, asAuthAs* OWL properties correspond to the "*speaks for*" relationships among the principals, and "*no less authoritative than*" ordering relationship among localPermissions introduced in SSAL, respectively. In the following, we define the concepts using property definitions. The constructor ($\forall$) is a value restriction which all values of a property for instances of a concept must belong to the specified concept or data type. The constructor ($\exists$) is an existential quantifier that restricts those individuals which have a relationship to instances of a concept. The constructor ($\sqcup$) is interpreted as a union of sets of individuals and the constructor ($\sqcap$) is interpreted as the intersection of sets of individuals. Inclusion ($\sqsubseteq$) states

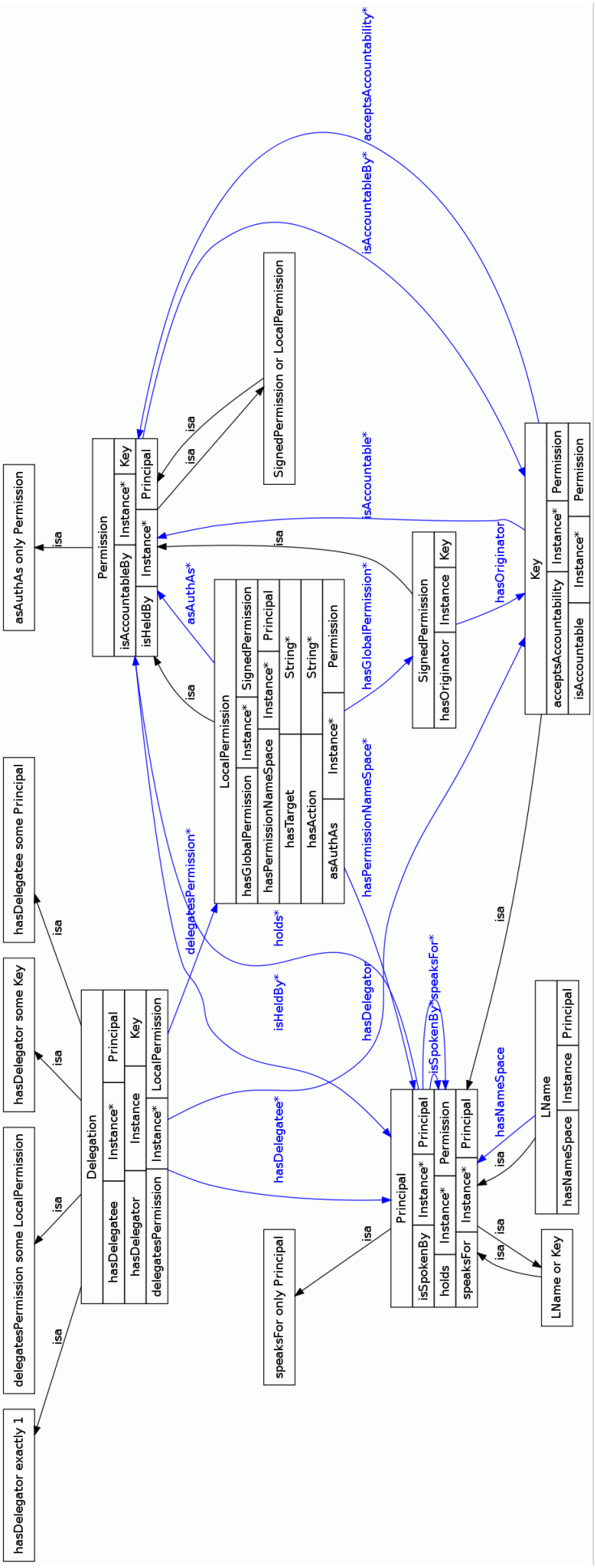


Figure 4.6: An overview on SSAL^O

a necessary but not sufficient condition for being an instance of a concept, and equality ($\equiv$) states necessary and sufficient conditions that an individual must hold in order for it to be included in a concept. The cardinality restriction ($=, \geqslant$) specifies the intended number of relationships that an individual must have for a given property.

Concept: Principal

This concept models the principal entity which was defined in the SSAL domain. A principal is identified by either its public key or local name. The concept *Principal* subsumes two sub-concepts as *LocalName* and *Key*; where an instance p is a member of concept *Principal* if and only if it is an instance of either *LocalName* or *Key*. This concept is expressed with the following restrictions:

$$Principal \equiv (LocalName \sqcup Key)$$

From this definition we do not know whether an individual which is a local name is also a public key or not. We can specify that any individual that is a local name is not a public key and vice-versa. That is, the set of public keys and the set of local names are disjoint in the following way:

$$(LocalName \sqcap Key) \sqsubseteq \bot$$

This states that the set formed by the intersection of concepts *LocalName* and *Key* will always be empty. An instance for example, $\mathtt{K_A}$, belongs to *Principal* concept if and only if it belongs to either concept *Key* or *LocalName*. Similary, $\mathtt{K_A}$ belongs to concept *Key* and concept *Principal*, if and only if it does not belong to concept *LocalName*. Figure 4.7 gives an overview of concept *Principal*.

Concept: LocalName

A local name is an arbitrary name N that principal P (identified by its public key) chooses for principal Q in its name space. Principal P refers to principal Q in its name space as N and uses the *speaks for* relation to link the local name to

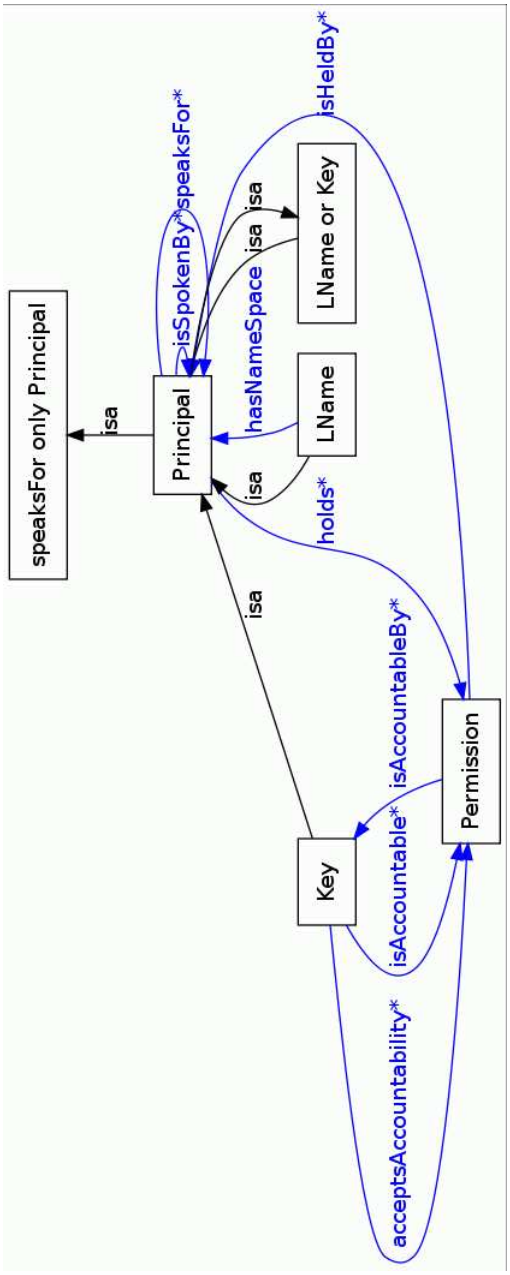


Figure 4.7: An overview on concept Principal in SSAL^O

principal Q. It means that any statement that is signed by Q can be viewed as originating from N in the name space of P. The concept *LocalName*, subsumed by concept *Principal* and disjoint with concept *Key*, is the conceptualization of the local name entity described in SSAL. This concept is expressed by the following restrictions:

$$\begin{aligned} LocalName \sqsubseteq\ & Principal \sqcap \\ & \exists\ isSpokenBy.Principal \sqcap \\ & \exists_{=1} hasNameSpace.Principal \sqcap \\ & \exists_{=1} hasName.string \end{aligned}$$

This expresses that the concept *LocalName* is a sub-concept of concept *Principal*. An individual belongs to the concept *LocalName* if it has the object property relation *isSpokenBy* (corresponding to the inverse of *speaks for* relation in SSAL) to at least one instance of the concept *Principal*. Similarly, the individual must have the object property relation *hasNameSpace* to exactly one instance of concept *Principal* (either an instance of concept *Key* or *LocalName*), and have data type property relation of *hasName* to string type. For example, an instance say, $\mathtt{k_B Alice}$ belongs to concept *LocalName* if it has the object property relation *isSpokenBy* to at least one instance of concept *Principal* such as $\mathtt{k_A}$. Similarly, $\mathtt{k_B Alice}$ belongs to the concept *LocalName* if it has the object property relation of *hasNameSpace* to exactly one instance of concept *Principal* (either an instance of concept *Key* or *LocalName*, in this example $\mathtt{k_B}$), and has data type property relation of *hasName* to string `Alice`. The individual $\mathtt{K_B Alice}$ as an instance of concept *LocalName* is defined as the following:

$$\begin{aligned} LocalName(\mathtt{k_B Alice}) \longleftarrow\ & hasNameSpace(\mathtt{k_B Alice, k_B}) \sqcap \\ & hasName(\mathtt{k_B Alice, Alice}) \sqcap \\ & isSpokenBy(\mathtt{K_B Alice, k_A}) \end{aligned}$$

Concept: Key

The concept *Key*, a sub-concept of *Principal* and disjoint with concept *LocalName*, models the public key entity described in SSAL as the global unique identifiers for principals. This concept is captured in SSAL^O with the following constraint:

$$Key \sqsubseteq Principal$$

which states that the concept *Key* is subsumed by concept *Principal*. This subsumption is valid if all instances of concept *Key* are necessarily instances of concept *Principal*.

Concept: LocalPermission

Each principal chooses a permission specification arbitrarily for its own resource and binds that specification to its name space (identified by a public key or local name). The concept of *LocalPermission* is defined to model the localPermission introduced in chapter 3. The binding of a permission to its name space is captured with the *hasNameSpace* property in SSAL^O. In addition, each permission must be originated by a principal and this is captured in SSAL^O with the *isHeldBy* property. There is always an ordering relationship (either implicitly or explicitly) among permissions in local policies defined by principals. The localPermission ordering relationship "*no less authoritative than*" is represented using the *asAuthAs* relation in the ontology where the statement *asAuthAs(Y,X)* indicates that permission Y is *no less authoritative than* permission X. The following necessary and sufficient condition restricts an individual to be included as an instance of this concept. This concept is defined with the following necessary condition that an individual must hold to be included in concept *LocalPermission*:

$$
\begin{aligned}
LocalPermission \equiv \\
&\exists_{=1} hasNameSpace.Principal \sqcap \\
&\exists_{\geqslant 1} isHeldBy.Principal \sqcap \\
&\exists_{\geqslant 1} isAccountableBy.Principal \sqcap \\
&\forall_{\geqslant 0} asAuthAs.LocalPermission \sqcap \\
&\exists_{\geq 1} hasTarget.Target \sqcap \\
&\exists_{\geq 1} hasAction.Action
\end{aligned}
$$

which states that an individual belongs to the concept *LocalPermission* if and only if it has exactly one *hasNameSpace* relation to the instances of concept *Principal*, has the property relation of *isHeldBy* to at least one instance of concept *Principal*, and has the property relation of *isAccountableBy* to at least one instance of concept *Principal*. Moreover, an instance of concept *LocalPermission* has binary relation of *hasTarget* to at least one instance of concept *Target*; and has binary relation of *hasAction* to at least one instance of concept *Action*. The constraint $\forall_{\geqslant 0} asAuthAs.LocalPermission$ states that all instances that have *asAuthAs* property relation to instances of concept *LocalPermission* must belong to the *LocalPermission* concept.

Concept: Target

The concept *Target* models the target element in a permission specification. Each permission specifies access to a target or a resource. For example, there is a property relation of *hasTarget* from instances of *LocalPermission* to concept *Target*. Concept *Target* has no restriction in its definition in SSAL^O.

Concept: Action

Each permission specification specifies an action that can be performed on a target or resource. The concept *Action* models the action element in a permission specification. Each permission specifies the actions that can be carried out on

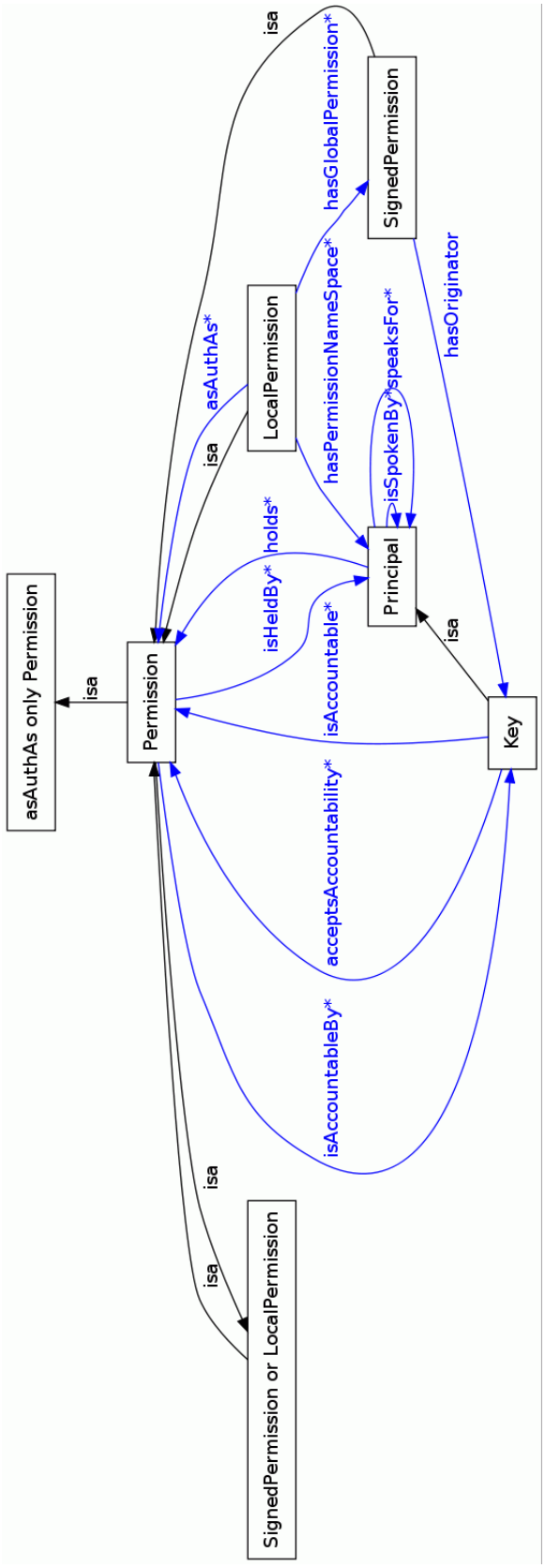


Figure 4.8: An overview on concept LocalPermission in SSAL^O

a target or a resource. There is a property relation of *hasAction* from instances of *LocalPermission* to concept *Target*. Concept *Action* has no restriction in its definition in SSAL^O.

Concept: Delegation

A delegation statement indicates that the authority for a permission is delegated from one principal to the other principal(s). An instance of *Delegation* concept is defined with the following restrictions:

$$\begin{aligned} Delegation \equiv\ & \exists_{=1} hasDelegator.Principal \sqcap \\ & \exists\ hasDelegatee.Principal \sqcap \\ & \exists\ delegatesPermission.LocalPermisssion \end{aligned}$$

These constraints state that an instance of a *Delegation* concept has property relation *hasDelegator* to exactly one instance of concept *Principal*. This means that the delegation statement is signed by only one principal called delegator. Each instance of the *Delegation* concept has a *hasDelegatee* relation to at least one instance of concept *Principal*. A delegation statement may state the delegation of multiple permissions to multiple principals (delegatees). Therefore, an instance of concept *Delegation* has the relation of *delegatesPermission* to at least one instance of concept *LocalPermission*.

Properties in SSAL^O

Properties in SSAL^O are binary relations between instances of the concepts in SSAL^O. SSAL^O includes sixteen object properties: *asAuthAs, delegates-Permission, hasDelegator, hasDelegatee, hasNameSpace, hasOriginator, holds, isHeld, speaksFor, isSpokenBy, isAccountable, isAccountableBy, impliesTarget, impliesAction, hasTarget,* and *hasAction*. The property *isHeldBy* is the inverse of property *holds*, property *speaksFor* is the inverse of property *isSpokenBy*, and the property *isAccountable* is the inverse of property *isAccountableBy*. In addition, it includes a data type (string) property: *hasName*. The domain and range for each property is outlined in Table 4.3 .

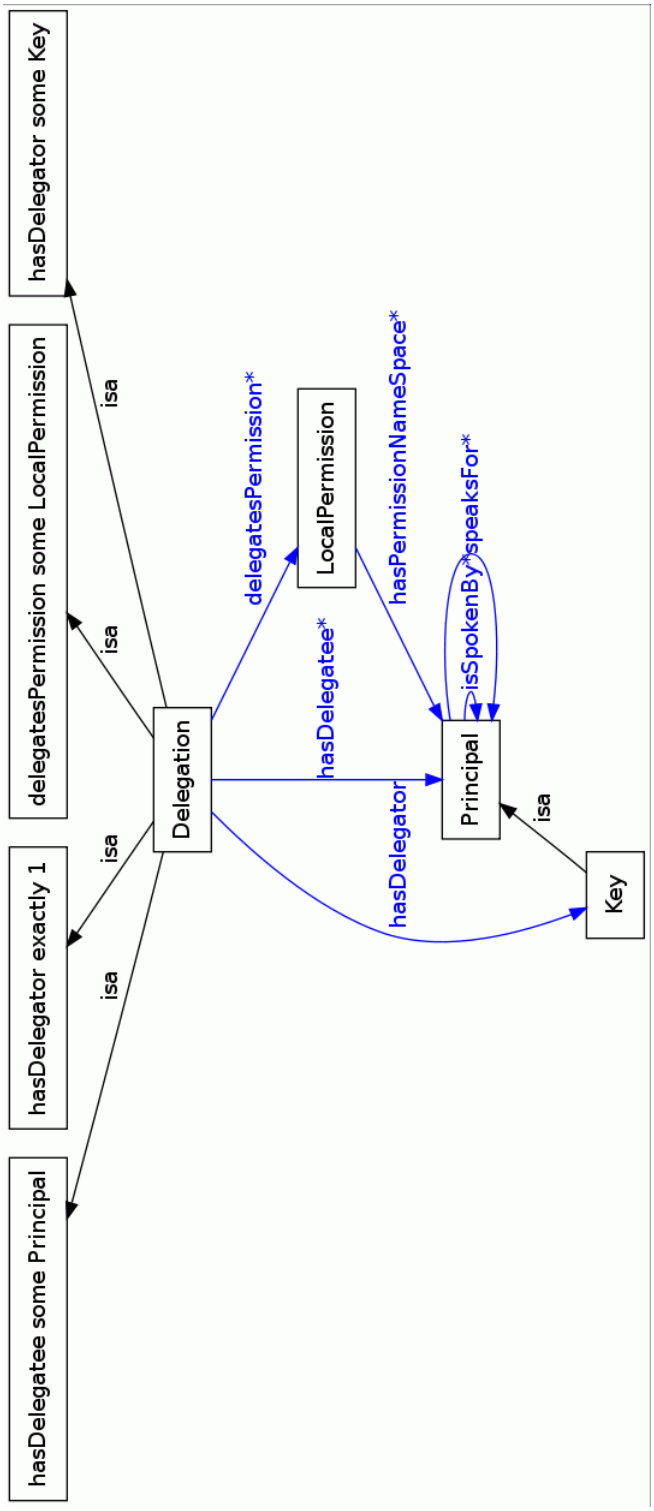


Figure 4.9: An overview on concept Delegation in SSAL^O

Table 4.3: Domain and range of properties in SSAL^O

Property	Domain	Range
asAuthAs	*LocalPermission*	*LocalPermission*
delegatesPermission	*Delegation*	*LocalPermission*
hasDelegatee	*Delegation*	*Principal*
hasDelegator	*Delegation*	*Key*
hasNameSpace	*LocalName*	*Principal*
hasPermission	*LocalPermission*	*Permission*
hasPermissionNameSpace	*LocalPermission*	*Principal*
holds	*Principal*	*LocalPermission*
isAccountable	*Principal*	*LocalPermission*
isAccountableBy	*LocalPermission*	*Principal*
isHeldBy	*LocalPermission*	*Principal*
speaksFor	*Principal*	*Principal*
isSpokenBy	*Principal*	*Principal*

The property names explain their meanings. For example, the postfix property *hasNameSpace*(`ln,p`) is interpreted to mean that, the local name `ln` is in the name space of the principal `p`. In addition, the domain and range for the property *hasNameSpace* indicate that the individual `ln` is an instance of either concept *LocalName* or *LocalPermission*, and the individual `p` is an instance of concept principal. Moreover, table 4.4 demonstrates the terminological axioms and their syntaxes with examples of SSAL^O.

4.2.2 Instantiating ABox with Individuals

The next step is to instantiate the various concepts and thereby populate the knowledge base, in fact the ABox, with individuals. This step includes enumerating the individuals, sorting them, and finally asserting knowledge about each individual.

Table 4.4: Terminological axioms and their syntaxes

Axiom	DL Syntax	Example
Concept Inclusion	$C \sqsubseteq D$	$Key \sqsubseteq Principal$
Concept Equivalence	$C \equiv D$	$Principal \equiv (Key \sqcup LocalName)$
Concept Assertion	$C(i)$	$Principal(k_A Alice)$
Disjoint Concepts	$C \sqsubseteq \neg D$	$Key \sqsubseteq \neg LocalName$
Property assertion	$p(i, j)$	$hasKey(k_A Alice, k_A)$
Property Inversion	$P_1 \equiv P_2^-$	speaksFor $\equiv$ isSpokenBy$^-$
Property Transitivity	$P^+ \sqsubseteq P$	speaksFor $^+ \sqsubseteq$ speaksFor
Disjoin Individuals	$i \neq j$	$k_A \neq k_B$ where $Key(k_A)$, $Key(k_B)$

Notation: C and D are concepts, P is property, i and j are individuals.

Enumerate and Classify Each Individual According to Available Concepts

This step concerns adding concrete data to SSAL^O. For example, adding all local names defined in the SSAL language to the concept of *LocalName*. A single statement such as the following is required for each introduction and classification of an individual:

$$LocalName(\texttt{k}_\texttt{B}\texttt{employees})$$

Relate Individuals via Available Properties in SSAL^O

With the axiomatization in the TBox, it is a relatively straightforward procedure to assert knowledge about individuals. For example, to relate each instance of concept *LocalName* to instances of concept *Principal*, with the *speaksFor* property, we have the following instantiation :

$$speaksFor(\texttt{k}_\texttt{A}\texttt{,k}_\texttt{B}\texttt{employees})$$

SSAL statements are captured as instances in SSAL^O. Note that, an individual is called an instance when it is classified in the appropriate concept and is linked

to other individuals with appropriate properties. The following notation is used for expressing instantiation in ABox. C is concept, $P_k(k = (1..n))$ is a property, and i, j, l are individuals:

- $C(\mathtt{i}) \longleftarrow P_{\mathrm{k}}(\mathtt{i},\mathtt{j}) \sqcap ... \sqcap P_{\mathrm{k}}(\mathtt{i},\mathtt{l})$: This denotes that the individual `i` has the property relation P_k to the individual `j`, has the property relation P_k to the individual `l`, and so on, to be an instance of concept C.

For example, the following delegation statement:

$$k_A \overset{\langle (k_{B1}\ employees) createAccount \rangle}{\Longrightarrow} k_B$$

is captured in SSAL^O by the following instantiation:

$$\begin{aligned} Delegation(\mathtt{delForCrAc}) \longleftarrow & hasDelegator(\mathtt{delForCrAc}, \mathtt{k_A}) \sqcap \\ & delegatesPermission(\mathtt{delForCrAc,createAccount}) \sqcap \\ & hasDelegatee(\mathtt{delForCrAc,k_B}) \end{aligned}$$

(This means that the individual `delForCrAc` has the property relation *hasDelegator* to individual $\mathtt{k_A}$, has the property relation *delegatesPermission* to individual `createAccount`, and has property relation to individual $\mathtt{k_B}$ to be an instance of concept *Delegation*.

$$\begin{aligned} LocalPermission(\mathtt{createAccount}) \longleftarrow & \\ & hasNameSpace(\mathtt{createAccount,k_{B1}employees}) \sqcap \\ & isHeldBy(\mathtt{createAccount,k_{B1}employees}) \sqcap \\ & isAccountableBy(\mathtt{createAccount,k_{B1}employees}) \end{aligned}$$

$$\begin{aligned} LocalName(\mathtt{k_{B1}\ employees}) \longleftarrow & hasNameSpace(\mathtt{k_{B1}\ employees,\ k_{B1}}) \sqcap \\ & isSpokenBy(\mathtt{k_{B1}\ employees,k_A}) \end{aligned}$$

4.2.3 Policy Rules

We use SWRL to represent SSAL rules introduced in chapter 3 and also in [74,75]. The policy statements are represented by individuals and their relations (via properties) with other individuals in SSAL^O. In addition to the asserted knowledge, there is also hidden knowledge in the ontology that can be inferred from the asserted knowledge. To infer that knowledge, a reasoning tool is required to perform inference. Along with SWRL there is a reasoning engine (Jess) [141, 148] that provides reasoning over the asserted knowledge and produces new knowledge regarding the asserted knowledge.

Name Rule

When principal Q identifies a name N in its name space, and Q *speaks for* R then it is inferred that R is the implicit name space of N. A reduction rule is derived in SSAL to reduce local names to principals; which considering $?p$, $?q$, and $?r$ as principals, if principal $?q$ *speaks for* principal $?r$, and $?q$ chooses name $?n$ for principal $?p$, it can be inferred $?p$ is identified as $?n$ in the name space of $?r$. This can be captured in SWRL with the following rule:

$$
\begin{aligned}
Ont-N1: & Principal(?p) \wedge Principal(?q) \wedge Principal(?r) \wedge \\
& speaksFor(?p, ?qn) \wedge LocalName(?qn) \wedge hasNameSpace(?qn, ?q) \wedge \\
& hasName(?qn, ?n) \wedge speaksFor(?q, ?r) \wedge \\
& swrlx : makeOWLIndividual(?rn, ?qn) \\
& \longrightarrow LocalName(?rn) \wedge hasNameSpace(?rn, ?r) \wedge \\
& speaksFor(?p, ?rn) \wedge hasName(?rn, ?n)
\end{aligned}
$$

Permission Delegation Rules

Delegation refers to the act of a principal propagating further the permission that it obtains from other principals. If a principal $?r$ speaks for principal $?q$, any permission $?x$ that is delegated to $?q$ is implicitly delegated to $?r$ (Ont-P1). Moreover, if a principal $?p$ delegates permission $?x$, it implicitly delegates

any permission $?y$ that is dominated by $?x$ (Ont-P2). This is modelled in the following SWRL rules:

$$\begin{aligned} Ont-P1: & Principal(?p) \wedge Principal(?q) \wedge LocalPermission(?x) \wedge \\ & delegatesPermission(?d, ?perm) \wedge hasDelegatee(?d, ?q) \wedge \\ & hasDelegator(?d, ?p) \wedge speaksFor(?r, ?q) \\ & \longrightarrow hasDelegatee(?d, ?r) \\ Ont-P2: & asAuthAs(?x, ?y) \wedge hasDelegator(?d, ?p) \wedge \\ & hasDelegatee(?d, ?q) \wedge delegatesPermission(?d, ?x) \\ & \longrightarrow delegatesPermission(?d, ?y) \end{aligned}$$

Executing the Jess engine has the effect of setting the *delegatesPermission* property as a relation from the individual $?d$ to a $?y$ that satisfies the rule.

Permission Holding Rule

Delegation of a permission does not necessarily imply that the recipient of the permission holds it. Holding a permission depends on either the delegator already originating the permission in its name space or already holding the permission to propagate it further. This prevents malicious principals delegating permissions that they do not hold, and as a consequence are not expected to delegate. A principal by originating a permission asserts that it *holds* that permission. Holding a permission means that a principal is authorized to perform actions based on that permission. The following implements this with a SWRL rule:

$$\begin{aligned} Ont-P3: & Principal(?p) \wedge Principal(?q) \wedge LocalPermission(?x) \\ & \wedge holds(?p, ?x) \wedge delegatesPermission(?d, ?x) \\ & \wedge hasDelegatee(?d, ?q) \wedge hasDelegator(?d, ?p) \\ & \longrightarrow holds(?q, ?x) \end{aligned}$$

Permission Global Ordering Rule

There are reduction rules to reduce localPermissions defined in the name space of a principal, and that principal is identified by a local name. Considering $?p$, $?q$, and $?r$ as principals, and $?pn$ as localPermission, the rule P4 indicates reduction of local permissions. It indicates that, if the permission $?pn$ is issued by principal $?p$ and principal $?q$ *speaks for* $?p$ then the permission $?pn$ is implicitly in the name space of $?q$. Note that the *asAuthAs* property is a transitive relation. This can be captured with the following SWRL rule:

$$\begin{aligned} Ont - P4 : & Principal(?p) \wedge Principal(?q) \wedge LocalPermission(?x) \\ & \wedge LocalPermission(?pn) \wedge hasNameSpace(?pn, ?p) \wedge \\ & speaksFor(?q, ?p) \wedge asAuthAs(?x, ?pn) \\ & \longrightarrow hasNameSpace(?pn, ?q) \end{aligned}$$

The set of policy rules encoded in SWRL are used to reason over the knowledge in SSAL^O and infer new knowledge from the existing knowledge to answer the queries that come to the query engine. We will discuss the query engine later in this chapter.

4.2.4 Integration of Policies within SSAL^O

Different principals define specific kinds of security policies to meet their specific needs. For example, a confidentiality hierarchy is a kind of security policy in some organizations, or access permission hierarchy is another kind of security policy in file systems. These security policies are defined by different principals in different name spaces and they may have their local techniques for implementation. To effectively manage security policies we must be able to produce compatible policy representations. The existence of a large number of representation methods leads to the conclusion that security policies, even when semantically compliant, can be represented in ways that differ substantially in terms of formalism, structure, and hierarchy, thus raising obstacles to their reconciliation. Therefore, for effective management of trust and authorization among distributed principals

one has to be able to integrate all policy representations to make proper access decisions. Each principal defines its own security policy in its name space to control access to its resources, so called a *local policy*. Local policies may have different implementations in each name space. For example, one principal may implement its security policy using XML, another principal may use an ontology, and one may use the Prolog programming language to implement its local security policy. In order to integrate with local policies of other principals for federation, SSAL^O provides a common vocabulary that is understandable by different parties. We assume that each local policy contains a set of permissions that constrain access to the corresponding resources. The set of all permissions SP in a principal's name space may be considered to form a pre-order relation as $(SP, \sqsubseteq) : (sp_i \sqsubseteq sp_j); (sp_i, sp_j \in SP)$. In other words, sp_j implies sp_i, thus a principal that holds the permission sp_j also holds permission sp_i (inferred by reasoning using the rules Ont-P2 and Ont-P3 in SSAL^O). A permission $?x$ for a given resource of principal $?p$ will be represented as the following:

$$LocalPermission(?p) \longleftarrow isHeldBy(?x\ ,\ ?p) \sqcap hasNameSpace(?x\ ,\ ?p)$$

In each local policy, the set of permissions and their ordering relationship is specified locally, therefore the principal who defines the policy must explicitly define a global interpretation for the set of permissions and their ordering relationship. By signing the set of permissions and their ordering a principal provides a global unique interpretation for permissions and consequently prevents subterfuge during open cooperation with other principals. This is modelled in the SSAL^O through OWL individuals and properties where: the set of permissions are considered as instances of concept *LocalPermission*; the name space of these permissions are instances of concept *Principal* that signs the whole set of permissions; and the pre-order relations among the permissions is considered as the *asAuthAs* property. For example, the permission $read$ of a set of permissions $\{read, write\}$, and the ordering relationship $read \sqsubseteq write$ signed by k_A $(\{\!|(read, write, \sqsubseteq)|\!\}_{s_{k_A}})$ will be captured in SSAL^O as the following individual and properties:

$$LocalPermission(k_A read) \longleftarrow isHeldBy(k_A\ read,\ k_A) \sqcap$$

$$hasNameSpace(k_A read,\ k_A) \sqcap asAuthAs(k_A write,\ k_A read)$$

Figure 4.10 depicts the locally defined policy by principal k_A as a fragment of SSAL^O. This way each principal can define its security policy locally, in any

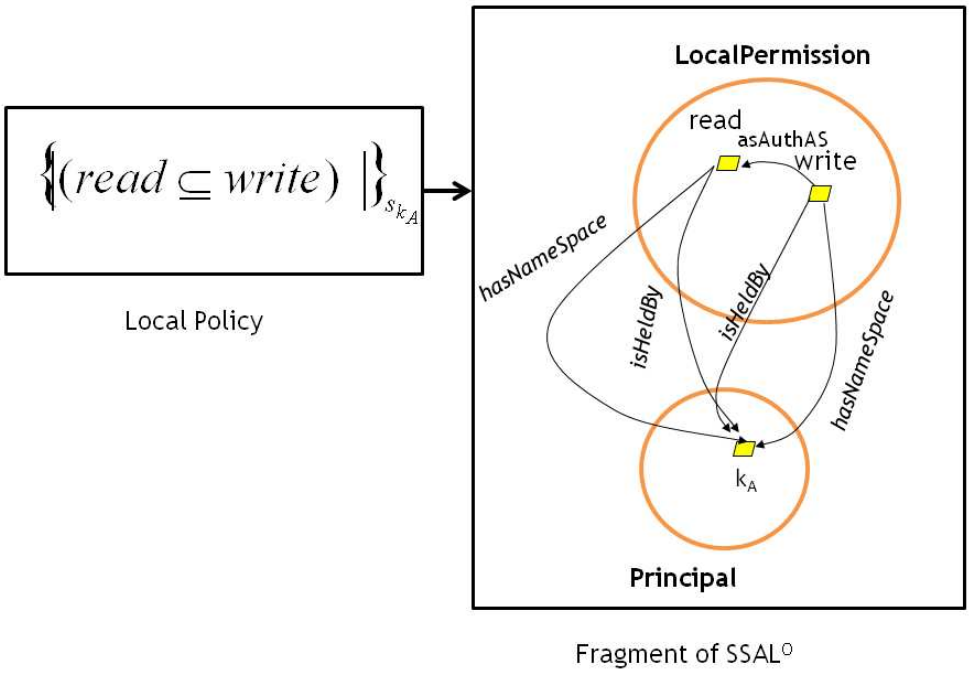


Figure 4.10: Integrating the locally defined policy to SSAL^O

policy language. The signed set of permissions and their pre-order relations are captured as instances of concept *LocalPermission* and the *asAuthAs* property (through an interpreter implemented in Java). These individuals will be part of the knowledge base in SSAL^O and can be reasoned over for making the access decision.

4.2.5 Queries for Trust Management

In addition to reasoning over the knowledge in SSAL^O, a query engine is required for querying over the knowledge to answer an access request. An access request can be evaluated by querying the knowledge in SSAL^O. For example, to verify whether or not a principal k_B is authorized to sell flight number 123 for airline A, it is necessary to verify if the relation *holds*(k_B,(k_A*sellFlightNo.123*)) exists or can be inferred in the ontology. This requires that before any possible request evaluation, all the policy rules (implemented in SWRL) have to be executed to infer all *holds* relations for principal k_B and permission *(*k_A*sellFlightNo.123)*. The following are the queries to evaluate if a request complies with the asserted policy in the knowledge base.

Query for Subterfuge Safe Authorization The following query determines an authorization which checks whether the requester holds the permission to accomplish what it requests.

$$Q1 : Principal(?p) \wedge Principal(?q) \wedge LocalPermission(?perm) \wedge$$
$$holds(?p, ?perm) \wedge isAccountable(?q, ?perm) \longrightarrow sqwrl : select(?p, ?perm, ?q)$$

This query returns all triples of a principal and the permissions that it holds as well as the principal that is accountable for the actions authorized by permission *?perm*.

Query On Subterfuge Safe Delegation The following query determines a subterfuge safe delegation which checks whether it is safe for a delegator to delegate the permission to other principals and that some principal is held accountable for this permission.

$$Q2 : Principal(?p) \wedge Principal(?q) \wedge Principal(?r) \wedge$$
$$Delegation(?delcert) \wedge LocalPermission(?perm) \wedge$$
$$hasDelegator(?delcert, ?p) \wedge hasDelegatee(?delcert, ?q) \wedge$$
$$delegatesPermission(?delcert, ?perm) \wedge isAccountable(?r, ?perm)$$
$$\longrightarrow sqwrl : select(?delcert, ?p, ?perm, ?q, ?r)$$

This query returns all tuples of delegation certificates with their issuer, subject, permission that is delegated to the subject, and the accountable principal for each delegation statement.

A DL reasoner, *Pellet*, is integrated into *Protégé* and performs reasoning tasks in SSAL^O. Thus, we use SSAL^O as a policy engine to make a decision for an access request. A policy engine takes a request, a set of certificates, and policy as input, then outputs a decision for that request [149]. In our model, a requester makes a request to access some protected resources through an Application Programming Interface (API) in the resource owner's trusted environment. The API queries the

trust management engine (SSAL^O) via a query interpreter. The query interpreter which is implemented in the Java programming language, queries the ontology about the request. Note that, the requester may present a set of certificates (encoded using the Security Asserted Markup Language (SAML) [150]) to the resource owner as proof of its authorization. The certificates are in XML data format and are added to the ontology as OWL individuals and their relations via OWL properties. We demonstrate this in the next section. Figure 4.11 depicts the application of SSAL^O as policy engine.

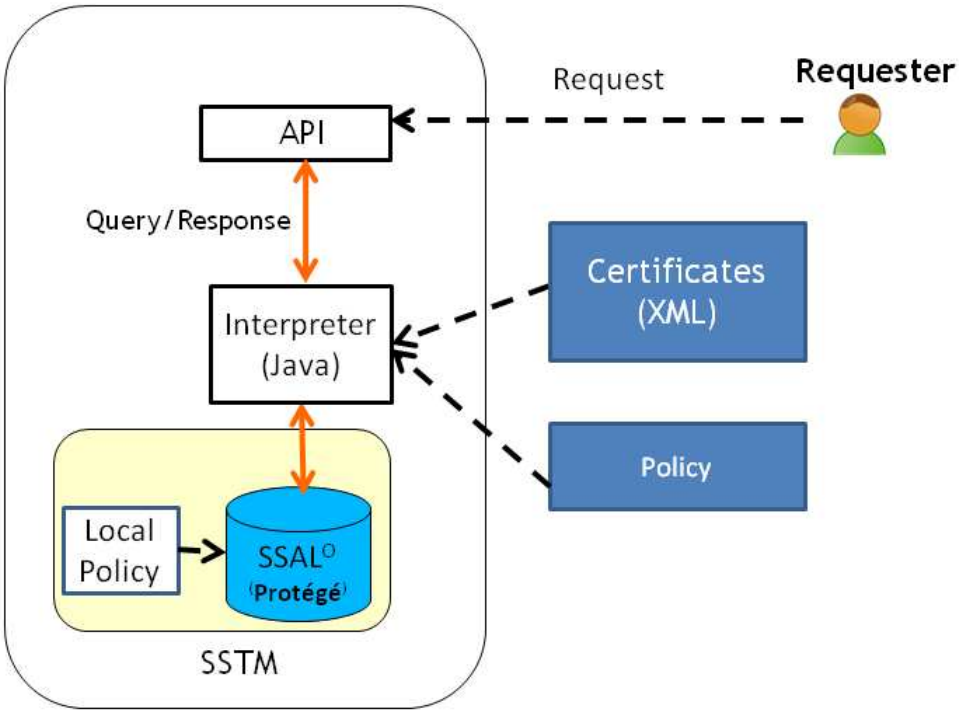


Figure 4.11: Implementation of SSTM

4.2.6 Case study: Trust Management for a Selling Service by Brokers

Consider that the owner of airline A trusts broker B to act as its broker. Brokers are authorized to sell flights. Airline A (the owner of public key k_A) originates (and therefore holds) a localPermission $\langle k_A sellFlightAll \rangle$. Then the airline A adds broker B to the group of brokers identified by the local name (k_A $flightBroker$) (broker B is the owner of public key k_B). The following is the

delegation certificate that airline A issues:

$$k_A \xrightarrow{\langle k_A \ SellFlightAll \rangle} (k_A \ flightBrokers)$$

Figures 4.12, 4.13, and 4.14 are the XML implementation of the above SSAL certificate.

```
<Delegation rdf:about="#delSell">
   <hasDelegator>
     <Key rdf:about="#kA"/>
   </hasDelegator>
   <hasDelegatee>
       <LName rdf:about="#kA_FlightBrokers"/>
   </hasDelegatee>
   <delegatesPermission rdf:resource="#kA_SellFlightAll"/>
</Delegation>
```

Figure 4.12: Delegation certificate

```
<LName rdf:about="#kA_FlightBrokers">
     <isSpokenBy>
          <Key rdf:about="#kB">
              <speaksFor rdf:resource="#kA_FlightBrokers"/>
          </Key>
     </isSpokenBy>
     <hasNameSpace rdf:"#kA"/>
</LName>
```

Figure 4.13: Name certificate

```
<LocalPermission rdf:about="#kA_SellFlightAll">
     <hasNameSpace rdf:resource="#kA"/>
     <isHeldBy rdf:resource="#kA"/>
</LocalPermission>
```

Figure 4.14: Permission certificate

These certificates are captured in the following statements about individuals and

their relationships in SSAL^O:

$$LocalName(k_A \textit{ flight broker}) \longleftarrow$$
$$isSpokenBy(k_A \textit{ flight broker, } k_B) \sqcap$$
$$hasNameSpace(k_A \textit{ flight broker, } k_A) \sqcap$$
$$hasName(k_A \textit{ flight broker, flight broker})$$

$$LocalPermission(sellFlightAll) \longleftarrow$$
$$hasNameSpace(sellFlightAll, k_{\mathrm{B}}) \sqcap$$
$$isHeldBy(sellFlightAll, k_{\mathrm{B}})$$

$$Delegation(delSell) \longleftarrow$$
$$hasDelegator(delSell, k_A) \sqcap$$
$$delegatesPermission(delSell, sellFlightAll\ ) \sqcap$$
$$hasDelegatee(delSell, k_A \textit{ flight broker})$$

The set of policy rules (Ont-P1, Ont-P2, Ont-P3, and Ont-P4) that is encoded in SWRL are used to reason over the asserted knowledge and infer new knowledge for making proper access decision. Therefore, in receiving broker B's request for selling flights, airline A wishes to check whether k_B's request for selling flight at the airline A is authorized or not. The policy engine executes the SWRL rule Ont-P1 to reason over the asserted knowledge within SSAL^O . Thus, the SQWRL query Q1 checks if the following statements can be inferred:

$$isHeldBy(sellFlightAll, k_{\mathrm{B}})$$
$$isAccountableBy(sellFlightAll, k_{\mathrm{A}})$$

As a result of a successful query, broker B's access for selling flights of airline A is granted.

4.2.7 Run-Time Performance

In this section, we present the evaluation of SSAL^O as a policy engine. The objective of these experiments is to evaluate the ability of SSAL^O to integrate a number of security policies defined by different principals within a certain time intervals. We used our implementation of SSAL^O (an OWL-DL model implemented in *Protégé*) and the description logic reasoner *Pellet* [95] to carry out the experiments. We used the Jena Semantic Web Toolkit [97], which supports rule-based inference over the OWL-DL knowledge base. The experiments have been conducted on a Linux workstation with the following hardware configuration: 8Gb RAM with AMD A10-4655M quad core processor. Figures 4.15 and 4.16 shows the results of the experiments. The run time performance of SSAL^O depends on two factors: size of the asserted individuals in the ontology, number of heterogeneous policies for integration.

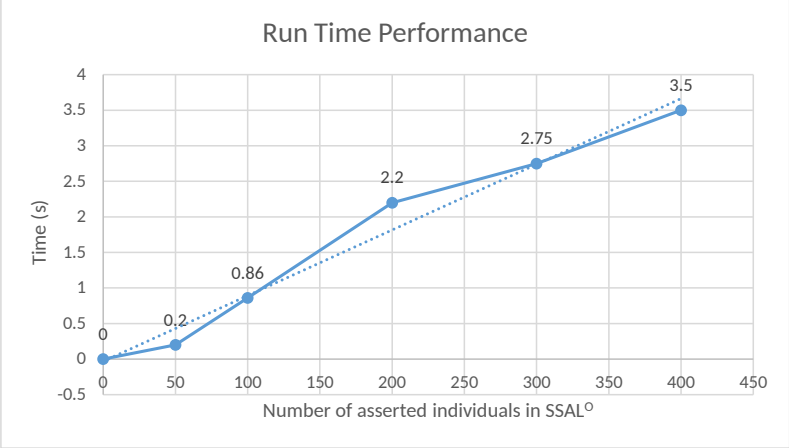


Figure 4.15: Run time performance of reasoning over SSAL^O

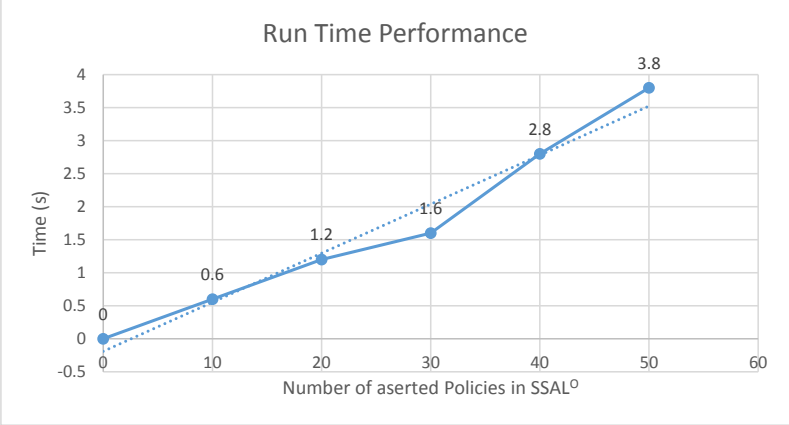


Figure 4.16: Run time performance of reasoning over SSAL^O

This experiment shows that the run-time increases approximately linear when the number of asserted individuals (certificates and policies) increases and that the performance is adequate for most intended scenarios such as web-based applications apart from time-critical ones. It also shows that reasoning based on SSAL^O is a feasible computational task.

4.2.8 Related Work for Solving Heterogeneity Problem

Grid technology was the very first effort to resolve heterogeneity problems in integration of data types such as security policies with different implementations [151]. The Grid infrastructure was a good start for principals to share their data, integrate them and consequently cooperate with each other. The main focus of that effort was resource sharing among virtual organizations for seamless interactivity. Another approach for solving heterogeneity problems for integration of security policies was using a matching scheme. Matching schemes are usually based on XML, the match operator takes two graph-like structures as input and produces matches between their elements. If two structures correspond semantically to each other, matching is successful. The work presented in [152] which performs matches between permission attributes, is a good example of this approach. Within the last few years, there has been an increase of interest in using ontologies for representing access control policies. Ontologies are believed to be important for solving heterogeneity problems in integration of security policies. Ontologies provide a formal specification and common vocabulary for a domain of interest. An example of an ontology approach for integration of policies is *AIR* (Accountability In RDF) proposed by Kagal et al. [153]. They used AIR as an ontology-based language to express the access control policy for sharing data. AIR enables users to share their data in open environments such as web services. However, the integration of multiple policies is based on matching the conditions of a set of rules. The rules also have to be designed in the same language to be able to perform matching on them. Integration and matching are technically difficult especially if multiple policies are defined in different languages with different implementation techniques. The identification of semantic relationships between these policies is also a difficult problem. Thus, an automated dynamic integration solution is required. In SSAL^O, we address not only the automatic integration of heterogeneous policies implemented with different techniques, but also we address the subterfuge safe collaboration of distributed principals after integration of their policies. This feature is unique to SSAL^O. To the best of our

knowledge, no formal framework for integration of heterogeneous policies that can address subterfuge safe cooperation among distributed principals exists.

4.2.9 Discussion

In the context of trust management, an ontology produces a shared understanding of different policies in different domains, represented as a set of concepts, relations, functions, axioms and individuals. There are several reasons for developing our trust management model based on an ontology:

Knowledge Sharing The use of SSAL^O enables different principals to have a common set of concepts about their security policies and interacting with one another.

Knowledge Reuse SSAL^O as a policy engine can be reused by different principals without building a new policy engine from scratch. Reusing ontologies reduces engineering costs since it avoids rebuilding existing ontologies. Moreover, since SSAL^O is understood as a means for sharing knowledge concepts, reusing that increases the secure interoperability between different principals both on the syntactic and on the semantic level. Principals using the same ontology are assumed to hold the same view upon the modelled universe of discourse, and thus define and use domain concepts in the same way.

Scalability The description of a trust management system in a machine-understandable fashion (ontology) is expected to have a great impact in areas of policy integration, as it is expected to enable dynamic and scalable cooperation among different principals of open environments such as web services, organizations, coalitions.

Complexity Choosing OWL-DL provides the possibility of using the OWL-DL reasoner as a policy engine and a query tool. The DL reasoner *Pellet* and the SWRL engine have high complexity (NExpTime-complete) but DL reasoners can handle all features of the OWL-DL language. The DL expressibility of SSAL^O model is *SHOIN(D)*, where S stands for ALC [154] plus role transitivity, H stands for role hierarchy, O stands for nominals, I stands for inverse role, N stands for cardinality restrictions, and D stands for datatypes. However, we did not use

some features such as nominals or role hierarchy in our model. We evaluated the complexity of OWL-DL with its specific features used in SSAL^O via a calculator for complexity of reasoning in description logics [155], and determined it to be NExpTime-complete.

Subterfuge Safe Policy Integration Ontologies are frameworks for organizing structured data. Defining permissions for hierarchical resources is a very common requirement for security policies. The permissions for hierarchical resources can be modelled in the ontology since ontologies provide a set of hierarchical and relational representational primitives. In the SSAL policy language we defined the "*no less Authoritative than*" relation as the global ordering relationship among permissions. This ordering relationship is assumed to be specified explicitly. However, resources in a system have a hierarchical structure and therefore permissions to access those resources form partial ordering relations. For example, airline A may specify permission $\langle k_A\ (sellFlightAll)\rangle$ for selling all flights in its domain and the permission $\langle k_A\ (sellFlightNo123)\rangle$ for selling flight number 123. Consider that there is an explicit ordering relationship among the permissions that A defines in its name space. For instance, there is an ordering relationship $(sellFlightNo123) \sqsubseteq (sellFlightAll)$ in which permission $(sellFlightAll)$ implies permission $(sellFlightNo123)$. Any requester that holds the permission $\langle k_A\ (sellFlightAll)$ implicitly should be able to sell flight number 123. The resource owner must specify the ordering relation among the permissions it defines in its local policies. This relation is modelled in SSAL^O using the OWL property *asAuthAs*. This property (*asAuthAs*) also supports subterfuge safe cooperation among distributed principals when their local policies are integrated with one another.

4.3 Summary

In this chapter, we first gave a background on the ontology technique, OWL-DL, and SWRL. Then we introduced and demonstrated SSAL^O which is used as the policy engine for SSTM. Using an ontology for representing and reasoning over the policies provides a common vocabulary and well understood approach for open environments, where multiple organizations with heterogeneous security policies wish to cooperate. In addition, the OWA ensures reasoning over an existing security policy can be easily extended to include further security policies. The implementation also supports integration of heterogeneous policies (policies specified in different languages which may have different implementation in their issuer's name space) to facilitate trust management in open environments. Multiple SWRL rules may be executed in order to retrieve a set of information to grant or deny the access request. SSAL^O can be potentially used in open systems such as distributed web services, cross coalitions cooperation, and cloud federations [156] for subterfuge safe, and dynamic cooperation. The complexity of reasoning over the knowledge in SSAL^O was evaluated as NP-complete which means that the runtime required to reason over the asserted knowledge increases as the size of the asserted knowledge grows in SSAL^O. Experiments have shown adequate performance for typical non-time critical situations.

Chapter 5

Extending SSTM for Supporting Secure Cross Coalition Cooperation

Sharing resources and information in a secure fashion is a requirement for collaboration by distributed entities. Appropriate access control mechanisms are required in order to manage access to those shared resources. A coalition provides a virtual space for collaborators to share their resources and interact with each other. To participate in a coalition, entities must ensure that their resources are safe from inappropriate access while sharing specific resources with coalition participants. Cross coalition cooperation happens while resources of one coalition are shared with participants of another coalition. In this chapter, we introduce a secure dynamic coalition framework that guarantees subterfuge safe cross coalition cooperation. We add two extra SSAL rules to support secure cross coalition cooperation. This allows participants in one coalition to openly cooperate and share resources in a secure manner with other coalitions. Section 5.1 introduces coalitions, and section 5.2 discusses coalition features for secure open cooperation. Section 5.3 reviews existing coalition frameworks. In section 5.4 we discuss the desirable characteristics in any coalition supporting framework. Section 5.5 outlines the process of formation of a new coalition, the issuing of membership, the two additional SSAL rules for sharing resources of coalition participants, and describes the coalition split and merge processes. The characteristics of our model for secure dynamic coalition cooperation are discussed in section 5.6. Section 5.7 provides an example. Finally, a summary for the chapter is given in section 5.8.

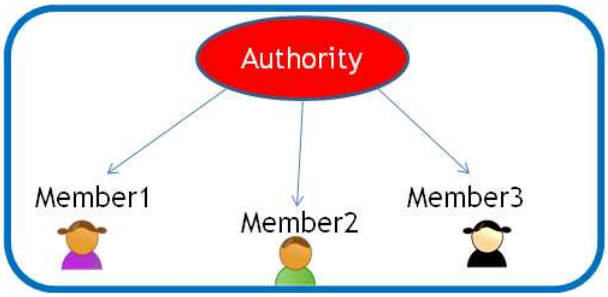


Figure 5.1: A sample of coalition structure

5.1 Coalition Definition

Sharing resources and cooperation among entities are defined in terms of a *coalition*. A coalition provides a virtual space across a distributed environment to allow participants to share resources and cooperate securely for a specific duration of time. The coalition can be formed for simple collaboration among individuals that share resources such as normal sharing of information via email, or can be for more structured and complex collaboration among organizations that share resources for business purposes. For example, a coalition might provide a virtual space for a group of citizens uniting behind a common goal. It also might be an operational structure and regulation for a business-to-business relationship. A group of companies that create a mutual trust between each other can form a coalition in order to increase their profit. For example, *Dunkin' Donuts* and *Baskin-Robbins* formed a coalition by sharing stores and thus sharing revenue [157]. At a system level, a coalition can be used to manage the relationships between its resources. For example, a distributed application may be thought of as forming a coalition between its execution components and the system resources that are available for them to use. A coalition consists of an authority who manages the coalition and coalition participants that provide/use coalition resources and services. A coalition participant can be a resource/service provider, and therefore provides services and resources for other participants of the coalition. A coalition participant can be also a resource/service user, and therefore uses the resources and services that are provided by the provider participants. Figure 5.1 demonstrates an example of a coalition structure.

The ability to securely share resources is critical for service providers when they join a coalition. Before accessing a coalition resource or using a coalition service, a service user should obtain permission from the coalition authority who makes the access decision for that resource or service. The service user sends a

request to the service provider (the coalition participant who provides services and shares its resources with other participants of the coalition). If the service provider believes that the requester is authorized by an authority to use the service/resource, then the request is allowed. Figure 5.2 depicts the request process between a service provider and service user of a coalition with the involvement of a central authority. Participants in a coalition may authenticate each other by asking the question "who said this?" and share resources by asking the authorization question " who is trusted to access this resource?". Usually authentication is accomplished with the involvement of a central authority (CA) and authorization is accomplished by an Access Control List (ACL), in which a set of trusted principals for an action listed [158, 159]. However, authentication and authorization mechanisms alone cannot provide cooperation control among participants of different coalitions. Authentication and authorization mechanisms rely on a super security administrator who is familiar with all available resources and assigns permissions for those resources to the appropriate users in a centralized manner. In centralized environments, both coalition participants and permissions for coalition resources are defined and controlled by the super security administrator. In decentralized environments, each participant in a coalition is familiar with and controls its own resources. The resource owners (principals) decide on their own who is trusted to access their resources. However, principals do not have a complete picture of all trusted principals in coalition cooperation; thus, a framework is required to support cooperation and sharing of resources across coalitions. This is called a *coalition framework*. In coalitions, because of the incomplete view of the system for making access decisions, trust management systems help participants to make appropriate access decisions in decentralized environments and cross coalition cooperation. Figure 5.3 depicts decentralized access control using trust management for coalitions.

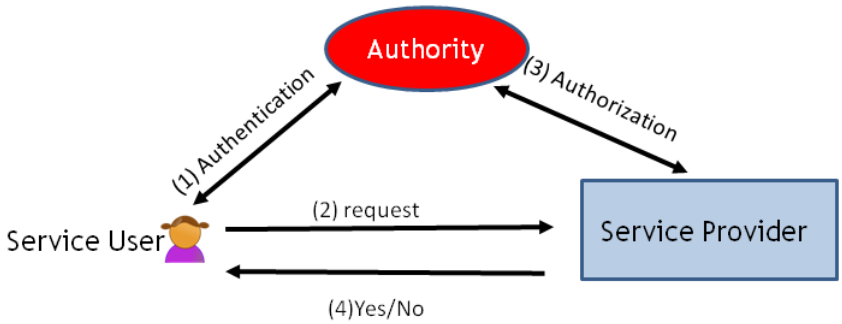


Figure 5.2: Centralized access control for coalitions

5.2 Coalition Features

A secure coalition framework provides a formalized template for membership management, type of administration, type of cooperation, coalition formation and evolution. The formalised template ensures that only the authorized principals can access the shared resources in the coalition. In this section, we outline these security features for coalitions.

5.2.1 Membership Management

Participants that gather together to form a coalition are called members of that coalition. Coalition members are a subset of all principals in the network. A principal must be able to prove its membership of a coalition to use the coalition services and resources. It is also important to distinguish coalition members from other principals in order to assign permissions only to members of a coalition.

5.2.2 Administration

A coalition can rely on a super security administrator (centralized) to control access to the coalition resources. Relying on a super security administrator brings challenges in forming dynamic coalitions. The super security administrator must be appointed and be agreed before coalition formation by all the participants willing to join in the coalition. The super security administrator controls the coalition access control model. Thus, a good access control model relies on the super security administrator's expertise and expertise. Moreover, giving all the power to the super security administrator will bring concerns about possible arbitrary behaviour by the administrator. For example, after a coalition is established and

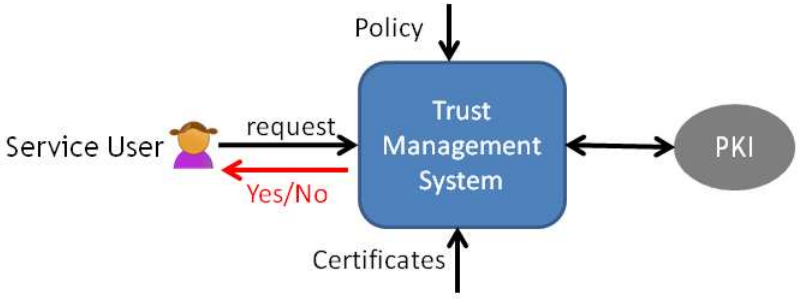


Figure 5.3: Decentralized access control for coalitions

participants agree on sharing resources, the super security administrator, having all powers on the shared resources, may arbitrarily authorize other individuals or organizations outside the coalition to access the shared resources. A coalition can also rely on decentralized security administrators, where two or more administrators (with either the same or different authority) control the appropriate access to the coalition resources. Decentralizing coalition administration among two or more administrators prevents failure in coalition operation when one of the security administrators fails. However, there exists a challenge in having the same authority for all security administrators. An administrator can use its full authority to authorize principals outside of the coalition to access the coalition resources without the agreement of the other coalition administrators. Having multiple coalition administrators with different authority allows principals in decentralized environments to share their resources. Each principal can be viewed as an administrator for sharing its own resources with other coalition members. Loss of an administrator affects only resources that are controlled by that administrator. Thus, the rest of the coalition can work properly.

5.2.3 Subterfuge Safe Open Cooperation

A coalition may define and allow operations only within the coalition, called closed cooperation. For closed cooperation, a coalition does not need any globally unique identifier to represent it in a global environment. Open cooperation relies on the ability of a coalition for subterfuge safe cross coalition delegation to principals outside of a coalition. An open coalition uses a globally unique identifier to represent itself. When cooperating with other coalitions, the members and permissions of an open coalition are bound to its global identifier. Thus, the members and permissions of an open coalition can be recognized. Therefore, an attempt by a malicious principal to access a resource by evading the intended controls of a security mechanism in a coalition is unsuccessful. In order to assign appropriate permissions to only trusted users, a delegation scheme was introduced in SSAL. Permissions that are delegated must have globally unique interpretation to prevent deceptive behaviour that may be accomplished by malicious principals. The following scenario demonstrates the subterfuge problem in cross coalition delegation for open cooperation among coalitions. *Alice*, *Bob*, and *Mary* are members of coalition *Blue*. *Alice* trusts *Bob* and *Bob* trusts *Mary* for selling flights at *Alice*'s company. By transitivity, *Mary* can prove her authorization for selling flights from *Alice*. *Eve*, *Bob*, and *Dave* are members of

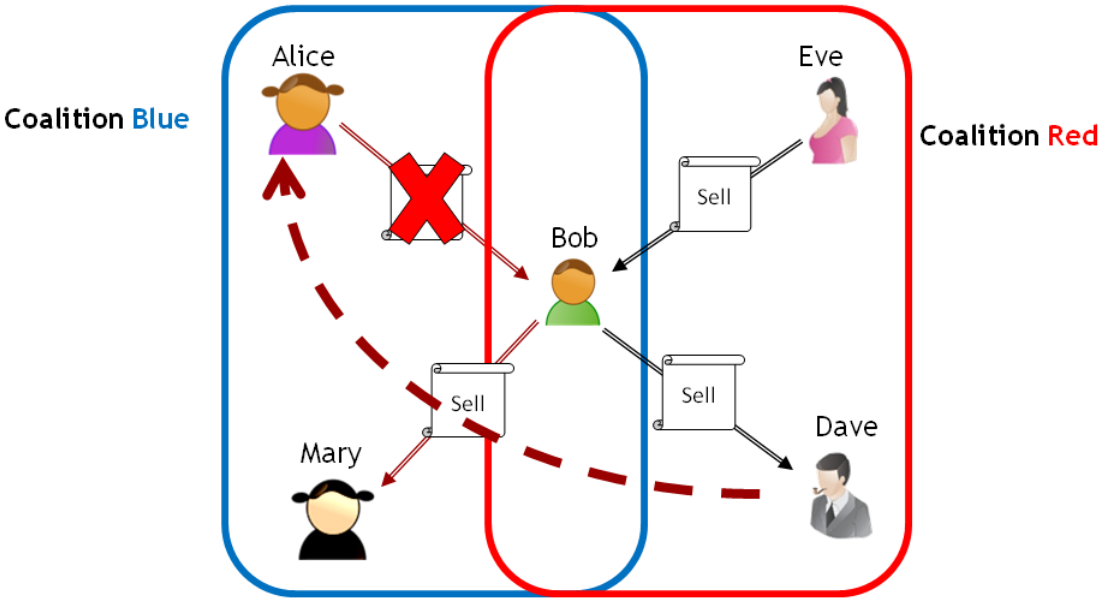


Figure 5.4: Subterfuge in open cooperation of coalitions

coalition *Red*. *Eve* intercepts the certificate for *sell* from *Alice* and delegates *sell* permission to *Bob*. *Bob* thinks that authorization is from *Eve* and delegates it to *Dave*. *Dave* can prove authorization for *sell* from *Alice*. Figure 5.4 illustrates this scenario.

5.2.4 Formation and Evolution

Before forming a coalition, the participants may need to decide whether it would be safe for them to establish the coalition and share their resources. This decision is based on the security mechanisms that a coalition may provide. The security mechanism must determine to what extent the members of a coalition can be trusted. Coalition frameworks are mechanisms that provide templates to establish secure coalitions. The participants may also decide which category of a coalition is desired. For example, the coalition could have a centralized administration or a decentralized administration, and a coalition could have closed cooperation or open cooperation. When the category of a coalition is decided, a suitable coalition framework is used to provide a template for secure formation and evolution of the coalition. Two or more coalitions may come together and merge, or one coalition may split into more than one coalitions.

5.3 Existing Coalition Frameworks

Several coalition frameworks have been proposed to form secure coalitions and their enhancements [40,64,87,160–164]. We investigate these coalition frameworks in terms of the mentioned security features as dynamic membership, type of administration, and subterfuge safe open cooperation.

5.3.1 Systems Research Centre Model

Lampson et.al. [165] introduced a framework to manage group membership at Systems Research Centre (SRC). This framework defines a group where each member speaks on behalf of the group. In order to speak on behalf of a group, a principal must become a member of that group. This will be done by obtaining group membership from the certificate authority of the group. A central authority manages the group membership. The group is represented by a group name and does not have a globally unique identifier. The certificate authority manages the group membership by listing all group members and the group name in a single membership certificate. The membership certificate is signed by the private key of the certificate authority and issued to all group members. None of the group members can admit new members or revoke the membership of the current members by modifying the membership certificate. When a membership certificate is issued, principals are listed in a membership certificate and therefore, principals can prove their membership by presenting the membership certificate. For any principal who receives the group membership list certificate, it can distinguish whether itself (or another principal) is a member of the group or not. Consider a group, named as *friends*, that has *Alice, Bob* and *Mary* as its members. The membership certificate $C1$ is issued by the authority of the group and is held by all members of the group, that is (Note that s_k means that the authority, identified by public key k, issues and signs the certificate):

$$\{|Alice, Bob, Mary|\}_{s_k}$$

In order to join the group *friends*, *Dan* sends a request to the authority. The authority issues certificate $C2$ to all group members, that is:

$$\{|Alice, Bob, Mary, Dan|\}_{s_k}$$

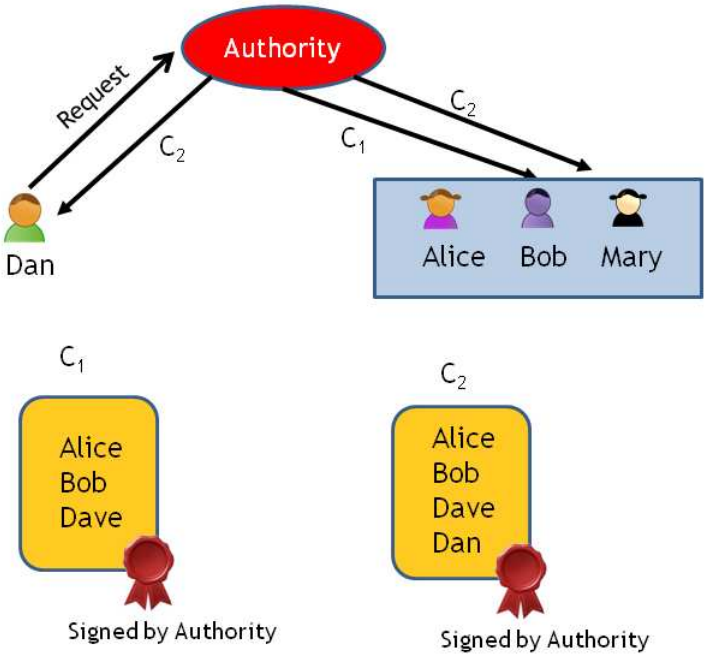


Figure 5.5: The SRC framework

Figure 5.5 demonstrates how *Dan* joins the *friends* group. In this framework, group maintenance would be difficult if the group has a large number of members. Issuing or revoking membership results in having the authority update and re-issue the entire membership list certificate to all related principals. On the other hand, a group is controlled by a single certificate authority. Thus, it is subject to the problem of having single point of failure. Open cooperation is impossible, since it is impossible to distinguish that the message is from the group. The reason for this is that a group is only represented by a group name that may not uniquely identify the group over the open network. Without a globally unique identity, it is impossible to distinguish a given group from other groups. Therefore, cross coalition cooperation is not possible.

5.3.2 Coalition-Based Access Control Model

The Coalition-Based Access Control model (CBAC) was proposed as an access control model for coalitions [163]. CBAC is a family of access control models to capture the semantics of coalition interrelationships and the characteristics of access control in such environments. CBAC_{basic} is the simplest model in the CBAC family, adding coalition entities and relationships to a role-based model. CBAC_{team} builds on CBAC_{basic} to support the collection of users, acting in roles,

into teams. CBAC_{task} builds on CBAC_{basic} to allow access to be determined based on task state. Finally, $\text{CBAC}_{team+task}$ combines the concepts of both teams and tasks to enable team activation based on task state. The administration in CBAC is distributed and is based on delegation of authority. Consider, for example, an organization A that participates in coalition C. Members of organization A have authorization to access coalition resources and the administrator of organization A defines roles and assigns members to roles that operate within the coalition. In defining roles and assigning members to those roles, the administrator delegates authority within its own organization. Therefore, CBAC supports decentralized administration. In CBAC a coalition is not represented by a unique identifier; therefore, open cooperation with other coalitions is impossible since the coalition is not identified in a global manner. The membership management for the coalition is not addressed in CBAC.

5.3.3 Virtual Private Network Model

Virtual Private Network (VPN) [164] was developed for supporting secure cooperation from different physical locations of the same company. Similar to ISC [162], the objective in VPN is to provide secure communication among members of the same coalition and therefore open cooperation was not the purpose of their design. VPN supports symmetric key and public key authentication. The routers or the AAA (authentication, authorization and accounting) servers are the security administrators. They keep all user information about keys and identities. In a VPN, there could be many routers, each assigned to a subnet, making it scalable. However, it focuses on the authentication of individual users to join a coalition, and it does not address cooperation with other coalitions.

5.3.4 Mäki-Aura Model

A model for ad-hoc membership management and later a distributed security architecture for ad-hoc networks were proposed by Mäki and Aura in [160,161]. In this model, a principal, who is the group leader, establishes a group and refers to that group by the group's signature key. For admitting new members, the group leader issues and signs a membership certificate as a verification of membership. The group leader may also issue and sign a leader certificate to specify a group leader with the same authority as the group establisher (the first group leader). When a principal is appointed as a leader, it can use its own signature key to

sign certificates for admitting members and appointing leaders. Also, all group leaders share the same signature key for issuing certificates. Different rights may be delegated to different types of members by defining different roles. In this framework, if a group leader leaves the group for any reason, the group will not be broken. The form of administration is distributed and all leaders have the same authority for managing the coalition resources. However, this framework suffers from the problem that a compromise of the signature key leads to breakdown failure of the entire group. The other disadvantage of this approach is that a malicious principal who obtains the leadership of a group with the same authority of other leaders, can misuse its authority for illegal behaviour.

5.3.5 Internet Services of Coalition Model

A framework for future Internet Services of Coalitions(ISC) was proposed in [162]. The focus of ISC is on collaboration of service providers in provisioning future internet services. The form of administration is distributed as resource sharing decisions are taken by each participant (an individual service provider) in the coalition. It supports dynamic membership where a coalition is formed autonomously by individual service providers. Several service providers may join and leave the coalition at any time. Cross coalition cooperation is not an objective in their framework. We argue that the ISC framework does not support cross coalition cooperation since the coalitions are not identified uniquely in the global environment.

5.3.6 Ellison-Dohrmann Model

A model of access control for mobile computing platforms was proposed in [64] based on SDSI name certificates. The form of administration is decentralized in this model. Because the coalition does not rely on a central authority. The coalition has a leader that admits members and controls all the resources of the coalition. A coalition leader represents the corresponding coalition using a SDSI name certificate. The leader may directly admit members by issuing name certificates that relate its local name for the coalition with the public keys of members. A coalition leader also controls all resources of the coalition. However, this model is vulnerable to the subterfuge problem which limits the usage of this model for open cooperation. A malicious principal E may deceive a principal P that is already a member of group A, to join its group M. A malicious principal

Q intercepts the principal P's group membership certificate C_1 in group A, and principal P may issue a name certificate C_2 to accept Q as a member of group M. However, regardless of P's intention, principal Q can use the set of certificates $\{C_1, C_2\}$ to prove its membership in group A. Thus, this framework does not support subterfuge safe open cooperation.

5.3.7 Distributed Authorization Language Model

A framework for establishing decentralized secure coalitions is proposed based on a role-based Distributed Authorization Language (DAL) [87]. The coalition is formed with the involvement of a constructor, founders, and oversight. The constructor defines the regulation of the coalition and creates the coalition based on those regulations. The founders agree on the coalition regulation that is specified by the constructor. The oversight is a pre-agreed penalty contract by the coalition constructor and all the coalition founders. This framework supports the dynamic establishment of coalitions, forming of further coalitions, and coalition merge. A coalition has a unique identifier that includes its signature key as defined in DAL. The purpose of the coalition key is to sign the initial coalition regulations during the formation of the coalition. The coalition key is generated and initially held by the constructor of the coalition. The constructor is selected by the coalition founders and may be a trusted external third party or prospective member of the coalition. Once the initial coalition regulations that identify the coalition constructor, founders and oversights have been signed and the coalition formed, the coalition key is not used for further signing. Further specified coalition regulations are signed by coalition founders. In forming a coalition, the constructor signs a penalty contract accepting responsibility for the proper use of the signing key. If the key is misused then the constructor becomes liable under the terms of the contract. The coalition regulations are such that it is not possible to establish a coalition without signing this contract. In practice, it is expected that after forming the coalition, the constructor will destroy the coalition key in order to avoid accidental compromise. This coalition framework does not rely on a central authority. Once a coalition is formed, then all the authority of the coalition stands with the founders who can create and regulate their own coalition structure. Moreover, delegation subterfuge can be prevented by associating the originator's public key to permissions. However, this suffers from the challenge of referencing public keys and relies on a globally defined function to define permission relationships. Also, DAL as a basis for a coalition framework, does not

Coalition Frameworks	Administration	Membership Management	Subterfuge Safe Open Cooperation
SRC	Centralized	Yes	No
CBAC	Decentralized	No	Ad-hoc
VPN	Centralized	Yes	No
Maki-Aura Model	Decentralized	Yes	Ad-hoc
ISC	Decentralized	Yes	No
Ellison-Dohrmann	Decentralized	Yes	No
DAL	Decentralized	Yes	Ad-hoc
SSTM-based	Decentralized	Yes	Systematic and Automatic

Figure 5.6: Summary of coalition features of different frameworks

support SDSI extended names as a way of providing indirect referencing among participants in different coalitions for more expressiveness. Figure 5.6 depicts a comparison of the reviewed coalition frameworks and the SSTM-based coalition framework that will be introduced in section 5.5.

5.4 Secure Coalition Characteristics

To support formation and evolution of a secure coalition that is able to cooperate openly with other coalitions we explain that the following characteristics are desirable in a coalition framework.

Globally Unique Identification A secure coalition must have a permanent and unique coalition identity to identify itself in open environments. This is important when a coalition makes a statement, as it must be clear for the principals outside the coalition which coalition the statement is from.

Dynamic Formation The formation and evolution of a coalition must be achieved entirely by its participants. Moreover, coalitions may form further coali-

tions (split). Coalitions may also come together to dynamically form a new coalition (merge).

Dynamic Leadership and Membership Membership and leadership management should not be managed by any central authority. In a dynamic coalition environment, any principal should be able to establish a coalition (following the regulations) and assign a leader for the coalition and admit members.

Subterfuge Safe Cross Coalition Cooperation Permissions that may be delegated from one coalition to another coalition must clearly reference their originator. Therefore, the recipient of a permission in one coalition cannot misuse the permission in another coalition. This way secure cooperation among multiple coalitions is possible.

Accountability A coalition framework must provide a mechanism for tracking responsibility for the permissions that are delegated within the coalition and also across the coalitions. This prevents malicious participants in a coalition from behaviour that results in misuse of coalitions resources while appearing legitimate.

Decentralized Administration The access control model for an open coalition should be decentralized. In other words, the open coalition access control model should not have to rely on a central authority or a centralized authorization server. In a decentralized approach, each resource owner in the coalition should prevent inappropriate access of its resources.

We are not aware of any existing approach that provides an infrastructure that has all of the above characteristics. Using SSTM, we introduce a framework for formation of secure and dynamic coalitions. Coalitions framework using SSTM can be formed in a fully distributed manner without relying on a centralized administration. This framework can be used to split a coalition into further coalitions or allow multiple coalitions to merge.

5.5 SSTM-Based Coalition Framework

Cross coalition delegation is a challenge for open cooperation among coalitions. When two or more participants from different coalitions define the same permis-

sion specification for accessing their resources, they are vulnerable to delegation subterfuge [71]. The same permission specification might be ambiguous and cause confusion for the principal who receives similar permissions. This ambiguity and associated confusion results in delegation subterfuge, in that the principal who receives the permission in one coalition can misuse it to access the resources of another coalition via an indirect and apparently authorized route. Defining two similar permission specifications happens because none of the participants in different coalitions have a complete picture of the name schema for permission specifications. This vulnerability for subterfuge was discussed in detail in chapter 1 and also presented in [71, 75].

Using policy based trust management systems provides a systematic security mechanism for automatic trust decisions regarding the open cooperation of members of different coalitions. This is a more controlled systematic way in contrast to reputation based trust management systems. A variety of trust management systems have been developed over the years to address the requirement for constructing trust and managing authorization without relying on a central authority [9, 10, 43, 46, 60, 65]. They assume unique and unambiguous permission names are provided by a global name provider service. Although global name servers provide a unique interpretation for each name, the principals participating in coalitions may still use arbitrary names to represent their own resources. It depends on the experience of the coalition participant administrator who creates the permissions to specify non ambiguous permissions. However, the design of non ambiguous permissions should not rely on this; it should be formalized in a formal authorization language. Therefore, without a reliable name schema for globally unique permission specification, it is impossible to prevent ambiguity and provide subterfuge safe cross coalition delegation and consequently open cooperation.

SSAL was proposed to support subterfuge safe delegation in large scale distributed systems without relying on a pre-agreed global naming service or super security administrator [75]. A SSTM-based coalition framework uses SSAL as the security policy to provide a secure framework when coalition participants are distributed. The proposed coalition framework provides dynamic membership and subterfuge safe delegation of permission across different coalitions without relying on a super security administrator and global naming scheme. Moreover, it does not have the problem of relying on a globally defined function to define permission ordering relationships, such as existed in [87]. In addition, SSAL supports SDSI's extended names that provides a mechanism for indirect referencing among participants in different coalitions. SSAL can be used as a policy language to construct state-

ments and manage authorization/delegation relationships to automate the decision making process for securely sharing resources among coalition participants. In this section, we describe the process by which new coalitions are formed using SSAL. To build an effective coalition, we must first establish the coalition entities and their relationships to one another. A coalition is identified by a unique coalition identifier and is composed of coalition leaders, coalition members, and a set of all shared resources of all members of the coalition.

5.5.1 Forming a New Coalition

A new coalition is created by generating a fresh (public/private) key pair. The new key pair $(k^C, k^{C^{-1}})$ is called the coalition key. The public key k^C is used as the coalition global unique identifier. The coalition signature key $k^{C^{-1}}$ is used to assign the coalition's leader. The owner of the coalition key is called the leader of this coalition and controls the coalition signature key. In this way, coalitions can be formed dynamically. In some existing frameworks, coalitions may not have a globally unique identifier [165]. If the coalition does not have a unique identifier, then it cannot be identified over the global distributed network. Therefore, when a coalition (spoken for by its leader) makes a statement it is not clear to the principals outside of the coalition that the statement is from the coalition, and thus cross coalition delegation is not possible. In forming a coalition, the coalition leader generates a fresh key pair and signs the leadership statement for itself. This is done by issuing a SSAL name certificate with the coalition signature key as the issuer of the certificate and the leader's global unique identification (either public key or local name) as subject of the certificate. For example, assuming k^c is the coalition public key, and L the leader of the coalition, the following certificate:

$$(k^c \texttt{ leader}) \longrightarrow L_1$$

means that L_1 is the leader of the coalition identified by the coalition public key k^c. Once the coalition is formed and the leader is appointed, the coalition signature key (ephemeral private key) will be destroyed in order to avoid accidental compromise of the coalition key. Then all the authority of the coalition lies with the leader who can admit members and regulate the coalition.

5.5.2 Issuing Membership

In the beginning, the leader is the only member of the coalition. The leader may admit members by issuing SSAL name certificates for principals willing to join any coalition. The certificates are signed with the leader's signature key rather than the coalition signature key. For example, the coalition leader L_1 may admit principal M_1 to join the coalition as the coalition member. The coalition leader L_1 issues the following name certificate to accept M_1 as a coalition member:

$$(L_1 \texttt{ member}) \longrightarrow M_1$$

Note that the leader uses its own key to admit new members. In other words, the coalition leader *speaks for* the coalition. When a coalition leader admits members to the coalition, it passes along all the certificates that prove its own status as a leader in the coalition. In this way, a member can prove its membership to another coalition member or to an outsider by presenting a set of certificates related to its membership. In the above certificate, the member M_1 proves its membership to other members of coalition k^C by presenting the following set of certificates:

$$(k^c \texttt{ leader}) \longrightarrow L_1$$

$$(L_1 \texttt{ member}) \longrightarrow M_1$$

where, the following statement can be inferred (applying the rule introduced in section 3.3.1):

$$(k^c \texttt{ leader member}) \longrightarrow M_1$$

In this way, the membership of M_1 in coalition k^c can be proved.

5.5.3 Local Policy for Cross Coalition Sharing Resources

To effectively participate in dynamic coalitions, participants must be able to share their resources within the coalition as stated in their access control policy. They need to make sure that their resources are safe from access that does not comply with their local policy. This requires each participant to define a security policy for accessing its resources called a *local policy* (LP). It is important that each participant in a coalition has a local policy to govern coalition members in accessing its resources. Each local policy may contain a set of permissions that constraint access to the corresponding resources. The set of all permissions S in

a coalition member's name space may be considered to form a pre-order relation:

$$(S, \sqsubseteq) : (S_i \sqsubseteq S_j); (S_i, S_j \in S)$$

This means if a principal holds the permission S_j it also holds the permission S_i, we say S_j implies S_i. For example, there might be a pre-order relation over the set of permissions including $\{read, write\}$ corresponding to a coalition member's file system resources, as $read \sqsubseteq write$. To avoid ambiguity in permission interpretation when two different coalition partners define the same permission specification for their resources, the SSAL framework provides a localPermission mechanism. A permission x for a given resource of a coalition member identified by either its public key or local name M will be represented as a localPermission $\langle M\ x \rangle$. The set of permissions is created locally and a coalition member must explicitly define how the permission that it originates locally, relates to other permissions globally. Thus, the coalition member M may define the permission global ordering *no less authoritative than* over the set of permissions in its name space. Assuming y as the other permission specification defined locally in the name space of M, where $x \sqsubseteq y$, the coalition member M defines the permission global ordering as $x \rightsquigarrow y$ (x is *no less authoritative than* y). However, it is not effective to define the permission global ordering *no less authoritative than* among permissions of the set of permissions individually forming a pre-order relationship. In other words, it is neither effective nor efficient for a principal that is willing to join a coalition to define localPermissions and their orderings through a multiple signing process. The participant in a coalition should be able to define a set of permissions for its resources and their orderings in its name space, and sign the entire permission set and their ordering in a single signing process to make them globally unique. We add two SSAL rules to transform the local pre-order relation among a set of permissions defined in a coalition member's local policy to the global permission ordering *no less authoritative than*. These rules are defined in the following:

LP_1

A resource owner signs the set of permissions for its resources and the pre-order relation among permissions with its public key in a single signing process. In this way, the resource owner holds any individual permission of that set in its name space. Given a set of permissions S with pre-order relation among the permissions of this set $(S, \sqsubseteq)$, a permission x in S, and public key K of the resource owner,

we define the following rule:

$$\frac{\{\!|(S, \sqsubseteq)|\!\}_{s_K}; x \in S}{K \ni \langle K\ x \rangle}$$

This denotes that a coalition participant defines the set of permission S and their ordering relationship as $(S, \sqsubseteq)$ for its resources. The participant holds each individual permission x by signing the entire permission set and their orderings $(S, \sqsubseteq)$ in a single signing process while willing to share its resources within the coalition.

LP_2

The pre-order relation among a set of permissions in a principal's name space may be transformed to the permission global ordering relation, *no less authoritative than*, in the following rule:

$$\frac{\{\!|(S, \sqsubseteq)|\!\}_{s_K}; x \in S; y \in S; x \sqsubseteq y}{\langle K\ x \rangle \rightsquigarrow \langle K\ y \rangle}$$

This denotes that by signing the entire set S, each permission in the set will have a global unique interpretation, and the permission local ordering relationship among each individual permission specification can be transformed to the permission global ordering relationship *no less authoritative than*. Therefore, in participating in a coalition, a resource owner only signs the set of permissions defined in its local policy and then by applying rules LP_1 and LP_2 the permissions and their ordering will have a global and unique interpretation. This prevents a participant of a coalition signing every single permission in its local policy to provide a global unique interpretation across one coalition and also among different coalitions.

A coalition participant that is willing to share its resources may sign the entire permission set and their orderings over its resources and delegate it to the coalition leader for further delegation (either within the coalition or cross coalitions). In this way (applying rule LP_1 and LP_2) each permission is globally and uniquely referenced by its originator as an accountable principal; so subterfuge will be avoided. Moreover, the coalition leader may accept accountability (signing an accountability statement) for the permissions that it receives from the member of its coalition for further delegation of that permission to other coalitions. Figure 5.7 illustrates a coalition that is formed by leader issuing certificates to members.

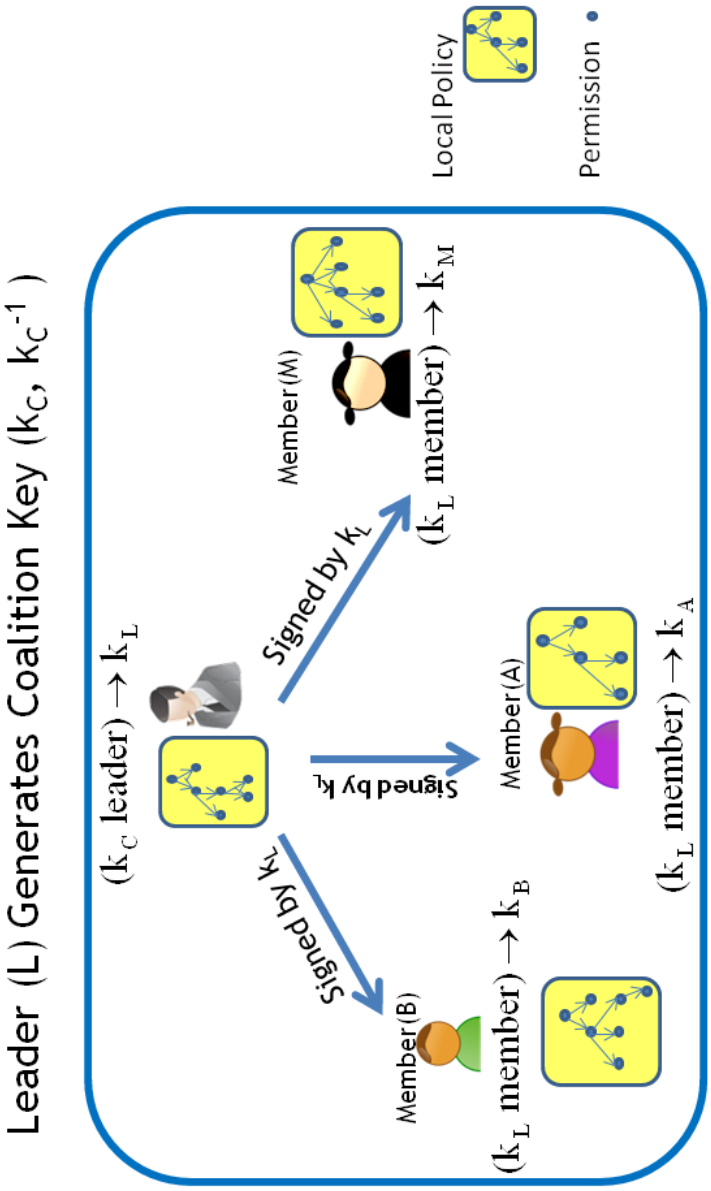


Figure 5.7: A coalition is formed by leader, issuing certificates to members

5.5.4 Coalition Split/Merge

SSTM-based coalition framework allows the dynamic formation of coalitions among participants that are willing to collaborate, as well as supporting arbitrary merge and split to modify a coalition partitioning. In the following sections, we explain how a coalition can be split into further coalitions, as well as merging multiple coalitions to one coalition.

Split

An existing coalition can decide to split into smaller coalitions where any coalition $U_{i=1}^{n}\ C_i$ can be split into the smaller coalitions as $\{C_1, ..., C_n\}$. A participant (including leader and members) of the existing coalition (where the coalition is identified by its public key k^{c_1}) generates a coalition key pair as $(k^{c_2}, k^{c_2^{-1}})$. The split coalition key k^{c_2} can be used to assign a leader for split coalition k^{c_2} and then the leader can admit members. Assigning a leader and admitting members will be done among the participants of coalition k^{c_1}. A coalition split happens when a coalition member prefers to share its resources with a number of coalition participants while still participating in the previous coalition. At this point, a coalition member decides to split the coalition, appointing itself as the leader of the split coalition, admitting members from the subset of current coalition participants, and sharing its desired resources with the participants of the new split coalition.

Merge

Multiple coalitions can merge into a larger coalition where any set of coalitions

$$\{C_1, ..., C_n\}$$

can be merged to one coalition as

$$U_{i=1}^{n}\ C_i$$

In other words, the union of participants of multiple coalitions in order to share their resources is called a coalition merge. Members of merged coalitions are the union of the members of the previous coalitions before merging. The leaders of two or more coalitions may come together and decide to merge their coalitions.

Assume two coalitions with two globally unique identifiers k^{c_1} and k^{c_2} want to merge. L_1 is the leader of k^{c_1} and L_2 is the leader of k^{c_2}, denoted by the following SSAL statements:

$$(k_1^c \texttt{ leader}) \longrightarrow L_1$$

$$(k_2^c \texttt{ leader}) \longrightarrow L_2$$

One of the leaders of these coalitions, issues a membership statement to admit, as its coalition member, the leader of the coalition willing to merge. Therefore, only a membership certificate needs to be issued for admitting the leader of a previous coalition as a member of the new merged coalition. An instance of membership certificate for merging coalitions is as follows:

$$(L_1 \texttt{ member}) \longrightarrow L_2$$

Thus, the certificate $(k_2^c \texttt{ leader}) \longrightarrow L_2$ will be revoked (we do not consider the mechanism for key revocation or certificate revocation in this thesis). The previous membership statements issued by leader L_2 can be used to refer to the membership in the merged coalition. These are a form of extended local names and can be inferred by applying the SSAL rule described in section 3.3.1. M_2 is admitted as a member of coalition k^{c_2} with the following SSAL statements (`employee` is an arbitrary chosen name that L_2 chooses for its members):

$$(k_2^c \texttt{ leader}) \longrightarrow L_2$$

$$(L_2 \texttt{ employee}) \longrightarrow M_2$$

The new membership of M_2 in the coalition k_1^c will be inferred from the above statements by applying rule described in section 3.3.1 to the existing membership statements:

$$(L_1 \texttt{ member}) \longrightarrow L_2$$

$$(L_2 \texttt{ employee}) \longrightarrow M_2$$

Applying SSAL rule 3.3.1, the membership of M_2 in the coalition k_1^c $((k_1^c \texttt{ Leader}) \longrightarrow L_1)$ can be inferred by the following extended name, that is:

$$(L_1 \texttt{ member employee}) \longrightarrow M_2$$

5.6 Discussion

STM-based coalition formation has a number of characteristics that we discuss in the following.

Globally Unique Identification for Coalitions Every coalition is identified by generating a key pair (public /private keys) by its leader. With the public key the coalition will identify itself over the distributed environments. When a coalition (spoken for by its leader) makes a statement, it is clear to the principals outside the coalition which coalition the statement is from.

Dynamic and Autonomous Formation All coalitions, leaders, and members can be viewed as principals which are identified as globally unique by a public key. Then each principal can form a coalition autonomously using the SSTM framework and consequently split/merge the coalitions.

Dynamic Leadership and Membership The leadership and membership process is dynamic. Any principal that generates a key pair can establish the coalition and consequently appoint itself as the leader of the coalition. Members are admitted by the leader (speaking for the coalition).

Subterfuge Safe Cross Coalition Delegation SSAL statements and rules can be used in the coalition framework to provide subterfuge safe cross coalition delegation. Permissions delegated from one coalition to another coalition clearly reference their originator. The recipient of a permission in one coalition, cannot misuse the permission in another coalition.

Accountability Originating a permission creates an accountability for the resource owner. The SSTM framework provides two forms of accountability, accountability by originating a permission, and accountability by issuing a statement regarding the acceptance of accountability by a principal that holds the permission (for example, the leader of a coalition).

Decentralized Access Control The coalition framework using SSAL statements and rules, does not require a super security administrator. The coalition

participants (including leader and members) define the permissions and their orderings in the local policy in order to control access to their resources. The permissions and their orderings in the local policy will be transformed to a globally unique interpretation and global ordering using the SSAL new rules in the framework. In this way, the resources shared in the coalition are under the distributed control of their owners. When a principal makes a request to the resource owner, SSTM can be used by the coalition leader to make the appropriate decision for a particular resource.

5.7 Case Study

In this study, we present about the formation of a coalition, assigning a leader, admitting members, coalition split, and coalition merging.

Forming a coalition *Alice* establishes a coalition called **Org** by generating the public key k^c_{org} as the coalition's global unique identifier. *Alice* is the coalition leader because she generates and consequently owns the coalition signature key. For proving her leadership, *Alice*, where, k_A is *Alice*'s public key, appoints herself as the coalition leader with the following certificate:

$$(k^c_{org}\ \texttt{leader}) \longrightarrow k_A \tag{5.1}$$

Admitting members *Alice* may accept *Mary*, the owner of public key k_M, as a member of coalition **Org** by issuing the following certificate:

$$(k_A\ \texttt{member}) \longrightarrow k_M \tag{5.2}$$

So, *Mary* as her proof of membership in coalition **Org** presents the certificates (5.1), and (5.2).

Sharing resources To allow the participants of a coalition *read* and *write* on her resource $filex$, *Mary* defines the following permission set and ordering in her local policy and signs it as follows:

$$\{|(read, write; read \sqsubseteq write)|\}_{s_{k_M}}$$

Mary delegates her signed set of permissions and orderings to *Alice* for further delegation to the coalition's participants as following:

$$k_M \overset{\{|(read,write;read \sqsubseteq write)|\}_{sk_M}}{\Longrightarrow} (k^c_{org}\ \texttt{leader})$$

Mary's locally defined permission and ordering will have the following globally unique interpretation $\langle k_M\ read\rangle$, $\langle k_M\ write\rangle$, and global unique ordering:

$$\langle k_M\ read\rangle \rightsquigarrow \langle k_M\ write\rangle$$

this prevents confusion and subterfuge in cross coalition delegations. Moreover, it provides a form of accountability for *Mary* for these permissions.

Splitting coalitions *Mary* may decide to split the coalition to form a new coalition **HR** and share part of her resources with specific participants in the coalition. *Mary* generates the key pair $(k^c_{HR}, {k^c_{HR}}^{-1})$, as the coalition **HR** globally unique identifier, and issues the following certificate to appoint the leader of the coalition **HR**:

$$(k^c_{HR}\ \texttt{leader}) \longrightarrow k_M \tag{5.3}$$

$$(k_M\ \texttt{hrMember}) \longrightarrow k_B \tag{5.4}$$

Merging coalitions Consider another coalition $\mathbf{Sale}^c$, identified by public key k^c_{SL}. *Smith*, the owner of public key k_S, establishes this coalition and consequently is the leader of this coalition. *Dave* is a member of this coalition. The following are the certificates that are issued for establishing coalition $\mathbf{Sale}^c$:

$$(k^c_{SL}\ \texttt{leader}) \longrightarrow k_S \tag{5.5}$$

$$(k_S\ \texttt{member}) \longrightarrow k_D \tag{5.6}$$

Mary, the leader of coalition $\mathbf{HR}^c$, and *Smith*, the leader of coalition $\mathbf{Sale}^c$, decide to merge to form a new coalition called $\mathbf{HS}^c$. *Lucy*, the owner of public key k_L,

generates the coalition public key as k^c_{HS} to identify the merged coalition. *Lucy* issues the following leadership certificates:

$$(k^c_{HS}\ \texttt{leader}) \longrightarrow k_L \tag{5.7}$$

$$(k_L\ \texttt{hsMember}) \longrightarrow k_M \tag{5.8}$$

$$(k_L\ \texttt{hsMember}) \longrightarrow k_S \tag{5.9}$$

Dave who was a member of coalition $\mathbf{Sale}^c$, after merging the coalitions $\mathbf{HR}^c$ and $\mathbf{Sale}^c$ to coalition $\mathbf{HS}^c$ can be inferred as a member of coalition $\mathbf{HS}^c$ from statements (5.7) and (5.9) (applying rule 3.3.1) as the following extended name:

$$(k^c_{HS}\ \texttt{leader hsMember}\ ) \longrightarrow k_S \tag{5.10}$$

and from statements (5.6) and (5.10) applying rule 3.3.1 we have the following as the proof of *Dave*'s membership:

$$(k^c_{HS}\ \texttt{leader hsMember member}) \longrightarrow k_D \tag{5.11}$$

Mary and *Smith* delegate the permissions that they hold to *Lucy* as the leader of the merged coalition. For example, *Mary* delegates the set of permissions in her local policy as $(all; \sqsubseteq)$ to *Lucy*, and *Lucy* delegates them further to share *Mary*'s resources with the merged coalition participants.

$$k_M \overset{\{\!|(all;\sqsubseteq)|\!\}_{sk_M}}{\Longrightarrow} (k^c_{HS}\ \texttt{leader})$$

$$k_L \overset{\{\!|(all;\sqsubseteq)|\!\}_{sk_M}}{\Longrightarrow} (k_L\ \texttt{hsMember})$$

5.8 Summary

In this chapter, we have used and extended SSTM as a policy based trust management framework for dynamic formation of coalitions among distributed participants. A coalition may be formed by any principal that generates a key pair to uniquely refer to the coalition in the global network. The principal appoints itself as the leader (consequently the first member) of a coalition and admits members. SSTM focuses on delegating permissions, whereby there is a global permission ordering defined over localPermissions that have globally unique interpretations. The globally unique interpretation provides a form of accountability for the originator of a permission and avoids the subterfuge problem that can occur in cross coalition delegation [71, 75]. This advances the work presented in [87] for subterfuge safe cross coalition delegation for open cooperation. The two new rules allow the permissions and their orderings to be defined locally and delegated globally. The two rules can be used to transform the locally defined permissions and their orderings to a globally unique interpretation and ordering. This chapter addressed how participants in the coalitions can originate, manage, and delegate their own policy rather than relying on a central definition of policy for defining the access control policy for the whole coalition.

Chapter 6

Application of SSTM

In the previous chapters we introduced the subterfuge safe authorization language, SSAL. We also developed SSAL^O, an ontology-based approach for subterfuge safe access control in open environments. Using SSAL^O as the policy engine, SSAL as the policy language, SSTM provides a trust model to build and manage trust and authorization relationships across distributed principals. In this chapter we describe the application of SSTM in two case studies. These case studies demonstrate the use of SSTM for federation of clouds in section 6.1 , and for managing the federation of XMPP instant messaging servers in section 6.2.

6.1 Secure Cloud Federation

Cloud computing is a new paradigm in which applications, data, and IT resources are provided to customers as services in an open manner (over the internet) rather than running locally on the customer's machine. A cloud computing platform enables customers to access these services with a high degree of freedom anywhere and any time. In order to be more beneficial and productive, different cloud computing platforms can share their resources with each other while guaranteeing that each cloud computing platform has enough resources to achieve adequate performance. This can be achieved through cloud federation.

A cloud federation [166] is a collaboration among different cloud computing platforms to share their resources to take advantage of aggregation and produce an enlarged computing utility. A cloud computing platform may share its resources with other cloud computing platforms when it has resources beyond the needs of its own customers. Similarly, a cloud computing platform may request resources

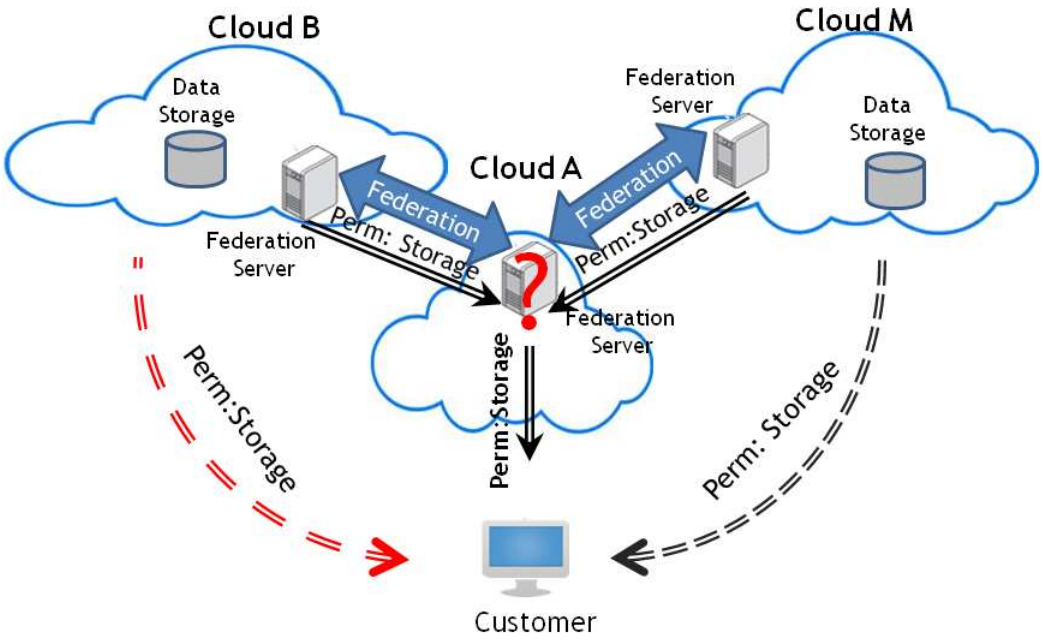


Figure 6.1: Breakdown in accountability for permissions in cloud federation

from another cloud computing platform when its workload cannot be satisfied by its own resources. In this way, both cloud computing platforms benefit because one can obtain more customers, and the other will retain its existing customers.

Despite the potential gains achieved from cloud federation, there are security concerns regarding access to the cloud computing resources [167,168]. Because of the cloud federation characteristics such as large amounts of distributed resources, and lots of distributed customers, a decentralized approach for managing access to cloud computing resources is required. Trust management systems provide a decentralized approach and are suitable to address access control for cloud computing resources when multiple cloud computing platforms establish federation. Permissions are delegated from the owner of resources of one cloud computing platform to the service provider of another cloud computing platform rather than directly controlling the access to cloud computing resources, thereby forming a chain of delegations. A delegation chain provides permission evidence for accessing a resource, and also ensures accountability for the delegated permission. Breakdown in accountability may arise when there is not a unique interpretation for a permission specification. Thus, the delegation chain which is used by service providers to verify their access to cloud computing resources may not reflect the correct accountability for a permission in the chain.

In receiving two identical permission specifications, the delegator may be confused

when inferring the accountability for further delegation. However, permission accountability should not rely on ad-hoc strategies, rather, a systematic way of providing accountability is required. The notion of *localPermission* was introduced to provide a systematic way of providing globally unique interpretation for locally defined permissions [147]. We use localPermissions to achieve robust accountability for delegated permissions in cloud federation. This case study discusses the breakdown in accountability for delegated permissions in cloud federation, and then demonstrates how using SSTM provides breakdown-robust accountability.

6.1.1 Breakdown in Permission Accountability in Cloud Federation

In this section, we introduce an example to explain the ambiguity regarding permission specification that results in breakdown of accountability for that permission when different cloud computing platforms establish a federation. Note that, accountability refers to the tracking of a principal's activity under the permission that the principal holds. The permission might be delegated by another principal who must also be held responsible for the actions activated by that permission. In trust management systems, certificates (cryptographic assertions) specify delegation of permissions among principals. Principals may further delegate their permissions to other principals. A delegation statement indicates that the authority for a permission is delegated from one principal to another principal(s). The delegation statement is denoted as $P \stackrel{X}{\Longrightarrow} Q$, whereby principal P signs a statement that it authorizes principal Q for permission X. A principal is considered to be accountable for a permission, if it accepts responsibility for how the permission is used by other principals. Ambiguity regarding the unique interpretation of a permission can result in confusion of the principal who is considered to be accountable for that permission. From now on, we refer to cloud computing platform with the term "cloud". Suppose that cloud A wants to federate with cloud B and with cloud M, to use the maximum capacity of their data storage resources. Cloud B, identified by its public key k_B, issues a delegation statement enabling cloud A, identified by its public key k_A, to access data storage space at cloud B. This is denoted by the statement:

$$c1 : k_B \stackrel{Storage}{\Longrightarrow} k_A$$

On the other hand, suppose that the malicious cloud M, identified by its public key k_M, intercepts the delegation statement issued by k_B ($c1$), and uses it in the following delegation statement issued by k_M:

$$c2 : k_M \overset{Storage}{\Longrightarrow} k_A$$

Cloud A does not realize that it received the same permission specification (from k_B) and therefore it is led to believe that permission specification $Storage$ is related to accessing data storage resources at cloud M. As a consequence of this confusion, cloud A grants access to the data storage space of cloud M to its customer, identified by public key k_C, denoted as:

$$c3 : k_A \overset{Storage}{\Longrightarrow} k_C$$

This scenario is depicted in Figure 6.1. However, customer k_C, colluding with the malicious cloud M, can use the certificates $c1$ and $c3$ as proof of authorization to access the data storage space of cloud B. Regarding accountability, when cloud A issues the certificate $c3$ (A is considered the accountable principal for delegated permission in $c3$), it believes that the certificate $c2$ provides the correct accountability for cloud M (as a track record of accountability for delegated permissions). However, k_A is confused and should not be held accountable for inadequacy in the permission specification that was specified by cloud B. One may argue that this breakdown in accountability could be prevented by adding extra information about the originator of the permission to the permission specifications. For example, the permission $cloudB/Storage$ is clearly related to its originator. However, a malicious service provider at cloud M, may intercept the certificate:

$$c4 : k_B \overset{cloudB/Storage}{\Longrightarrow} k_A$$

and issue a delegation certificate to delegate permission $cloudB/Storage$ to cloud A, as:

$$c5 : k_M \overset{cloudB/Storage}{\Longrightarrow} k_A$$

Cloud A does not realize that cloud M does not have any authority over cloud B's resources and in further delegation to its customer, k_C, issues the following certificate as :

$$c6 : k_A \overset{cloudB/Storage}{\Longrightarrow} k_C$$

Cloud A claims that it cannot be held accountable for the confusion when k_C uses

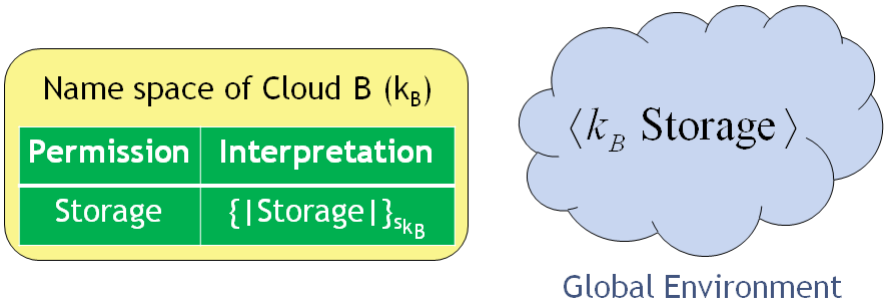


Figure 6.2: Locally defined permission with global unique interpretation

the delegation certificates $c4$ (instead of $c5$) and $c6$ to access the data storage of cloud B.

6.1.2 Accountability for Delegated Permission

Using localPermission prevents the resource owners from issuing ambiguous permissions. The localPermission $\langle P\ Perm\rangle$ represents a permission specification named locally as $Perm$ in the name space of cloud P. P is identified by public key k. The permission $Perm$ signed by P's public key (s_k), $\{|Perm|\}_{s_k}$ is the globally unique reference that corresponds to (refers to) permission $Perm$ in the name space of P. This provides a globally unique interpretation for permission specifications and prevents ambiguity. Therefore, principals such as a cloud administrator receiving two identical permission specifications cannot misuse the permission or get confused and use of them for non-intended purposes. Unambiguous interpretation for each permission makes the originator of each permission accountable for any actions enabled by that permission. This is denoted as $P \rhd \langle P\ Perm\rangle$. For example, by originating a permission for granting access to its data storage space, cloud B (owner of public key k_B) is implicitly accepting accountability for the use of this storage space, i.e. $k_B \rhd \langle k_B\ Storage\rangle$. Figure 6.2 depicts the localPermission of this example. Moreover, in existing trust management systems the set of permissions implicitly have a globally defined pre-order relation. localPermissions are specified locally and an originator must explicitly define how the permissions which are originated locally, relate to other permissions globally. An ordering relation $X \rightsquigarrow Y$ is explicitly defined between permissions X and Y, where permission X is dominated by permission Y. In other words, a principal that is authorized for the resources enabled by permission Y is considered to be authorized for resources enabled by permission X.

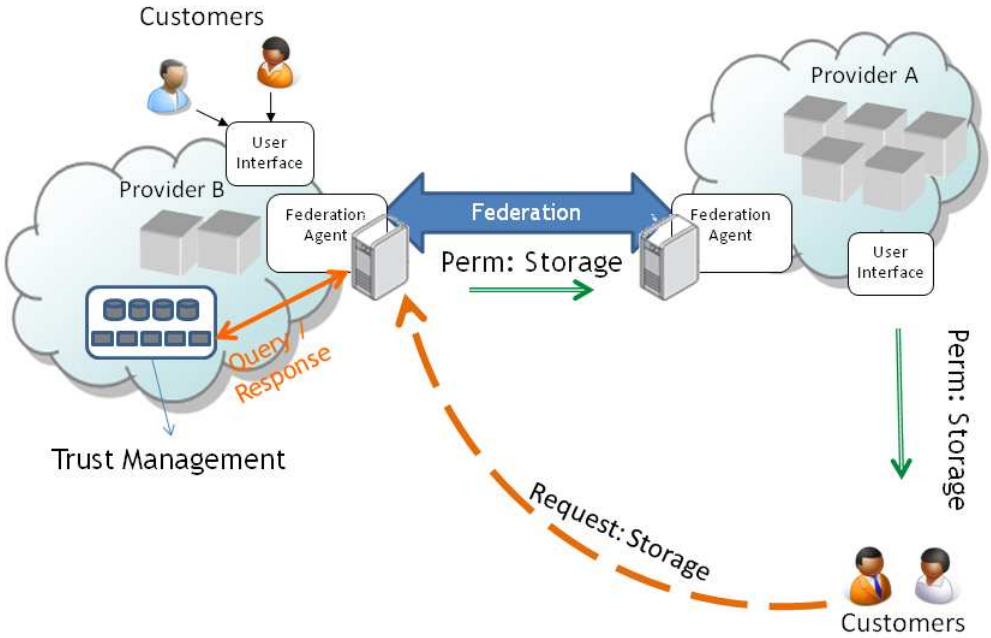


Figure 6.3: Trust management for cloud federation

For example, cloud B originates a localPermission $\langle k_B\ federation\rangle$ authorizing federation with another cloud. Cloud B also asserts that anyone authorized for federation has authority to use its data storage space, denoted as:

$$\langle k_B\ Storage\rangle \rightsquigarrow \langle k_B\ federation\rangle$$

This represents a policy that is local to cloud B but is globally interpretable. A principal Q may accept accountability for the permission $\langle P\ Perm\rangle$ by signing a statement to that effect. However, the principal asserting accountability must be authorized for the permission in the first place. This prevents a malicious principal claiming accountability for a permission (enabling access to a resource) for which it is not trusted. For the example in Figure 6.1, k_M asserts that it accepts accountability for $\langle k_B\ Storage\rangle$, however, k_M is not authorized for $\langle k_B\ Storage\rangle$ and therefore, k_A cannot deduce that k_M is accountable for $\langle k_B\ Storage\rangle$ (denoted as $k_M \rhd \langle k_B\ Storage\rangle$).

6.1.3 Compliance Checking for Accountability

Compliance checking is at the heart of a trust management system [14]. The inputs for a compliance checker are a request, a set of certificates, and a security policy. The compliance checker checks whether a set of certificates proves that

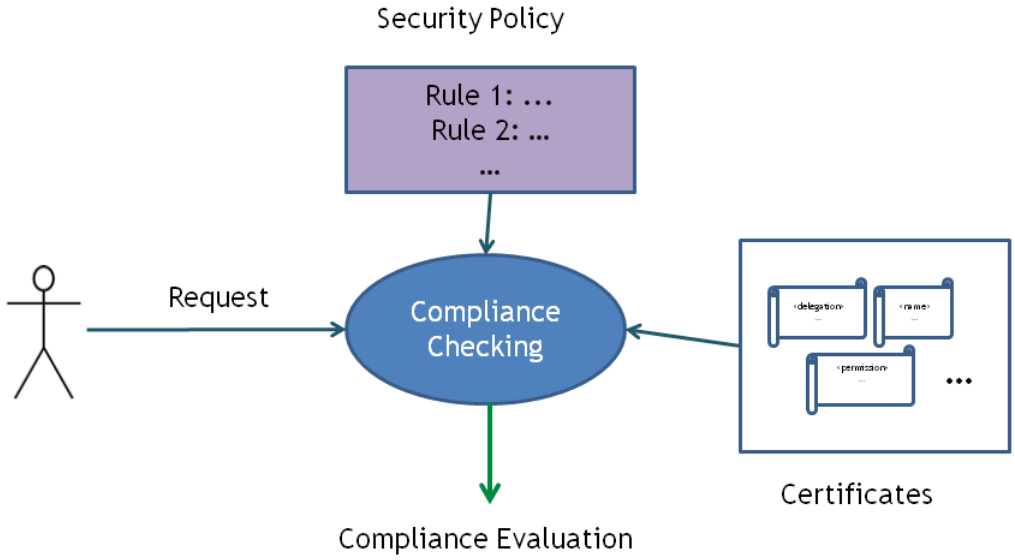


Figure 6.4: Inputs and output of a compliance checking system

a requested action complies with the security policy. Figure 6.4 illustrates the inputs and outputs of a compliance checking system. Compliance checking in current trust management systems corresponds to answering the query (by principal P):

Is requester r authorized to access the resources specified in permission X?

This is evaluated by verifying that the request is supported by a set of certificates that complies with the security policy. Thus, accountability is not addressed directly in current compliance checking mechanisms for trust management. In our approach, if principal P originates permission X then it is deduced that principal P, as the originator of non-ambiguous permission X, is accountable for that permission ($P \rhd X$). However, in the case that P is not the originator of permission X, principal P should determine that some principal R can be held accountable for actions associated with permission X. This needs to be determined before P can sign a delegation statement to delegate permission X to other principals. This is evaluated by determining whether principal R (an earlier principal in the delegation chain) is accountable for permission X, denoted as ($R \rhd X$), and that P trusts R to provide accountability. If the check succeeds then P delegates permission X to other principals. We define two types of compliance checking for determining accountability: one for accountability for authorization to access a resource, and the other for accountability for further delegation of permissions.

6.1.3.1 Authorization Check

Compliance checking for accountability regarding accessing a resource owned by principal P corresponds to the query:

CK1: *Is requester r allowed to access the resource governed by permission X, for which a principal R is accountable?*

This is evaluated by determining whether the requester r can prove (with a set of certificates) that it holds permission X and an accountable principal for permission X can be inferred.

6.1.3.2 Delegation Check

Similarly, compliance checking for accountability regarding further delegation of permission X corresponds to the query (by principal P):

CK2 : *Is a principal R accountable for permission X, that was delegated to principal P?*

This is evaluated by determining whether a principal R is accountable and whether P trusts R to provide accountability. We assume that if a principal is trusted for some permission, then any assertion that the principal makes for accepting accountability for that permission is also trusted.

6.1.4 Managing Cloud Federation Using SSTM

SSTM can be used to manage the trust relationships among different clouds for securely sharing their resources in a federation. Suppose that the service provider for cloud B (identified by public key k_B) in Figure 6.3 uses SSTM for controlling access to its resources. When a request from an untrusted customer is made to access cloud B's resources, SSTM helps the service provider of cloud B in making well-founded access decisions. Suppose all cloud A's resources are in use and cloud A, identified by its public key k_A, is unable to instantiate further resources for its customers. In order to be able to continue providing service to its customers, cloud A decides to federate with cloud B. Thus, the federation agent in cloud A (FA), identified by public key k_{FA}, sends a signed request for federation, including the IP address ranges of cloud A, to the federation agent in

cloud B (FB), identified by public key k_{FB} as in the following:

$$msg1: \ FA \twoheadrightarrow FB : \{federation, IPrange = 192.168.1.1 - 192.168.1.100\}_{s_{k_{FA}}}$$

Note that, all customers for cloud A are in this IP range. In addition, along with the request, FA presents the delegation certificates that it obtained to prove that FA can be trusted for using cloud B's resources:

$$msg2: \ FA \twoheadrightarrow FB : \{k_B \overset{\langle k_B \ federation.IPrange \rangle}{\Longrightarrow} k_{FA}\}_{s_{k_{FA}}}$$

The federation agent for cloud B (FB) confirms the signature on the message $msg1$ from the requester k_{FA} and, if valid, then it queries SSTM as to whether cloud A is trusted to federate with cloud B. As a consequence of a successful query, cloud A federates with cloud B and the resources of cloud B can be shared with cloud A. It is worth noting that the shared resources are then within cloud B. After establishing federation, a customer S (identified by public key k_S) for cloud A sends a signed request including its IP address (which is in the range of IP addresses of cloud A) for using the data storage space of cloud B:

$$msg3 : S \twoheadrightarrow FB : \{Storage, 192.168.1.50\}_{s_{k_S}}$$

In addition, cloud B defines the following local policy that any principal that is authorized for federation in the mentioned IP range is also authorized to access the data storage space (LP_B denote the local policy of cloud B):

$$LP_B : \ \langle k_B \ Storage \rangle \rightsquigarrow \langle k_B \ federation.IPrange \rangle$$

Then cloud A defines its own local policy and originates a permission that explicitly identifies how the permission that it originates is related to the access permission for cloud B's data storage space. Cloud A asserts (LP_A denote the local policy of cloud A):

$$LP_A : \langle k_B \ Storage \rangle \rightsquigarrow \langle k_A \ Storage \rangle$$

and authorizes customer S for the data storage space:

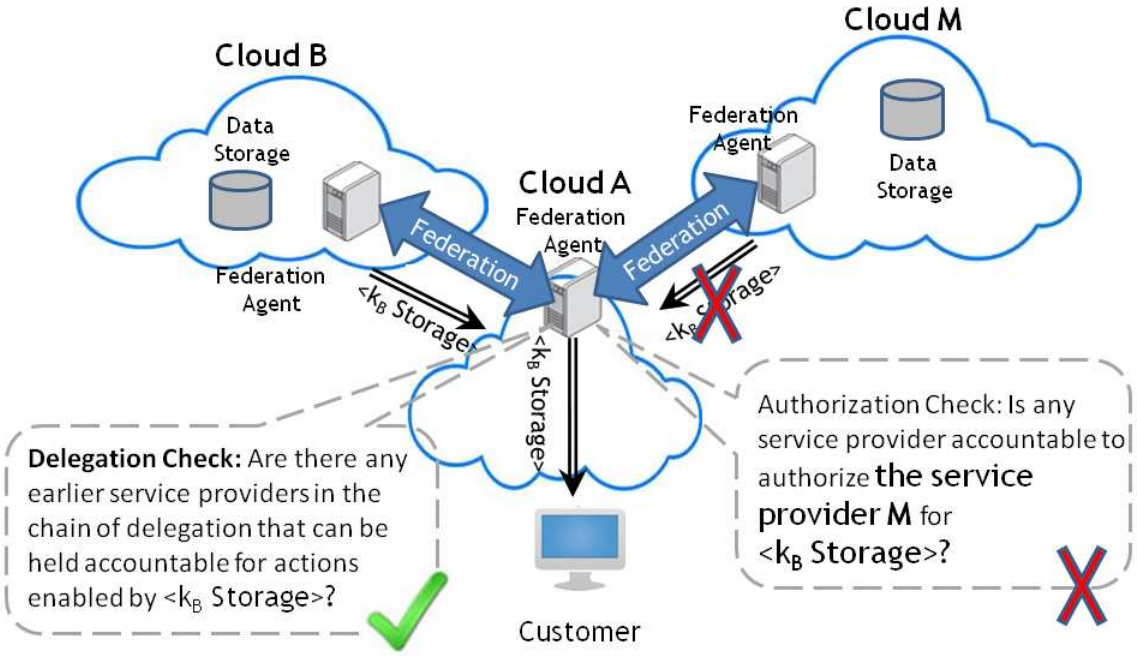


Figure 6.5: Robust accountability for permissions in cloud federation

$$k_A \overset{\langle k_A \ Storage\rangle}{\Longrightarrow} k_S$$

Thus, when customer S requests cloud B's data storage space, it presents a set of certificates (LP_B, and LP_A) that are intended to satisfy the compliance checking. The compliance checking evaluates whether it is safe to allow customer S to access the data storage space at cloud B for which cloud A is held accountable. Returning to Figure 6.1, cloud A checks whether any earlier principal in the chain of delegation can be held accountable for actions authorized by permission $\langle k_B \ Storage\rangle$. Cloud A received this permission from k_B, and k_B is both the originator of this permission and is accountable for it. Therefore, the compliance check for delegation is successful. On the other hand, cloud A might not aware of the statement $\langle k_B \ Storage\rangle \rightsquigarrow \langle k_B \ federation.IPrange\rangle$. Then in the presence of a malicious delegation statement $k_M \overset{\langle k_B \ Storage\rangle}{\Longrightarrow} k_{FA}$, cloud A will not mistakenly think that k_M is accountable. When cloud A searches for an accountable principal, it cannot find any statement that a principal is held accountable to authorize k_M for the permission $\langle k_B \ Storage\rangle$. Therefore, verification of accountability is unsuccessful. This scenario is depicted in Figure 6.5.

6.1.5 Discussion

SSTM provides a systematic security mechanism for automatic trust determination for open federation of clouds. Although a global name service provides a unique interpretation for each name, the cloud administrator may still use arbitrary names to represent permissions for their own cloud resources. The cloud computing administrator who originates the permissions must try to specify non ambiguous permissions for cloud resources in order to provide accountability. However, providing accountability for delegation of permissions should not rely on ad-hoc strategies; it should be formalized in a systematic way. The localPermission provides a reliable name schema for globally unique interpretations for permissions without relying on a central authority. Using localPermissions, none of the service providers have an excuse for confusion caused by inadequate information in permissions specifications. Thus, SSTM provides a reliable scheme for robust accountability for permission delegation in cloud federation.

6.2 Trust Management for Secure Federation of XMPP Servers

6.2.1 Introduction to XMPP Servers

The Extensible Messaging and Presence Protocol (XMPP) is a communication protocol based on XML for sending and receiving messages between distributed entities. The conventional use of XMPP is in instant messaging (IM) and presence such as GoogleTalk. However, it supports a much wider range of other applications such as multi party chat, voice and video calls, and remote computing. The XMPP network consists of XMPP servers, XMPP clients, and XMPP services shown in Figure 6.6. XMPP services are hosted by XMPP servers and offer remote functionality to other XMPP entities connected to the network, for example, to XMPP clients. All traffic is routed through the XMPP servers. When users of different XMPP servers want to exchange messages, the relevant XMPP servers initiate server-to-server connections called XMPP federations. Establishing XMPP server federation requires management of access control to XMPP services. A system administrator who manages the XMPP services controls the appropriate access to those services. This administrator must also deal with requests for federation with new XMPP servers. In practice, XMPP servers federation is non-trivial, time-consuming and vulnerable to the delegation subterfuge.

6.2.2 SSTM for XMPP Servers Federation

SSTM can be used to automate the security administration of XMPP servers involved in a federation. Using SSTM, rather than implementing directly on the access control, the administrator creates a security policy that defines the conditions controlling access to the XMPP services. SSTM provides the policy framework and can be used to manage the trust relationships between the administrator and requesters. The advantage of using SSTM as a trust management framework is that it does not necessarily rely on a centralized authorization/policy service and the policy rules can be distributed across the network without any subterfuge vulnerability in the open federation of XMPP servers. A Federated Autonomic Configuration for Network Access Controls (FACNAC) security agent was introduced in [169] to provide individual XMPP servers with autonomic configuration of end-to-end services. SSTM can be used by the FACNAC agent to

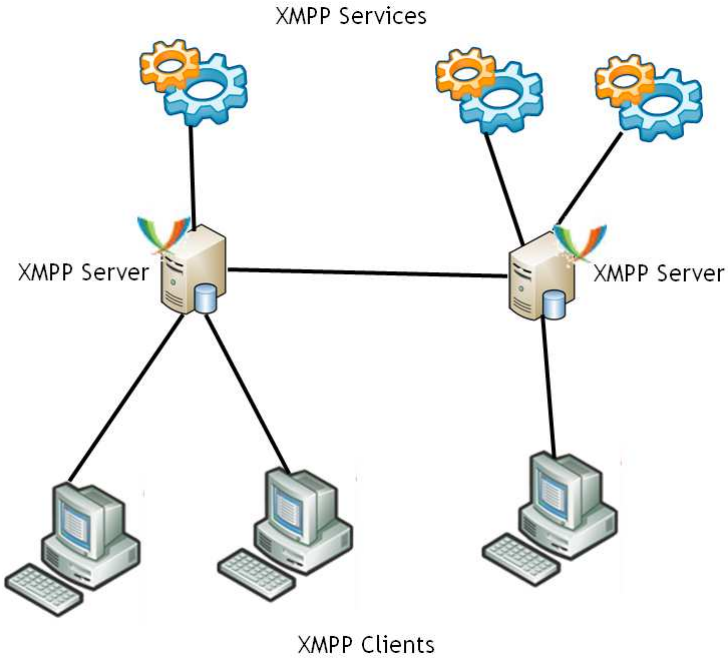


Figure 6.6: XMPP server, client, and service connections

help to manage the trust relationships across the federated servers and to decide when it is safe to federate. FACNAC running on an XMPP server, on behalf of the administrator, accepts configuration change requests and, if the request is permitted by the SSTM policy, then the agent updates the configuration. The request may originate from the administrator or from the user of another XMPP server, requesting federation. The FACNAC agent must determine that the security policy indicates that the requester can be trusted. We use a case study to demonstrate the federation of XMPP servers for instant messaging (IM) services. The trust relationships between FACNAC agents are managed using SSTM.

6.2.3 Case Study

Two organizations *Org1* and *Org2* sign a business agreement and decide to federate their XMPP servers. *Alice* (the owner of public key k_A) is the administrator for *Org1*, and *Bob* (the owner of public key k_B) is the administrator for *Org2*. A trusted *certification authority* CA, identified by public key k_{CA}, issues name certificates for *Alice* and *Bob* as follows:

$$(k_{CA}\ Alice) \longrightarrow k_A \tag{6.1}$$

$$(k_{CA}\ Bob) \longrightarrow k_B \tag{6.2}$$

Alice, the administrator of *Org1*, also acts as the certification authority for *Org1* and issues a name certificate for IM service manager, *Dave* (owner of public key k_D); that is:

$$(k_A\ Dave) \longrightarrow k_D \tag{6.3}$$

Alice defines a role *imManager* for IM services management at *Org1* and adds *Dave* to this role by issuing the following certificate:

$$(k_A\ imManager) \longrightarrow (k_A\ Dave) \tag{6.4}$$

On the other hand, *Bob* defines a group ($k_B\ admins$) that defines his administration staff, and issues a name certificate to add *Clare* (identified by public key k_C) to this group as the following:

$$(k_B\ admins) \longrightarrow k_C \tag{6.5}$$

Dave manages *Org1*'s IM servers and defines permission $\langle k_D\ federation\rangle$. This permission specification indicates that the holder of this permission can federate with *Org1*'s XMPP server. *Dave* is the originator of this permission and it is inferred that he holds this permission denoted as: $k_D \ni \langle k_D\ federation\rangle$. *Dave* also defines a local policy that any principal holding the permission $\langle k_D\ federation\rangle$ is also permitted to federate with its IM servers with IP address i, that is:

$$\forall i : \langle k_D\ fedIPrange.i\rangle \rightsquigarrow \langle k_D\ federation\rangle \tag{6.6}$$

Dave trusts *Alice* (the administrator of *Org1*) to decide with whom to federate and delegates the permission $\langle k_D\ federation\rangle$ to *Alice* by issuing the following delegation certificate:

$$k_D \overset{\langle k_D\ federation\rangle}{\Longrightarrow} k_A \tag{6.7}$$

On the other hand, *Alice*, the administrator of *Org1* wishes the IM servers of *Org1* federate with IM servers of *Org2* and therefore, accepts any email that is signed by users of *Org2*. She originates permission $\langle k_A\ fedOrg1\rangle$ for federation with *Org2*, and permission $\langle k_A\ email\rangle$ for authenticated email services. The organization *Org2*'s IP address range is 192.168.1.∗.

$$\langle k_A\ email\rangle \rightsquigarrow \langle k_A\ fedOrg1\rangle \tag{6.8}$$

Alice trusts the administrator of *Org2*, *Bob*, for these services and delegates the permission $\langle(k_{CA}\ Alice)\ fedOrg1\rangle$ to the administrator of *Org2*, *Bob*, with the following delegation certificate:

$$k_A \overset{\langle(k_{CA}\ Alice)\ fedOrg1\rangle}{\Longrightarrow} (k_{CA}\ Bob) \tag{6.9}$$

and *Bob*, in turn delegates this permission to his administration group by issuing the following certificate:

$$k_B \overset{\langle(k_{CA}\ Alice)\ fedOrg1\rangle}{\Longrightarrow} (k_B\ admins) \tag{6.10}$$

Recall from chapter 4 that these statements are represented as instances within SSAL^O. For example, the delegation statement (6.9):

$$k_A \overset{\langle(k_{CA}\ Alice)\ fedOrg1\rangle}{\Longrightarrow} (k_{CA}\ Bob)$$

is captured in terms of the following instances and their relationships in SSAL^O:

$$\begin{aligned}
LocalName(k_{CA}\ Alice) \longleftarrow\ & \\
& isSpokenBy(k_{CA}\ Alice,\ k_A)\sqcap \\
& hasNameSpace(k_{CA}\ Alice,\ k_{CA})\sqcap \\
& hasName(k_{CA}\ Alice,\ Alice) \\
LocalPermission(fedOrg1) \longleftarrow\ & \\
& hasNameSpace(fedOrg1, k_{\mathrm{CA}}Alice)\sqcap \\
& isHeldBy(fedOrg1, k_{\mathrm{A}}) \\
Delegation(fedDel2) \longleftarrow\ & \\
& hasDelegator(fedDel2, k_A)\sqcap \\
& delegatesPermission(fedDel2,fedOrg1)\sqcap \\
& hasDelegatee(fedDel2,k_{CA}\ Bob)
\end{aligned}$$

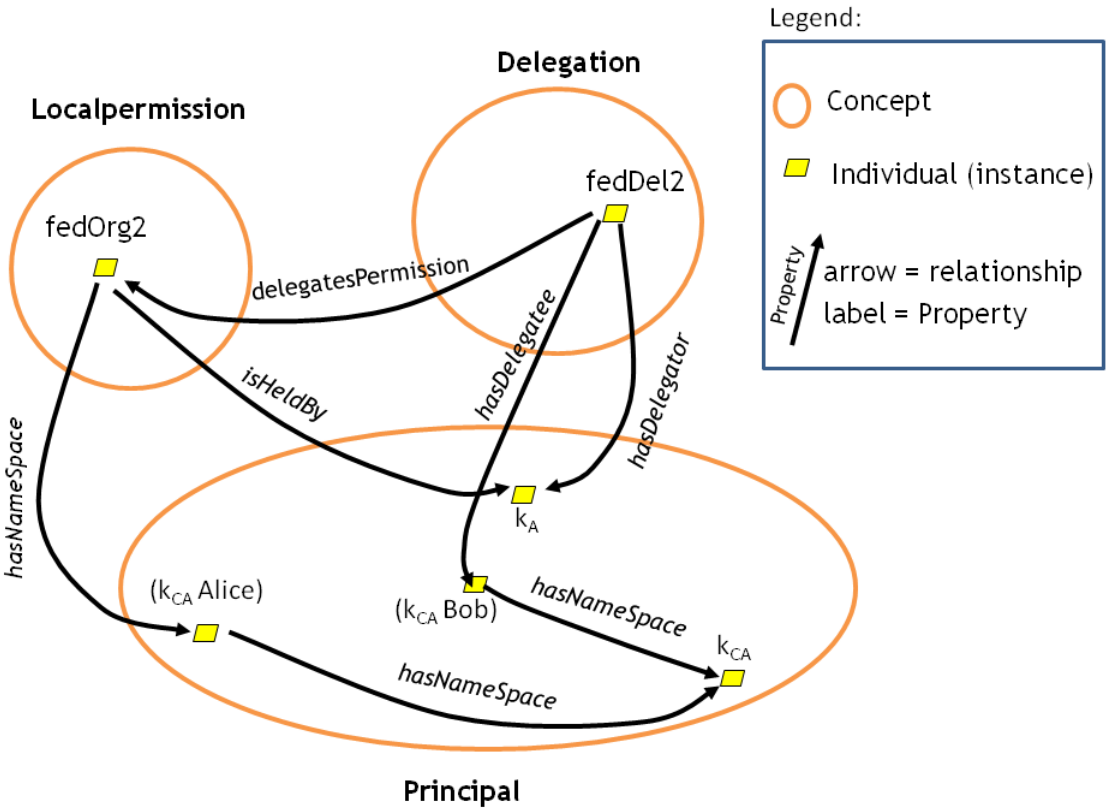


Figure 6.7: Representation of the delegation statement 6.7 in SSAL^O

This fragment of SSAL^O is illustrated in Figure 6.7. The SSAL statements 6.1, 6.2, 6.3, 6.4, and 6.5 are captured in SSAL^O in terms of the following instances and their relationships:

The SSAL^O fragment corresponding to statement 6.1:

$$LocalName(k_{CA}\ Alice) \longleftarrow$$
$$isSpokenBy(k_{CA}\ Alice,\ k_A) \sqcap$$
$$hasNameSpace(k_{CA}\ Alice,\ k_{CA})$$

The SSAL^O fragment corresponding to statement 6.2:

$$LocalName(k_{CA}\ Bob) \longleftarrow$$
$$isSpokenBy(k_{CA}\ Bob,\ k_B) \sqcap$$
$$hasNameSpace(k_{CA}\ Bob,\ k_{CA})$$

The SSAL^O fragment corresponding to statement 6.3:

$$LocalName(k_A\ Dave) \longleftarrow isSpokenBy(k_A\ Dave,\ k_D)\sqcap hasNameSpace(k_A\ Dave,\ k_A)$$

The SSAL^O fragment corresponding to statement 6.4:

$$LocalName(k_A\ imManager) \longleftarrow isSpokenBy(k_A\ imManager,\ k_A\ Dave)\sqcap hasNameSpace(k_A\ imManager,\ k_A)$$

The SSAL^O fragment corresponding to statement 6.5:

$$LocalName(k_B\ admins) \longleftarrow isSpokenBy(k_B\ admins,\ k_C)\sqcap hasNameSpace(k_B\ admins,\ k_B)$$

The SSAL^O fragment corresponding to statement 6.6:

$$LocalPermission(federation) \longleftarrow hasNameSpace(federation, k_{\mathrm{D}})\sqcap isHeldBy(federation, k_{\mathrm{D}})$$

$$LocalPermission(fedIPrangei) \longleftarrow hasNameSpace(fedIPrangei, k_{\mathrm{D}})\sqcap isHeldBy(fedIPrangei, k_{\mathrm{D}})\sqcap AsAuthAs(federation, fedIPrangei)$$

The SSAL^O fragment corresponding to statement 6.7:

$$Delegation(fedDel1) \longleftarrow$$
$$hasDelegator(fedDel1, k_D) \sqcap$$
$$delegatesPermission(fedDel1, federation) \sqcap$$
$$hasDelegatee(fedDel1, k_A)$$

The SSAL^O fragment corresponding to statement 6.8:

$$LocalPermission(fedOrg1) \longleftarrow$$
$$hasNameSpace(fedOrg1, k_{\mathrm{CA}}Alice) \sqcap$$
$$isHeldBy(fedOrg1, k_{\mathrm{A}})$$
$$LocalPermission(email) \longleftarrow$$
$$hasNameSpace(email, k_{\mathrm{A}}) \sqcap$$
$$isHeldBy(email, k_{\mathrm{A}}) \sqcap$$
$$AsAuthAs(fedOrg1, email)$$

The SSAL^O fragment corresponding to statement 6.10:

$$Delegation(fedDel3) \longleftarrow$$
$$hasDelegator(fedDel3, k_B) \sqcap$$
$$delegatesPermission(fedDel3, fedOrg1) \sqcap$$
$$hasDelegatee(fedDel3, k_B\ admins)$$

Having these instances and their relationship in SSAL^O, the set of SWRL rules implementing the axioms of SSAL in chapter 4 are then used to reason over the known facts in SSAL^O and infer new facts. For example, consider the above statements. *Dave* wishes to check whether *Clare*'s request to federate with IM services at IP address 192.168.1.10 is authorized. *Dave* originates the permis-

sion $\langle k_D\ fedIPrange.192.168.1.10\rangle$ and in SSAL^O defines the instances and their relationship as:

$$LocalPermission(fedIPrange192168110) \longleftarrow$$
$$hasNameSpace(fedIPrange192168110, k_{\mathrm{A}}Dave) \sqcap$$
$$isHeldBy(fedIPrange192168110, k_{\mathrm{D}})$$

He then executes the SWRL rules for the SQWRL query to check whether it is possible to infer that $k_C \ni \langle k_D\ fedIPrange.192.168.1.10\rangle$ holds. Assuming the scenario depicted in Figure 6.8, *Dave* manages *Org1*'s IM server with the help of his FACNAC agent. *Clare*, an *Org2* administrator, is responsible for managing *Org2* servers and relies on a FACNAC agent. *Clare*'s FACNAC agent uses public key k_C on behalf of $Clare$ and sends a signed request to federate with *Org2*'s IM server to the relevant FACNAC agent (associated with public key of *Dave* k_D), along with *Org2*'s IP address.

$$\text{Msg1:}k_C \twoheadrightarrow k_D : \{\text{federation request, ip} = 192.168.1.10\}_{s_{k_C}}$$

Org1's FACNAC agent (*Dave*) checks whether the requester holds the authority to federate from IP address 192.168.1.10 with its IM server. *Dave* uses SSAL^O to check whether it is possible to deduce that k_C holds the permission $\langle k_D\ fedIPrange.192.168.1.10\rangle$, denoted as: $k_C \ni \langle k_D\ fedIPrange.192.168.1.10\rangle$.

The following SQWRL query determines whether the requester "$?q$" is authorized by permission "$?x$", for which "$?r$" is accountable.

$$Principal(?q) \wedge Principal(?r) \wedge LocalPermission(?x) \wedge$$
$$holds(?q, ?x) \wedge isAccountable(?r, ?x)$$
$$\longrightarrow sqwrl : select(?q, ?x, ?r)$$

This query is issued by *Dave* to check whether it is possible to deduce that the requester k_C holds $\langle k_D\ fedIPrange.192.168.1.10\rangle$. The result is shown in the

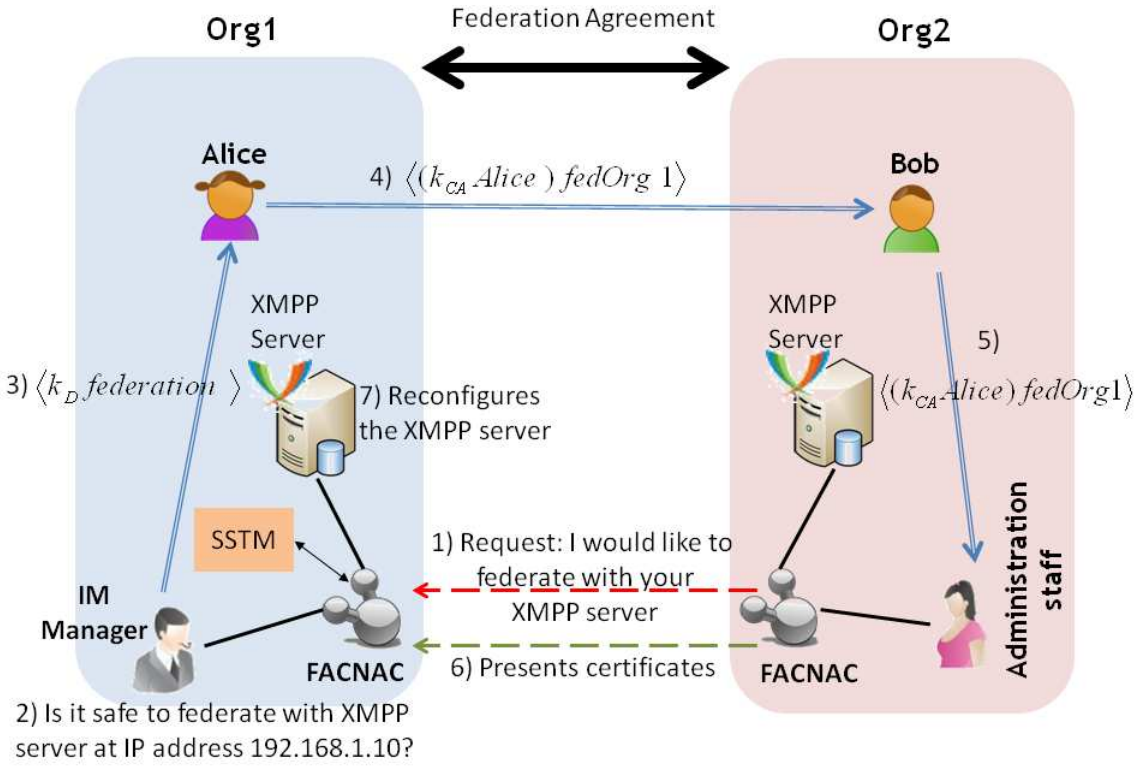


Figure 6.8: FACNAC/SSTM federation scenario

following table.

Requester ?q	Permission ?x	Accountable by ?r
k_D	k_D fedIPrange192168110	k_D
k_C	k_D fedIPrange192168110	k_D

6.2.4 Checking for Subterfuge Safe Delegation

Before a principal signs a delegation statement, it should check whether it leads to subterfuge. In other words, the principal needs to make sure that there is an accountable principal for the actions associated with the permission.

The following SQWRL query determines whether the principal "?r" can be held accountable for permission "?x" for further delegation.

$$Principal(?r) \wedge LocalPermission(?x) \wedge isAccountable(?r, ?x)$$
$$\longrightarrow sqwrl : select(?x, ?r)$$

When *Bob* wants to delegate permission $\langle(k_{CA}\ Alice)\ fedOrg1\rangle$ to his administration staff ($k_B\ admins$), he checks if there is a principal who is accountable for this further delegation. The result of *Bob*'s query is shown in the following table

Permission ?x	Accountable by ?r
fedOrg1	(k_{CA} Alice)
fedOrg1	k_A

The sequences of this scenario is depicted in Figure 6.8.

6.2.4.1 Discussion

Federation of XMPP servers require security controls to provide end-to-end services. A system administrator who manages the XMPP services controls the appropriate access to those services. This administrator must also deal with requests for federation with new XMPP servers. In practice, XMPP servers federation is non-trivial, time-consuming and vulnerable to the delegation subterfuge. In establishing a federation, the agent uses a local knowledge-base, $SSAL^O$, which implements local policy, integration of different policies defined by each XMPP server for access control. Each XMPP server can have its own local FACNAC agent, resulting in the distributed management of trust and authorization relationships. SSTM is used by the FACNAC agents to help managing the distributed access control for secure federation and to help make decision for subterfuge safe federation.

6.3 Summary

In this chapter we demonstrated the applicability of SSTM in two real world examples: cloud federation; federation of XMPP servers. Although cloud federation and federation of XMPP servers offer effective distributed sharing of resources, both require mechanisms to control access and provide accountability for the use of resources. Permissions are delegated among distributed principals to allow each other access to non-local resources. By using unambiguous permission specifications, there is no confusion regarding the accountability for the permissions in sharing resources. The results show that SSTM is a robust model for subterfuge safe delegation of permissions in federation and also provides strong accountability for principals. The success of SSTM for these case studies indicates that it provides a general solution for subterfuge safe management of trust and authorization relationships in open cooperation of entities. It can apply to other examples of open cooperation in open distributed environments.

Chapter 7

Conclusions and Future Work

In this chapter, we draw conclusions, summarise the contributions made in this thesis, and indicate some future research directions.

7.1 Overview

Future distributed applications will be large scale, open, and will often need to deal with complex collaborative interactions. A key necessity for the development of these applications will be a powerful, scalable, flexible and extensible security mechanism. A security mechanism is composed of a sequence of legitimate actions that are performed by a number of principals. It provides a way to ensure that principals will achieve those purposes that are consistent with the security policy. When designing a security mechanism, we would like to ensure that a malicious principal cannot bypass the security mechanism via some legitimate (according to the design) but unexpected behaviour. A malicious principal can be a trojan horse exploiting a covert channel; a spy engaging in a replay attack on a security protocol; an ordinary principal engaging in authorization subterfuge in a trust management scheme.

Many existing security mechanisms are designed in an ad-hoc manner and their design follows best practice based on the expertise of their designer. Expertise-based design only prevents known malicious behaviours. Delegation subterfuge may still occur because a principal receiving a permission in one domain may misuse the permission in another domain via some deceptive but apparently authorized route.

The research question in this thesis was:

> Can we design a well-founded systematic method to avoid subterfuge for cooperation of distributed principals in open environments?

7.2 Summary of Contributions

An overview of access control models, especially trust management systems was presented in this thesis. Trust management provides operations that are used for decentralized access control. As with any protection mechanism the challenge is to make sure that the mechanisms are configured in such a way that they ensure some useful and consistent notions of security.

localPermission We showed how poorly specified permissions within delegation certificates can lead to delegation subterfuge during indirect delegation of permissions. The subterfuge vulnerability results in another vulnerability concerning the accountability of the authorization provided by the delegation chain. The challenge in this thesis was to ensure that permissions have a unique global interpretation. Since permissions are intended to be shared in open environments, then their references must be global. We discussed some ad-hoc strategies to ensure globalization of permissions. However, we showed that the design of a security mechanism should not rely on ad-hoc methods, rather, it should be formalized in a systematic way to prevent subterfuge. The notion of localPermission was introduced for this purpose.

SSAL An authorization language, SSAL, was provided to specify trust-related policies. Using SSAL, a principal may define, without reference to any central authority, its own local permissions, and define a local ordering over the permissions in its name space. In addition, a principal who holds a permission, X, from another name space can assert a global ordering between the permissions in its own name space and permission X. Typical trust management systems make the implicit assumption that there exists a super security administrator that defines the permission name space and its orderings. For example, Distributed Authorization Language prevents subterfuge by restricting delegation to permissions that have an associated originating public key [87]. While effective, in contrast to SSAL this approach suffers the challenge of reliably referencing public keys

and relies on a central authority to define permission ordering relationships.

Distributed Certificate Chain Discovery Algorithm An efficient algorithm was introduced to discover the certificate chain between a resource owner and the requester when certificates are stored distributively. It has been shown that the algorithm returns the chain of delegations if one exists.

SSAL^O An ontology, SSAL^O, was built as the policy engine of SSTM. SSAL^O provides a generic representation for knowledge related to the SSAL-based security policy. SSAL^O enables integration of heterogeneous security policies which is useful for secure cooperation among principals in open environments where each principal may have a different security policy with different implementation. SSAL^O can be used for subterfuge safe and dynamic cooperation in open distributed systems. Example applications include distributed web services, cross coalition cooperation, and cloud federation. The experiments in this study have shown adequate performance for typical non-time critical situations.

SSTM Subterfuge Safe Trust Management (SSTM) was designed to support subterfuge safe delegation of permissions in open environments. SSTM uses localPermissions to provide support for subterfuge safe access control and trust management.

SSTM-Based Coalition Framework SSAL was extended for dynamic formation of a coalition among distributed participants. A coalition may be formed by any principal that generates a key pair to uniquely refer to the coalition in the global network. The principal appoints itself as the leader/first member of the coalition and admits members. The SSTM-based approach (using localPermission) mostly focuses on subterfuge safe cross coalition delegation of permissions. Two additional SSAL rules are used to transform the locally defined permissions and their ordering to a globally unique interpretation and ordering (localPermission). We also addressed how participants in a coalition can originate, manage, and delegate their own policy rather than relying on a central authority to define an access control policy for the whole coalition.

Federation The application of SSTM in two real world federation case studies has been presented. A cloud federation offers effective distributed sharing of

resources but requires mechanisms to control access and provide accountability for the delegated permissions. In the first case study, the application of SSTM for sharing resources in a cloud federation was demonstrated. SSTM provides robust accountability for the use of permissions in a cloud federation. By using localPermissions there is no confusion regarding the accountability for the delegated permissions. The proposed compliance checking provides a means of determining accountability, and therefore can be used in preventing unauthorized access to the cloud resources.

The second case study demonstrates the application of SSTM for managing trust and authorization for federation of XMPP servers. The results showed that SSTM is a robust mechanism for subterfuge safe delegation of permissions for federation of XMPP servers. These case studies is an evidence of using SSTM to provide a robust trust management for subterfuge safe open cooperation of distributed principals.

7.3 Future work

Possible future directions of this work can include extending the theoretical model and improving the implementation. An example of the former is incorporating threshold structures into the model, and an example of the latter is run-time optimization of SSTM authorization queries.

7.3.1 Threshold Structures

A *threshold* structure means that at least K of N number of principals are required to grant a permission or further delegate of that permission. In other words, multiple principals are required to sign a certificate. The SSAL logic does not directly support threshold structures. Without this support, a delegation to a threshold of principals can only be implemented by conjunction of many delegation statements and each delegation statement delegates the same permission to different principals. This kind of complex statement is difficult to implement and manage in practice. As a topic of ongoing research, one can investigate the incorporation of threshold structures into SSTM. There are situations that a request should require several signatures. For example, Amazon Relational Database Service makes it easy to use replication to enhance availability and reliability for production workloads. A replication is a collection of nodes, with one primary

read-write cluster and up to five secondary read-only clusters, which are called read replicas. Applications can read from any cluster in the replication group. More than one replica is required to grant permission to the application which wants to access the data resources if one replica is not available. Incorporating the threshold structures to the model would make this type of applications more manageable in practice.

7.3.2 Run-Time Optimization

To answer a query using SSTM, the interpreter first instantiates the SSAL^O with individuals, then the reasoner builds all the possible relations (either asserted or inferred) among all individuals, and finally the SQWRL runs over the knowledge base and answers the query. When the number of individuals increases in SSAL^O, the time the reasoner consumes to build up the relations and then infer the statements to answer the query increases. For example, to answer the query of the form: "is Alice authorized to read file X?", the query processor queries SSAL^O to retrieve the instances that satisfies this query. Before querying, the knowledge in SSAL^O is loaded into the reasoner. This step ensures the consistency, concept satisfiability, and classification. During the loading phase, axioms about concepts are put into the TBox and assertions about individuals are stored in the ABox. For example, the interpreter interprets *Alice*'s requests $\{\texttt{(readfileX)}\}_{sk_A}$ and sends it to the query processor. The query processor checks whether it can satisfy the following statement in SSAL^O:

$$holds(\texttt{k}_\texttt{A}\texttt{,readfileX)} \wedge isAccountable(?p, \texttt{readfileX})$$

In our experiment, we have evaluated the total time that it takes the reasoner to check the consistency of the knowledge base and the time it takes to answer each query. The experimental results show that this approach works well with a relatively small number of asserted individuals and policies but it takes longer time as the size of the target data set increases. While lack of efficiency may be tolerable for some applications where time is not critical, but when we expect a rather wide-scale application for security policies integration, optimizing run-time for reasoning needs to be investigated as an ongoing research direction.

7.4 Summary

In this chapter we gave a brief overview on the work done in this thesis. The works contributed to the state of the art are: introducing the notion of localPermission; Subterfuge Safe Authorization Language (SSAL); an algorithm for distributed certificate chain discovery; an ontology-based implementation, SSAL^O, as policy engine; Subterfuge Safe Trust Management(SSTM); SSTM-based coalition framework; and the application of SSTM in two real world federation case studies. We produced a well-founded mechanism for dealing with an important problem in distributed open systems. Future research such as those suggested on threshold structures and run-time optimization can build on, and add value to the current work.

References

[1] J. Park and R. Sandhu, "Towards usage control models: Beyond traditional access control," in *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '02. New York, NY, USA: ACM, 2002, pp. 57–64.

[2] B. Thuraisingham, "Mandatory access control," in *Encyclopedia of Database Systems*. Springer, 2009, pp. 1684–1685.

[3] S. Upadhyaya, "Mandatory access control," in *Encyclopedia of Cryptography and Security*. Springer, 2011, pp. 756–758.

[4] N. Li, "Discretionary access control," *Encyclopedia of Cryptography and Security*, pp. 353–356, 2011.

[5] D. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-based access control*. Artech House, 2003.

[6] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," *arXiv preprint arXiv:0903.2171*, 2009.

[7] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[8] S. Jajodia and D. Wijesekera, "Recent advances in access control models," in *Database and Application Security XV*. Springer, 2002, pp. 3–15.

[9] N. Li and J. Mitchell, "RT: a role-based trust-management framework," in *DARPA Information Survivability Conference and Exposition*, vol. 1, 2003, pp. 201–212.

[10] M. Y. Becker, C. Fournet, and A. D. Gordon, "Secpal: Design and semantics of a decentralized authorization language," Technical Report MSR-TR-2006-120, Microsoft Research, Tech. Rep., 2006.

[11] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *Data and applications security and privacy XXVI*. Springer, 2012, pp. 41–55.

[12] D. F. Brewer and M. J. Nash, "The chinese wall security policy," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1989, pp. 206–214.

[13] J. Pieprzyk, T. Hardjono, and J. Seberry, *Fundamentals of computer security*. Springer, 2003.

[14] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance checking in the policymaker trust management system," in *Financial cryptography*. Springer, 1998, pp. 254–274.

[15] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," DTIC Document, Tech. Rep., 1973.

[16] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," DTIC Document, Tech. Rep., 1976.

[17] K. J. Biba, "Integrity considerations for secure computer systems," DTIC Document, Tech. Rep., 1977.

[18] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Proceedings of the 1987 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1987, pp. 184–184.

[19] B. W. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

[20] M. Nyanchama and S. L. Osborn, "Access rights administration in role-based security systems." in *Proceedings of 8th Conference on Data and Applications Security and Privacy (DBSec)*. Citeseer, 1994, pp. 37–56.

[21] M. Nyanchama and S. Osborn, "The role graph model and conflict of interest," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 3–33, 1999.

[22] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn, "A role-based access control model and reference implementation within a corporate intranet," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 34–64, 1999.

[23] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role-based access control: towards a unified standard," in *ACM workshop on Role-based access control*, vol. 2000, 2000.

[24] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[25] D. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-based access control (RBAC): Features and motivations," in *Proceedings of 11th annual computer security application conference*, 1995, pp. 241–48.

[26] R. Sandhu, "Role activation hierarchies," in *Proceedings of the third ACM workshop on Role-based access control*. ACM, 1998, pp. 33–40.

[27] R. M. Needham and A. J. Herbert, *The Cambridge distributed computing system*. Addison-Wesley, 1982.

[28] J. M. Bacon, I. M. Leslie, and R. M. Needham, *Distributed computing with a processor bank*. Springer, 1990.

[29] A. D. Birrell and R. M. Needham, "A universal file server," *Software Engineering, IEEE Transactions on*, no. 5, pp. 450–453, 1980.

[30] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "Hydra: The kernel of a multiprocessor operating system," *Communications of the ACM*, vol. 17, no. 6, pp. 337–345, 1974.

[31] A. S. Tanenbaum, R. Van Renesse, H. Van Staveren, G. J. Sharp, and S. J. Mullender, "Experiences with the amoeba distributed operating system," *Communications of the ACM*, vol. 33, no. 12, pp. 46–63, 1990.

[32] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos authentication and authorization system," in *In Project Athena Technical Plan*. Citeseer, 1987.

[33] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems." in *USENIX Winter*, 1988, pp. 191–202.

[34] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[35] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[36] P. R. Zimmermann, *The official PGP user's guide.* MIT press, 1995.

[37] S. Garfinkel, *PGP: pretty good privacy.* O'Reilly Media Inc., 1995.

[38] C. I'Anson and C. Mitchell, "Security defects in CCITT recommendation X.509–the directory authentication framework," *Computer Communications Review*, vol. 20, no. 2, pp. 30–34, 1990.

[39] D. Cooper, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," 2008. [Online]. Available: https://tools.ietf.org/html/rfc5280

[40] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," 2002.

[41] D. Solo, R. Housley, and W. Ford, "Internet X.509 public key infrastructure certificate and CRL profile," 1999.

[42] S. Farrell and R. Housley, "An internet attribute certificate profile for authorization," 2002. [Online]. Available: http://tools.ietf.org/html/rfc3281

[43] R. L. Rivest and B. Lampson, "SDSI- a simple distributed security infrastructure." Crypto, 1996.

[44] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI certificate theory," IETF RFC 2693, September, Tech. Rep., 1999.

[45] C. M. Ellison, "SPKI requirements." [Online]. Available: http://www.ietf.org/rfc/rfc2692.txt

[46] M. Blaze and A. D. Keromytis, "The KeyNote trust-management system version 2," 1999. [Online]. Available: https://tools.ietf.org/html/rfc2704

[47] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust management for public-key infrastructures," in *Security Protocols.* Springer, 1999, pp. 59–63.

[48] N. Li, B. N. Grosof, and J. Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 1, pp. 128–171, 2003.

[49] A. A. Selcuk, E. Uzun, and M. R. Pariente, "A reputation-based trust management system for p2p networks," in *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 251–258. [Online]. Available: http://dl.acm.org/citation.cfm?id=1111683.1111799

[50] Q. Zhang, T. Yu, and K. Irwin, "A classification scheme for trust functions in reputation-based trust management," in *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*, 2004.

[51] L. Xiong and L. Liu, "PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.

[52] T. H. Noor, Q. Z. Sheng, S. Zeadally, and J. Yu, "Trust management of services in cloud environments: Obstacles and solutions," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 12, 2013.

[53] S. P. Weeks and X. Serret-Avila, "Trust-management systems and methods," Nov. 5 2013, US Patent 8,578,151. [Online]. Available: http://www.google.com/patents/US8104075

[54] V. Shmatikov and C. Talcott, "Reputation-based trust management," *Journal of Computer Security*, vol. 13, no. 1, pp. 167–190, 2005.

[55] S. Sen and N. Sajja, "Robustness of reputation-based trust: Boolean case," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. ACM, 2002, pp. 288–293.

[56] Q. Zhang, T. Yu, and K. Irwin, "A classification scheme for trust functions in reputation-based trust management." in *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*, 2004.

[57] S. Mc Gonigle, Q. Wang, M. Wang, A. Taylor, and E. O. Nuallain, "Reputation-based trust management for distributed spectrum sensing," in *Networks and Communications (NetCom2013)*. Springer, 2014, pp. 255–264.

[58] Q. Pei, B. Yuan, L. Li, and H. Li, "A sensing and etiquette reputation-based trust management for centralized cognitive radio networks," *Neurocomputing*, vol. 101, pp. 129–138, 2013.

[59] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest, "Certificate chain discovery in SPKI/SDSI," *Computer Security*, vol. 9, no. 4, pp. 285–322, 2001.

[60] C. M. Ellison, C. Inc, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen, "SPKI certificate theory," *IETF RFC*, 1999.

[61] J. Wang, J. Li, C. Ma, and J. Peng, "Research on key-based trust model," in *10th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. IEEE, 2013, pp. 153–157.

[62] B. L. Fox and B. A. Lamacchia, "Intelligent trust management method and system," Jan. 15 2013, US Patent 8,355,970.

[63] C. J. Creed and D. J. Rosenblum, "Method, system, and apparatus for facilitating trust certificate management/exchange," Jul. 8 2014, US Patent 8,775,285.

[64] C. Ellison and S. Dohrmann, "Public-key support for group collaboration," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 547–565, Nov. 2003. [Online]. Available: http://doi.acm.org/10.1145/950191.950195

[65] N. Li, J. Mitchell, and W. Winsborough, "Design of a role-based trust-management framework," in *2002 IEEE Symposium on Security and Privacy*, 2002, pp. 114–130.

[66] P. Z. Revesz, *Introduction to constraint databases*. Springer, 2002, vol. 393.

[67] G. Kuper, L. Libkin, and J. Paredaens, *Constraint databases*. Springer, 2000.

[68] N. Li and J. C. Mitchell, "Datalog with constraints: A foundation for trust management languages," in *Practical Aspects of Declarative Languages*. Springer, 2003, pp. 58–73.

[69] N. Li, B. Grosof, and J. Feigenbaum, "A practically implementable and tractable delegation logic," in *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2000, pp. 27–42.

[70] M. Y. Becker, C. Fournet, and A. D. Gordon, "Secpal: Design and semantics of a decentralized authorization language," *Journal of Computer Security*, vol. 18, no. 4, pp. 619–665, 2010.

[71] H. Zhou and S. N. Foley, "A logic for analysing subterfuge in delegation chains," in *Formal Aspects in Security and Trust*. Springer, 2006, pp. 127–141.

[72] S. Foley, "Authorisation subterfuge by delegation in decentralised networks," in *Security Protocols*. Springer, 2007, pp. 103–111.

[73] (1999, May) Internet Assigned Numbers Authority. Internet corporation for assigned names and numbers. [Online]. Available: http://www.iana.org/

[74] S. N. Foley and S. Abdi, "Avoiding delegation subterfuge using linked local permission names," in *Proceedings of the 8th international conference on Formal Aspects of Security and Trust*. Springer-Verlag, 2011, pp. 100–114.

[75] ——, "Avoiding delegation subterfuge using linked local permission names," *Formal Aspects of Security and Trust*, pp. 100–114, 2012.

[76] T. Ylonen and C. Lonvick. (2006) The secure shell (SSH) protocol architecture. [Online]. Available: https://tools.ietf.org/html/rfc4251

[77] S. Farrell, C. T. de Laat, P. R. Calhoun, and G. M. Gross, "AAA authorization requirements," 2000. [Online]. Available: http://tools.ietf.org/html/rfc2906

[78] L. Zhang and H. Zhang. (2012) Compromised-key digest signature (CKDS) introduction and requirement. [Online]. Available: https://tools.ietf.org/html/draft-haikuo-ckds-01

[79] R. J. Perlman, "Ephemeral decryptability," 2002, US Patent 6,363,480.

[80] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security*. Springer, 2007, pp. 1–16.

[81] A. Menezes and B. Ustaoglu, "On reusing ephemeral keys in diffie-hellman key agreement protocols," *International Journal of Applied Cryptography*, vol. 2, no. 2, pp. 154–158, 2010.

[82] S. Jha and T. Reps, "Analysis of SPKI/SDSI certificates using model checking," in *Proceedings of 15th IEEE Computer Security Foundations Workshop*. IEEE, 2002, pp. 129–144.

[83] J.-E. Elien, "Certificate discovery using SPKI/SDSI 2.0 certificates," Masters Thesis, MIT LCS, Tech. Rep., 1998.

[84] N. Li, W. H. Winsborough, and J. C. Mitchell, "Distributed credential chain discovery in trust management," *Journal of Computer Security*, vol. 11, no. 1, pp. 35–86, Feb. 2003.

[85] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[86] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust management framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2002, pp. 114–130.

[87] H. Zhou and S. Foley, "A framework for establishing decentralized secure coalitions," in *19th IEEE Computer Security Foundations Workshop*, 2006, pp. 13 pp.–282.

[88] K. Feeney, D. Lewis, and D.O'Sullivan, "Service oriented policy management for web-application frameworks," *IEEE Internet Computing Magazine*, vol. (13):6, pp. 39–47, 2009.

[89] K. Feeney, R. Brennan, and S. N. Foley, "A trust model for capability delegation in federated policy systems," in *International Conference on Network and Service Management*. IEEE, 2010, pp. 226–229.

[90] S. Abdi, "Integration of heterogeneous policies for trust management," in *2014 IEEE 38th Annual Computer Software and Applications Conference Workshops (COMPSACW)*. IEEE, 2014.

[91] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The description logic handbook: theory, implementation, and applications*. New York, NY, USA: Cambridge University Press, 2003.

[92] D. L. McGuinness and F. Van Harmelen, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 2004-03, p. 10, 2004.

[93] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml," World Wide Web Consortium, W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/SWRL

[94] N. F. Noy and D. L. Mcguinness, "Ontology development 101: A guide to creating your first ontology," Tech. Rep., 2001.

[95] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semant.*, vol. 5, no. 2, pp. 51–53, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

[96] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language*. Addison-wesley Reading, 1996, vol. 2.

[97] B. McBride, "Jena: A semantic web toolkit," *IEEE Internet computing*, vol. 6, no. 6, pp. 55–59, 2002.

[98] T. R. Gruber. (1992) What is an ontology? [Online]. Available: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[99] ——, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[100] D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, 2nd ed. Springer-Verlag New York, Inc., 2003.

[101] Y. Ding, D. Fensel, M. Klein, and B. Omelayenko, "The semantic web: yet another hip?" *Data & Knowledge Engineering*, vol. 41, no. 2, pp. 205–227, 2002.

[102] R. Meersman, "Semantic ontology tools in is design," in *Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, ser. ISMIS '99. London, UK, UK: Springer-Verlag, 1999, pp. 30–45. [Online]. Available: http://dl.acm.org/citation.cfm?id=646358.689943

[103] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: principles and methods," *Data & knowledge engineering*, vol. 25, no. 1, pp. 161–197, 1998.

[104] J. de Bruijn, "Using Ontologies - Enabling Knowledge Sharing and Reuse on the Semantic Web," DERI, Technical Report, 2003. [Online]. Available: http://www.debruijn.net/publications/DERI-TR-2003-10-29.pdf

[105] C. Coral, R. Francisco, and P. Mario, *Ontologies for Software Engineering and Software Technology*. Berlin, Heidelberg: Springer-Verlag, 2006.

[106] D. Vallet, M. Fernández, and P. Castells, "An ontology-based information retrieval model," in *The Semantic Web: Research and Applications*. Springer, 2005, pp. 455–470.

[107] P. Castells, M. Fernandez, and D. Vallet, "An adaptation of the vector-space model for ontology-based information retrieval," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 2, pp. 261–272, 2007.

[108] N. H. Anderson, "Foundations of information integration theory," 1981.

[109] J. D. Ullman, "Information integration using logical views," in *Database Theory—ICDT'97*. Springer, 1997, pp. 19–40.

[110] A. Farquhar, R. Fikes, W. Pratt, and J. Rice, "Collaborative ontology construction for information integration," Technical Report KSL-95-63, Stanford University Knowledge Systems Laboratory, Tech. Rep., 1995.

[111] F. T. Fonseca, M. J. Egenhofer, P. Agouris, and G. Câmara, "Using ontologies for integrated geographic information systems," *Transactions in GIS*, vol. 6, no. 3, pp. 231–257, 2002.

[112] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," *ACM Sigmod Record*, vol. 33, no. 4, pp. 65–70, 2004.

[113] V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg, "Ontology in information security: a useful theoretical foundation and methodological tool," in *Proceedings of the 2001 workshop on New security paradigms*. ACM, 2001, pp. 53–59.

[114] A. Herzog, N. Shahmehri, and C. Duma, "An ontology of information security," *International Journal of Information Security and Privacy (IJISP)*, vol. 1, no. 4, pp. 1–23, 2007.

[115] W. M. Fitzgerald and S. Foley, "Management of heterogeneous security access control configuration using an ontology engineering approach," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, ser. SafeConfig '10. New York, NY, USA: ACM, 2010, pp. 27–36. [Online]. Available: http://doi.acm.org/10.1145/1866898.1866903

[116] A. Masoumzadeh and J. B. Joshi, "OSNAC: An ontology-based access control model for social networking systems," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 751–759. [Online]. Available: http://d-scholarship.pitt.edu/6024/

[117] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham, "R owl bac: representing role based access control in

owl," in *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, 2008, pp. 73–82.

[118] W. M. Fitzgerald and S. N. Foley, "Aligning Semantic Web applications with network access controls," *Computer Standards & Interfaces*, vol. 33, no. 1, pp. 24–34, Jan. 2011.

[119] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. H. Winsborough, and B. Thuraisingham, "Role Based Access Control and OWL," in *Proceedings of the fourth OWL: Experiences and Directions Workshop*, 2008.

[120] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting attribute-based access control with ontologies," in *Proceedings of The First International Conference on Availability, Reliability and Security, 2006 (ARES 2006).*, 2006, pp. 8 pp.–.

[121] T.-Y. Chen, "Knowledge sharing in virtual enterprises via an ontology-based access control approach," *Computers in Industry*, vol. 59, no. 5, pp. 502 – 519, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361507001790

[122] S. Javanmardi, M. Amini, R. Jalili, and Y. GanjiSaffar, "Sbac: A semantic based access control model," in *11th Nordic Workshop on Secure IT-systems (NordSec'06), Linkping, Sweden*, vol. 22, 2006.

[123] M. A. Ehsan, M. Amini, and R. Jalili, "A semantic-based access control mechanism using semantic technologies," in *Proceedings of the 2nd international conference on Security of information and networks*, ser. SIN '09. New York, NY, USA: ACM, 2009, pp. 258–267. [Online]. Available: http://doi.acm.org/10.1145/1626195.1626259

[124] M. Knechtel and J. Hladik, "RBAC authorization decision with DL reasoning," 2008.

[125] T. Priebe, W. Dobmeier, C. Schläger, and N. Kamprath, "Supporting attribute-based access control in authorization and authentication infrastructures with ontologies," *Journal of Software*, vol. 2, no. 1, pp. 27–38, 2007.

[126] B. Shields and O. Molloy, "Using description logic and rules to determine xml access control," in *Proceedings of 18th International Workshop on Database and Expert Systems Applications (DEXA'07), 2007.* IEEE, 2007, pp. 718–724.

[127] C. Zhao, N. Heilili, S. Liu, and Z. Lin, "Representation and reasoning on RBAC: A description logic approach," in *Theoretical Aspects of Computing–ICTAC 2005*. Springer, 2005, pp. 381–393.

[128] V. D. S. Almendra and D. Schwabe, "Trust policies for semantic web repositories," in *Proceedings of 2nd International Semantic Web Policy Workshop (SWPW'06), at the 5th International Semantic Web Conference (ISWC)*, 2006, pp. 5–9.

[129] B. Thuraisingham, "Building trustworthy semantic webs," in *Proceedings of the 10th IEEE international conference on Information Reuse & Integration*, ser. IRI'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 455–457. [Online]. Available: http://dl.acm.org/citation.cfm?id=1689250.1689340

[130] A. Squicciarini, E. Bertino, E. Ferrari, and I. Ray, "Achieving privacy in trust negotiations with an ontology-based approach," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, no. 1, pp. 13–30, 2006.

[131] W. Nejdl, D. Olmedilla, M. Winslett, and C. C. Zhang, "Ontology-based policy specification and management," in *2nd European Semantic Web Conference (ESWC)*. Springer, 2005, pp. 290–302.

[132] B. Thuraisingham, "Building trustworthy semantic webs," in *Proceedings of 2009 IEEE International Conference on Information Reuse Integration (IRI '09)*, 2009, pp. xiii–xv.

[133] H. Yu, C. Jin, and H. Che, "A description logic for PKI trust domain modeling," in *Third International Conference on Information Technology and Applications (ICITA 2005)*, vol. 2, 2005, pp. 524–528.

[134] S. Farrar and D. T. Langendoen, "An owl-dl implementation of gold," in *Linguistic Modeling of Information and Markup Languages*. Springer, 2010, pp. 45–66.

[135] F. Baader, I. Horrocks, and U. Sattler, "Handbook of knowledge representation," *by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Elsevier*, pp. 135–180, 2008.

[136] R. M. Smullyan, *First-order logic*. Courier Dover Publications, 1995.

[137] F. Baader, I. Horrocks, and U. Sattler, "Description logics as ontology languages for the semantic web," in *Festschrift in honor of Jörg Siekmann, Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2003, pp. 228–248.

[138] R. van Glabbeek. The undecidability of first order logic. [Online]. Available: http://kilby.stanford.edu/~rvg/154/handouts/fol.html

[139] D. Nardi and R. J. Brachman, "An introduction to description logics." in *Description Logic Handbook*, 2003, pp. 1–40.

[140] J. Hebeler, M. Fisher, R. Blace, and A. Perez-Lopez, *Semantic web programming*. John Wiley & Sons, 2011.

[141] M. O Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen, "Supporting rule system interoperability on the semantic web with SWRL," in *Proceedings of the 4th international conference on The Semantic Web*, ser. ISWC 05. Springer-Verlag, 2005, pp. 974–986.

[142] G. Wagner, "How to design a general rule markup language," in *In Invited talk at the Workshop XML Technologien für das Semantic Web (XSW 2002*, 2002, pp. 24–25.

[143] M. O'connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen, "Supporting rule system interoperability on the semantic web with swrl," in *The Semantic Web–ISWC 2005*. Springer, 2005, pp. 974–986.

[144] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining owl and ruleml," *W3C Member submission*, vol. 21, p. 79, 2004.

[145] M. J. O'Connor and A. K. Das, "SQWRL: A query language for OWL." in *OWLED*, vol. 529, 2009.

[146] D. H. Fudholi, N. Maneerat, R. Varakulsiripunth, and Y. Kato, "Application of protégé, swrl and sqwrl in fuzzy ontology-based menu recommendation," in *Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on*. IEEE, 2009, pp. 631–634.

[147] S. Abdi, "An autonomic trust management framework for secure dynamic coalition cooperation," in *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*. IEEE, 2013, pp. 422–429.

[148] E. Friedman-Hill *et al.*, "Jess, the rule engine for the java platform," 2008.

[149] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The role of trust management in distributed systems security," in *Secure Internet Programming*. Springer, 1999, pp. 185–210.

[150] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0," Tech. Rep., Mar. 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[151] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, 2002.

[152] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting attribute-based access control with ontologies," in *he First International Conference on Availability, Reliability and Security (ARES 2006)*. IEEE, 2006, pp. 8–pp.

[153] L. Kagal, I. Jacobi, and A. Khandelwal, "Gasping for AIR why we need linked rules and justifications on the semantic web," 2011. [Online]. Available: http://dspace.mit.edu/handle/1721.1/62294

[154] S. Schlobach, Z. Huang, R. Cornet, and F. Van Harmelen, "Debugging incoherent terminologies," *Journal of Automated Reasoning*, vol. 39, no. 3, pp. 317–349, 2007.

[155] E. Zolin, "Complexity of reasoning in description logics," 2013, http://www.cs.man.ac.uk/~ezolin/dl/.

[156] S. Abdi, "I was confused: Robust accountability for permission delegation in cloud federations," in *2014 IEEE 38th Annual Computer Software and Applications Conference Workshops (COMPSACW)*. IEEE, 2014.

[157] [Online]. Available: http://news.dunkinbrands.com/

[158] S. M. Shalabi, C. L. Doll, J. D. Reilly, and M. B. Shore, "Access control list," Dec. 5 2011, US Patent App. 13/311,278.

[159] J. Qian, S. Hinrichs, and K. Nahrstedt, "ACLA: A framework for access control list (acl) analysis and optimization," in *Communications and Multimedia Security Issues of the New Century*. Springer, 2001, pp. 197–211.

[160] S. Mäki, T. Aura, and M. Hietalahti, "Robust membership management for ad-hoc groups," in *Proc. 5Th Nordic Woekshop On Secure IT Systems (NORDSEC 2000), Reyjavic*, 2000.

[161] T. Aura and S. Mäki, "Towards a survivable security architecture for ad-hoc networks," in *Security Protocols*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, vol. 2467, pp. 63–73. [Online]. Available: http://dx.doi.org/10.1007/3-540-45807-7_9

[162] J. Rubio-Loyola, C. Merida-Campos, S. Willmott, A. Astorga, J. Serrat, and A. Galis, "Service coalitions for future internet services," in *Communications, 2009. ICC '09. IEEE International Conference on*, 2009, pp. 1–6.

[163] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands, "Models for coalition-based access control (CBAC)," in *Proceedings of the seventh ACM symposium on Access control models and technologies*, ser. SACMAT '02. New York, NY, USA: ACM, 2002, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/507711.507727

[164] B. Gleeson, G. Armitage, and J. Heinanen, "A framework for IP based virtual private networks," 2000. [Online]. Available: https://tools.ietf.org/html/rfc2764

[165] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: theory and practice," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 265–310, Nov. 1992. [Online]. Available: http://doi.acm.org/10.1145/138873.138874

[166] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[167] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[168] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments." *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[169] S. N. Foley, W. M. Fitzgerald, and W. Adams, "Federated autonomic network access control," in *Proceedings of 4th Symposium on Configuration Analytics and Automation (SAFECONFIG)*. IEEE, 2011, pp. 1–2.

Appendices

Appendix A

List of Abbreviations and Symbols

Abbreviations

ACL	Access Control List
ALC	Attribute Concept Language
ABox	Assertional Axioms
CBAC	Coalition-based Access Control
CWA	Closed World Assumption
DAL	Distributed Authorization Language
DL	Description Logic
FACNAC	Federated Autonomic Configuration for Network Access Controls
FOL	First Order Logic
ISC	Internet Services of Coalitions
OWA	Open World Assumption
OWL	Web Ontology Language
OWL-DL	Sub-pieces of OWL with maximum expressiveness while retains decidable
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RDF	Resource Description Framework
RT	Role-based Trust management

SDSI	Simple Distributed Security Infrastructure
SPKI	Simple Public Key Infrastructure
SSAL	Subterfuge Safe Authorization Language
SSAL^O	An Ontology for Subterfuge Safe Authorization Language
SSTM	Subterfuge Safe Trust Management
SQWRL	Semantic Query Web Rule Language
SWRL	Semantic Web Rule Language
TBox	Terminological Axioms
VPN	Virtual Private Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Symbols

$P \xrightarrow{Perm} Q$	This is used in RT and represents that principal P *delegates* permission $Perm$ to principal Q
$P \ni X$	principal P *holds* permission X
$P \longrightarrow Q$	principal Q *speaks for* principal P
$P \overset{X}{\Longrightarrow} Q$	principal P *delegates* permission X to principal Q
$P \rhd X$	principal P is *accountable* for permission X in the delegation chain
$X \rightsquigarrow Y$	localPermission Y is *no less authoritative than* localPermission X
$(P\ N)$	local name N in the name space of principal P
$\langle P\ Perm \rangle$	locally defined permission $Perm$ in the name space of principal P
$A \twoheadrightarrow B : \{a, b, c, ...\}_{s_A}$	The entity A sends a message to entity B. The message $\{a, b, c, ...\}$ is signed by the sender (s_A)

Appendix B

Proof of properties of SSAL Logic

B.1 Proof for Property 1

Theorem

Assuming principals P, Q, R, and permissions X, Y, Z, then when P delegates permission X to Q and Q delegates permission Y to R, we can infer that the greatest lower bound of X and Y is delegated to R. In the following Z is the lower bound of permissions X and Y and is dominated by the greatest lower bound of X and Y:

$$((P \overset{X}{\Longrightarrow} Q) \wedge (Q \overset{Y}{\Longrightarrow} R) \wedge (Z \rightsquigarrow X) \wedge (Z \rightsquigarrow Y)) \Rightarrow (P \overset{Z}{\Longrightarrow} R) \quad (1)$$

Proof

Axiom P5:

$$\frac{Z \rightsquigarrow X; Z \rightsquigarrow Y}{Z \rightsquigarrow (X \sqcap Y)}$$

From axiom P5 we have:

$$((Z \rightsquigarrow X) \wedge (Z \rightsquigarrow Y)) \Rightarrow (Z \rightsquigarrow (X \sqcap Y)) \quad (\mathrm{I})$$

Axiom D4:

$$\frac{P \overset{X}{\Longrightarrow} Q; Q \overset{Y}{\Longrightarrow} R}{P \overset{X \sqcap Y}{\Longrightarrow} R}$$

From D4 we have:

$$((P \overset{X}{\Longrightarrow} Q) \wedge (Q \overset{Y}{\Longrightarrow} R)) \Rightarrow (P \overset{X \sqcap Y}{\Longrightarrow} R) \quad \text{(II)}$$

Axiom D3:

$$\frac{P \overset{Y}{\Longrightarrow} Q; X \rightsquigarrow Y}{P \overset{X}{\Longrightarrow} Q}$$

From D3 we have:

$$((P \overset{X \sqcap Y}{\Longrightarrow} R) \wedge (Z \rightsquigarrow (X \sqcap Y)) \Rightarrow (P \overset{Z}{\Longrightarrow} R) \quad \text{(IV)}$$

From (I), (II) and (IV) we have:

$$((P \overset{X}{\Longrightarrow} Q) \wedge (Q \overset{Y}{\Longrightarrow} R) \wedge (Z \rightsquigarrow X) \wedge (Z \rightsquigarrow Y)) \Rightarrow (P \overset{Z}{\Longrightarrow} R) \quad (1)$$

B.2 Proof for Property 2

Theorem

If the delegator holds some permission then the delegatee also holds any dominated permission.

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad (2)$$

Proof

Axiom H2:

$$\frac{P \ni Y, P \overset{Y}{\Longrightarrow} Q}{Q \ni Y}$$

Axiom H4:

$$\frac{Q \ni Y; X \rightsquigarrow Y}{Q \ni X}$$

From H2 we have:

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q)) \Rightarrow (Q \ni Y) \quad \text{(I)}$$

From H4 we have:

$$((Q \ni Y) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad \text{(II)}$$

From (I) and (II) we have:

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad (2)$$

B.3 Proof for Property 3

Theorem

If a principal holds a permission, Y, and delegates it to Q then any principal that *speaks for* the delegatee also holds all permissions that are dominated by Y.

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y) \wedge (Q \longrightarrow R)) \Rightarrow (R \ni X) \quad (3)$$

Proof

Property 2:

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad (2)$$

Axiom H3:

$$\frac{P \ni X; P \longrightarrow Q}{Q \ni X}$$

From axiom H3 we have:

$$((Q \ni X) \wedge (Q \longrightarrow R)) \Rightarrow (R \ni X) \quad (\mathrm{I})$$

From property (2) and (I) we infer:

$$((P \ni Y) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y) \wedge (Q \longrightarrow R)) \Rightarrow (R \ni X) \quad (3)$$

B.4 Proof for Property 4

Theorem

If a principal is being delegated a permission and the delegator holds any dominated permissions, the delegatee also holds the dominated permissions.

$$((P \ni X) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad (4)$$

Proof

Axiom D3:

$$\frac{P \overset{Y}{\Longrightarrow} Q; X \rightsquigarrow Y}{P \overset{X}{\Longrightarrow} Q}$$

Axiom H2:

$$\frac{P \ni X; P \overset{X}{\Longrightarrow} Q}{Q \ni X}$$

From axiom D3 we have:

$$((P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (P \overset{X}{\Longrightarrow} Q) \quad \text{(I)}$$

From axiom H2 we have :

$$((P \overset{X}{\Longrightarrow} Q) \wedge (P \ni X)) \Rightarrow (Q \ni X) \quad \text{(II)}$$

From (I) and (II) we have:

$$((P \ni X) \wedge (P \overset{Y}{\Longrightarrow} Q) \wedge (X \rightsquigarrow Y)) \Rightarrow (Q \ni X) \quad (4)$$

B.5 Proof for Property 5

Theorem

Given principal P and Q, if Q *speaks for* P ($P \rightarrow Q$) then any permission N that is originated by P ($\langle P\ N \rangle$) is also a valid permission in Q's name space ($\langle Q\ N \rangle$), and is dominated by $\langle P\ N \rangle$.

$$((P \ni \langle P\ N \rangle) \wedge (P \longrightarrow Q)) \Rightarrow (\langle Q\ N \rangle \rightsquigarrow \langle P\ N \rangle) \quad (5)$$

Proof

Axiom P3:

$$\frac{\langle P\ N \rangle \rightsquigarrow X; P \longrightarrow Q}{\langle Q\ N \rangle \rightsquigarrow X}$$

From P3 we have:

$$((\langle P\ N \rangle \rightsquigarrow X) \wedge (P \longrightarrow Q)) \Rightarrow \langle Q\ N \rangle \rightsquigarrow X \quad \text{(I)}$$

X is a permission originated by P. Therefore we replace X with $\langle P\ N\rangle$ in the statement (I):

$$((\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q)) \Rightarrow (\langle Q\ N\rangle \rightsquigarrow \langle P\ N\rangle) \quad \text{(II)}$$

Lemma

The permission ordering $\rightsquigarrow$ is reflexive if and only if a permission is originated by a principal:

$$(P \ni X) \Longleftrightarrow (X \rightsquigarrow X) \quad \text{(L1)}$$

Proof for Lemma Axiom H1:

$$\frac{\{\!|N, \{\!|N|\!\}_{sK}|\!\}_{sK}}{K \ni \langle K\ N\rangle}$$

From H1 we infer that if there exists a permission $\langle K\ N\rangle$, some principal originated the $\langle K\ N\rangle$.

Therefore, From L1 we have:

$$(P \ni \langle P\ N\rangle) \Longleftrightarrow (\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle) \quad \text{(III)}$$

From (II) and (III) we infer that $\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle$ is always true if P has originated $\langle P\ N\rangle$ denoted as $(P \ni \langle P\ N\rangle)$. Consequently, we rewrite the statement (II) as:

$$((P \ni \langle P\ N\rangle) \wedge (P \longrightarrow Q)) \Rightarrow (\langle Q\ N\rangle \rightsquigarrow \langle P\ N\rangle) \quad (5)$$

B.6 Proof for Property 6

Theorem

Permission global ordering relation *No less authoritative than* denoted as $\rightsquigarrow$, is transitive.

$$((X \rightsquigarrow Y) \wedge (Y \rightsquigarrow Z)) \Rightarrow (X \rightsquigarrow Z) \quad (6)$$

Proof

Axiom P4:

$$\frac{X \rightsquigarrow \langle P\ N\rangle; P \longrightarrow Q; \langle Q\ N\rangle \rightsquigarrow Y; Q \rhd \langle P\ N\rangle}{X \rightsquigarrow Y}$$

The *speaks for* relation is transitive. From axiom P4 and substituting Q for P, and Z for Y:

$$((X \rightsquigarrow \langle Q\ N\rangle) \wedge (Q \longrightarrow Q) \wedge (\langle Q\ N\rangle \rightsquigarrow Z) \wedge (Q \rhd \langle Q\ N\rangle)) \Rightarrow (X \rightsquigarrow Z) \quad \text{(I)}$$

The statement $Q \rightarrow Q$ is always true, therefore statement (I) is rewritten as:

$$((X \rightsquigarrow \langle Q\ N\rangle) \wedge (\langle Q\ N\rangle \rightsquigarrow Z) \wedge (Q \rhd \langle Q\ N\rangle)) \Rightarrow (X \rightsquigarrow Z) \quad \text{(II)}$$

Axiom A5 (a principal which is accountable for a localPermission also holds it):

$$\frac{R \rhd X}{R \ni X}$$

From A5 substituiting Q for R and $\langle Q\ N\rangle$ for X we have:

$$(Q \rhd X) \Rightarrow (R \ni \langle Q\ N\rangle) \quad \text{(III)}$$

From (II) and (III) we have:

$$((X \rightsquigarrow \langle Q\ N\rangle) \wedge (\langle Q\ N\rangle \rightsquigarrow Z) \wedge (Q \ni \langle Q\ N\rangle)) \Rightarrow (X \rightsquigarrow Z) \quad \text{(III)}$$

If $\langle Q\ N\rangle$ is a properly defined permission, there is a principal that originated it and therefore holds it. The statement $(Q \ni \langle Q\ N\rangle)$ is a always true.

Finally, by rewriting $\langle Q\ N\rangle$ as Y and eliminating the true statement $Q \ni \langle Q\ N\rangle$ we can infer transitivity of permission ordering $\rightsquigarrow$:

$$((X \rightsquigarrow Y) \wedge (Y \rightsquigarrow Z)) \Rightarrow (X \rightsquigarrow Z) \quad (6)$$

B.7 Proof for Property 7

Theorem

If a principal Q accepts accountability for permission N in the name space of a principal P, and principal Q also *speaks for* P, any permission, X, that dominates permission $\langle Q\ N\rangle$, also dominates the permission $\langle P\ N\rangle$:

$$((\langle Q\ N\rangle \rightsquigarrow X) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow X) \quad (7)$$

Proof

Axiom P4:

$$\frac{X \rightsquigarrow \langle P\ N\rangle; P \longrightarrow Q; \langle Q\ N\rangle \rightsquigarrow Y; Q \rhd \langle P\ N\rangle}{X \rightsquigarrow Y}$$

From axiom P4, substituting $\langle P\ N\rangle$ for X and X for Y we have:

$$((\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (\langle Q\ N\rangle \rightsquigarrow X) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow X) \quad \text{(I)}$$

The statement $\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle$ is true from lemma L1, and also $P \ni \langle P\ N\rangle$ is implicitly stated in the statement $Q \rhd \langle P\ N\rangle$. Permission $\langle P\ N\rangle$ is defined by principal P and that P as the originator holds it ($P \ni \langle P\ N\rangle$). Therefore we have:

$$((\langle Q\ N\rangle \rightsquigarrow X) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow X) \quad (7)$$

B.8 Proof for Property 8

Theorem

If a principal Q accepts accountability for the permission N in the name space of the principal P ($\langle P\ N\rangle$), and Q speaks for P , any permission that is dominated by permission N in P's name space ($\langle P\ N\rangle$), is also dominated by the same permission N in the name space of Q ($\langle Q\ N\rangle$).

$$((X \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle Q\ N\rangle)) \Rightarrow (X \rightsquigarrow \langle Q\ N\rangle) \quad (8)$$

Proof

Axiom P4:

$$\frac{X \rightsquigarrow \langle P\ N\rangle; P \longrightarrow Q; \langle Q\ N\rangle \rightsquigarrow Y; Q \rhd \langle P\ N\rangle}{X \rightsquigarrow Y}$$

From axiom P4 and substituting $\langle Q\ N\rangle$ for Y we have:

$$((X \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (\langle Q\ N\rangle \rightsquigarrow \langle Q\ N\rangle) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (X \rightsquigarrow \langle Q\ N\rangle) \quad \text{(I)}$$

Axiom A4:

$$\frac{R \rhd \langle P\ N\rangle; P \longrightarrow Q}{R \rhd \langle Q\ N\rangle}$$

From A4 and substituting Q for R we have:

$$((Q \rhd \langle P\ N\rangle) \wedge (P \longrightarrow Q)) \Rightarrow (Q \rhd \langle Q\ N\rangle) \quad \text{(II)}$$

From (I) and (II) we have:

$$((X \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (\langle Q\ N\rangle \rightsquigarrow \langle Q\ N\rangle) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (X \rightsquigarrow \langle Q\ N\rangle)$$

From lemma L1 the statement $\langle Q\ N\rangle \rightsquigarrow \langle Q\ N\rangle$ is always true, therefore we have:

$$((X \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle Q\ N\rangle)) \Rightarrow (X \rightsquigarrow \langle Q\ N\rangle) \quad (8)$$

B.9 Proof for Property 9

Theorem

If principal Q *speaks for* P and is accountable for permission $\langle P\ N\rangle$, then the permission $\langle Q\ N\rangle$ dominates permission $\langle P\ N\rangle$:

$$((P \longrightarrow Q) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow \langle Q\ N\rangle) \quad (9)$$

Proof

From property 8 and substitute $\langle P\ N\rangle$ for X we have:

$$((\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle) \wedge (P \longrightarrow Q) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow \langle Q\ N\rangle) \quad \text{(I)}$$

From lemma L1 the statement $\langle P\ N\rangle \rightsquigarrow \langle P\ N\rangle$ is always true. Therefore, we have:

$$((P \longrightarrow Q) \wedge (Q \rhd \langle P\ N\rangle)) \Rightarrow (\langle P\ N\rangle \rightsquigarrow \langle Q\ N\rangle) \quad (9)$$